О.В.Бартеньев

ФОРТРАН ДЛЯ ПРОФЕССИОНАЛОВ. Математическая библиотека IMSL. Выпуск 1

Излагаются средства математической библиотеки IMSL, входящей в состав профессиональных версий Фортрана фирм Microsoft и Compaq, позволяющие анализировать линейные модели и выполнять матричные вычисления. Приводятся алгоритмы, положенные в основу процедур библиотеки, а также варианты их программных реализаций. Поясняются все используемые при описании процедур библиотеки понятия. Излагаемый материал иллюстрируется большим числом примеров.

Предназначено для научных работников, инженеров, преподавателей, студентов и аспирантов вузов.

Содержание

ПРЕДИСЛОВИЕ 3
1. ХАРАКТЕРИСТИКИ ПРОЦЕДУР IMSL 5
1.1. Состав библиотеки 5
1.2. Вызов процедур IMSL 6
1.3. Выделение памяти 10
1.4. Соглашения об именах 10
1.5. Вычислительная сложность алгоритмов 11
1.6. Оптимизация кода 12
1.7. Учет особенностей машинной арифметики 13
1.7.1. Разрывы между вещественными числами 13
1.7.2. Ошибки округления 14
1.7.3. Устранение переполнения и исчезновения порядка 15
1.7.4. Оценка точности результата 16
1.8. Обработка ошибок 18
1.9. Соглашения, действующие при описании процедур библиотеки IMSL 18
1.10. Вспомогательная функция ru_doswin 19
2. ВЫЧИСЛЕНИЯ С ПЛАВАЮЩЕЙ ТОЧКОЙ 20
2.1. Двоичные представления чисел 20
2.1.1.Представлениецелых чисел 20
2.1.2. Вещественные числа с плавающей точкой 20
2.1.3. Виды вещественных с плавающей точкой 22
2.1.4. Характеристики модели представления чисел 23
2.1.5.Абсолютная и относительная ошибки 26
2.2. Исключения 27
2.3. Управление вычислениями с плавающей точкой 28
2.3.1. Значения статуса и управляющего слова 28
2.3.2. Чтение статуса 29
2.3.3. Изменение управляющего слова 30
2.4. Обработка исключений 31
2.4.1. Обработка исключений с плавающей точкой 31
2.4.2. Обработка математических исключений 33

3. ВСПОМОГАТЕЛЬНЫЕ ПРОЦЕДУРЫ 36
3.1. Машинные константы 36
3.1.1. Целые машинные константы 36
3.1.2. Вещественные машинные константы 37
3.1.3. Проверка на NAN 38
3.2. Вывод результатов 39
3.2.1. Список, вызовы и параметры процедур 39
3.2.2. Вывод прямоугольной матрицы с нумерацией строк и столбцов 41
3.2.3. Вывод прямоугольной матрицы по заданному формату и с заданными именами строк и столбцов 43
3.2.4. Управление выводом 46
3.2.5. Управление размером страницы 53
3.2.6. Задание номера устройства В/В 54
3.3. Обработка ошибок 55
3.3.1. Виды ошибокIMSL 55
3.3.2. Примеры ошибокIMSL 56
3.3.3. Изменение характера реагирования на информационные ошибки 57
3.3.4. Использование системы обработки ошибок 59
3.4. Процедуры даты и времени 60
3.5. Задание и чтение настроек процедур IMSL 64
3.6. Некоторые специальные процедур ы 67
3.6.1. Разложение числа на простые множители 67
3.6.2. Дополнительные символьные функции 68
3.6.3. Вычисление SQRT(a**2+b**2) без исчезновения порядка и переполнения 70
3.6.4.Математическиеифизическиеконстанты 71
3.6.5.Переводвеличинизоднойсистемыединицвдругую 73
3.7.ВерсиябиблиотекиIMSL 75
4. БАЗОВЫЕ ПРОЦЕДУРЫ ЛИНЕЙНОЙ АЛГЕБРЫ 76
4.1. Введение 76
4.2. Некоторые сведения о векторах и матрицах 76
4.2.1. Обозначение векторов и матриц 76
4.2.2. Операции над векторами 78
4.2.3. Операции над матрицами 79
4.2.4. Элементарные преобразования 80
4.2.5. Определитель и миноры матрицы 80
4.2.6. Виды матриц 82
4.2.7. Умножение матриц 85
4.2.8. Умножение матрицы на вектор 86
4.2.9. Блочные матрицы 87
4.2.10. Двумерные матрицы вращения и отражения 88
4.2.11. Линейная независимость и базис 89
4.2.12. Ортогональность векторов 90

4.2.13. Ранг матрицы 90
4.2.14. Векторные нормы 91
4.2.15. Матричные нормы 92
4.2.16. Нормы и ортогональные преобразования 93
4.3. Представление матриц в программах 93
4.3.1. Представление всех элементов матрицы 94
4.3.2. Ленточные матрицы 95
4.3.3. Ленточные симметрические матрицы 95
4.3.4. Ленточные эрмитовы матрицы 96
4.3.5. Ленточные треугольные матрицы 97
4.3.6. Кодиагональное представление ленточных симметрических матриц 97
4.3.7. Кодиагональное представление ленточных эрмитовых матриц 98
4.3.8. Разреженные матрицы 99
4.4. Базовые процедуры линейной алгебры уровня 1 101
4.4.1. Замечания для программиста 101
4.4.2. Список процедур уровня 1 103
4.4.3. Присваивание вектору скаляра 104
4.4.4. Копирование вектора 105
4.4.5. Масштабирование вектора 106
4.4.6. Умножение вектора на скаляр 106
4.4.7. Сумма вектора и скаляра 106
4.4.8. Вычитание из скаляра вектора 107
4.4.9. Присваивание вида $y = ax + y$ 107
4.4.10. Обмен данными векторов 107
4.4.11. Скалярное произведение векторов 108
4.4.12. Скалярное произведение векторов с аккумулятором двойной точности 108
4.4.13. Сумма скаляра и скалярного произведения векторов с аккумулятором двойной точности 109
4.4.14. Скалярное произведение векторов с аккумулятором двойной точности, передаваемым в качестве параметра 109
4.4.15. Произведение Адамара 110
4.4.16. Сумма произведений трех векторов 110
4.4.17. Сумма элементов вектора 111
4.4.18. Сумма абсолютных значений элементов вектора 111
4.4.19. Евклидова, или 2-норма вектора 112
4.4.20. Произведение элементов вектора 112
4.4.21. Индекс минимального элемента вектора 112
4.4.22. Индекс максимального элемента вектора 112
4.4.23. Индекс минимального по модулю элемента вектора 112
4.4.24. Индекс максимального по модулю элемента вектора 113
4.4.25. Построение плоского вращения Гивенса 113
4.4.26. Применение плоского вращения Гивенса 114

4.4.27. Построение быстрого вращения Гивенса 115
4.4.28. Применение быстрого вращения Гивенса !15
4.5. Базовые процедуры линейной алгебры уровней 2 и 3 116
4.5.1. Замечания для программиста 116
4.5.2. Список процедур уровней 2 и 3 118
4.5.3. Умножение матрицы общего вида на вектор 119
4.5.4. Умножение ленточной матрицы на вектор 120
4.5.5. Умножение эрмитовой матрицы на вектор 121
4.5.6. Умножение эрмитовой ленточной матрицы на вектор 121
4.5.7. Умножение вещественной симметрической матрицы на вектор 121
4.5.8. Умножение симметрической ленточной матрицы на вектор 121
4.5.9. Умножение треугольной матрицы на вектор 121
4.5.10. Умножение треугольной ленточной матрицы на вектор 122
4.5.11. Умножение обратной треугольной матрицы на вектор 122
4.5.12. Умножение обратной треугольной ленточной матрицы на вектор 122
4.5.13. Преобразования ранга 1 матрицы общего вида 122
4.5.14. Преобразования ранга 1, 2, k и 2k симметрической матрицы 123
4.5.15. Преобразования ранга 1, 2, k и 2k эрмитовой матрицы 125
4.5.16. Произведение матриц общего вида 126
4.5.17. Произведение с симметрической матрицей 127
4.5.18. Произведение с эрмитовой матрицей 127
4.5.19. Произведение с треугольной матрицей 127
4.5.20. Вычисления с треугольной матрицей 128
4.6. Дополнительные операции над векторами и матрицами 128
4.6.1. Процедуры, которые могут быть заменены встроенными возможностями Фортрана 128
4.6.2. Преобразование матриц 129
4.6.3. Билинейная и квадратичная формы 135
4.6.4. Оценка матричного степенного ряда 136
4.6.5. Умножение ленточной матрицы на вектор 137
4.6.6. Сложение ленточных матриц 138
4.6.7. Вычисление нормы матрицы 140
4.6.8. Вычисление расстояния между точками 142
4.6.9. Вычисления с повышенной точностью 143
5. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ 145
5.1. Постановка задачи 145
5.2. Метод исключения Гаусса 146
5.2.1. Идея метода исключения 146
5.2.2. LU-разложение матрицы 149
5.2.3. Повышение качества разложения 153
5.2.4. Устойчивость метода исключений Гаусса 159
5.2.5. Некоторые приложения метода исключений Гаусса 159
5.2.6. Число обусловленности матрицы 161

5.2.7. Оценка точности и уточнение решения 165
5.2.8. Системы с разреженными матрицами 166
5.3. Линейные системы с симметрической матрицей 167
5.5.7. Положительно определенные системы 167
5.3.2. Получение разложения Холецкого 168
5.3.3. Неотрицательно определенные системы 171
5.3.4. Симметрические неопределенные системы 176
6. ПРОЦЕДУРЫ IMSL ДЛЯ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ И ВЫЧИСЛЕНИЯ ОПРЕДЕЛИТЕЛЕЙ 181
6.1. Введение 181
6.2. Системы линейных уравнений с вещественной несимметрической матрицей 182
6.2.1. Список процедур с вещественной несимметрической матрицей 182
6.2.2. Возможные ошибки процедур 185
6.2.3. Параметры процедур с вещественными несимметрическими и комплексными неэрмитовыми матрицами 186
6.2.4. Процедуры с вещественной несимметрической общего вида матрицей 188
6.2.5. Пример описания процедуры с вещественной несимметрической ленточной матрицей 197
6.2.6. Процедуры с вещественной несимметрической трехдиагональной матрицей 198
6.2.7. Процедуры с вещественной треугольной матрицей 201
6.2.8. Процедуры с вещественной несимметрической разреженной матрицей 206
6.3.Процедуры с вещественной симметрической матрицей 211
6.3.1. Список процедур 211
6.3.2. Возможные ошибки процедур с вещественной симметрической матрицей 215
6.3.3. Параметры процедур с вещественной симметрической и комплексной эрмитовой матрицами 216
6.3.4. Процедуры с вещественной симметрической положительно определенной матрицей 216
6.3.5. Процедуры с вещественной симметрической положительно определенной ленточной матрицей 221
6.3.6. Процедуры с вещественной симметрической положительно определенной разреженной матрицей 227
6.3.7. Разложение Холецкого симметрической неотрицательно определенной матрицы 241
6.3.8. Разложение Холецкого симметрической неотрицательно определенной матрицы с применением симметричного выбора 243
6.3.9. Процедуры с вещественной симметрической неопределенной матрицей 247

6.3.10. Разложение симметрической матрицы с добавлением к ней диагональной матрицы 251
6.3.11. Комментарий 254
6.4. Системы линейных уравнений с вещественной матрицей Теплица 254
6.5. Системы линейных уравнений с комплексной матрицей 255
6.5.1. Процедуры с комплексной неэрмитовой матрицей 256
6.5.2. Процедуры с комплексной эрмитовой матрицей 263
6.6. Вычисление определителей 269
6.6.1. Список процедур 269
6.6.2. Параметры процедур 271
6.6.3. Описание подпрограммы LFDRG 271
6.6.4. Примеры вычисления определителей 272
7. МАТРИЧНЫЕ РАЗЛОЖЕНИЯ И ИХ ПЕРЕСЧЕТ 274
7.1. Введение 274
7.2. Преобразования Хаусхолдера иГивенса 275
7.2.1. Преобразование Хаусхолдера 275
7.2.2. Преобразование Гивенса 279
7.3. Методы вычисления QR-разложения 285
7.3.1. QR-разложение прямоугольной матрицы с применением преобразования Хаусхолдера 285
7.3.2. QR-разложение симметрической матрицы 288
7.4. QR-разложение процедурами IMSL 294
7.4.1. QR-разложение прямоугольной матрицы 294
7.4.2. Восстановление ортогональной матрицы Q из QR-разложения прямоугольной матрицы 296
7.5. Пересчет матричных разложений 297
7.5.1. Алгоритм пересчета QR-разложения 297
7.5.2. Перечень процедур 302
7.5.3. Пересчет разложения Холецкого 303
7.5.4. Пересчет QR-разложения 308
7.6. Вычисление сингулярного разложения 310
7.6.1. Понятие сингулярного разложения 310
7.6.2. Сингулярное разложение комплексной матрицы 312
7.6.3. Сингулярное разложение вещественной матрицы 315
7.6.4. Вычисление обобщенной обратной матрицы 316
8. СОБСТВЕННЫЕ ЗНАЧЕНИЯ И СОБСТВЕННЫЕ ВЕКТОРЫ 319
8.1. Подходы к решению проблемы собственных значений 319
8.1.1. Понятия собственного значения и собственного вектора 319
8.1.2. Преобразования подобия 322
8.1.3. Некоторые свойства собственных значений и собственных векторов 323
8.1.4. Собственные значения и собственные векторы вещественной симметрической и комплексной эрмитовой матрицы 324
8.2. Процедуры IMSL для вычисления собственных значений 330

8.2.1. Организация материала 330
8.2.2. Обычная проблема собственных значений 330
8.2.3. Обобщенная проблема собственных значений 346
8.3. Оценка точности решения проблемы собственных значений 352
8.3.1. Список, вызовы, параметры и ошибки процедур 352
9. РЕШЕНИЕ ПРЯМОУГОЛЬНЫХ СИСТЕМ $Ax \approx b$ с ОГРАНИЧЕНИЯМИ И БЕЗ НИХ 355
9.1. Метод наименьших квадратов 355
9.1.1. Постановка задачи 355
9.1.2. Применение QR-разложения в методе наименьших квадратов 356
9.1.3. Вырожденная задача наименьших квадратов 358
9.1.4. Решение задачи наименьших квадратов с помощью сингулярного разложения 361
9.2. Подпрограммы IMSL для решения переопределенной и недоопределенной проблемы $Ax \approx b$ 362
9.2.1. Перечень подпрограмм 362
9.2.2. Проблема $Ax \approx b$ без ограничений 363
9.3. Проблема $Ax \approx b$ с линейными ограничениями 372
10. ПРОЦЕДУРЫ И ОПЕРАЦИИ БИБЛИОТЕКИ IMSL FORTRAN 90 MP 375
10.1. Введение 375
10.2. Операции библиотеки IMSL 90 378
10.3. Функции библиотеки IMSL 90 379
10.4. Решение систем линейных уравнений 381
10.4.1. Процедура LIN_SOL_GEN для линейных систем с матрицей общего вида 381
10.4.2. Процедура LIN_SOL_SELF для линейных систем с самоприсоединенной матрицей 387
10.4.3. Процедура LIN_SOL_TRI для линейных систем с трехдиагональной матрицей 379
10.5. Сингулярные и собственные значения 394
10.5.1. Процедура L1N_SVD для вычисления сингулярного разложения 394
10.5.2. Процедура LIN_E1G_SELF для вычисления собственных значений самоприсоединенной матрицы 397
10.5.3. Процедура LIN_E1G_GEN для вычисления собственных значений матрицы общего вида 399
10.5.4. Процедура L1N_GEIG_GEN для обобщенной проблемы собственных значений 403
10.6. Решение линейных систем методом наименьших квадратов 406
10.6.1. Процедура LIN_SOL_LSQ для решения прямоугольных линейных систем методом наименьших квадратов 406
10.6.2. Процедура LIN_SOL_SVD, использующая сингулярное разложение для решения прямоугольных линейных систем 410

ПРИЛОЖЕНИЕ 1. Вывод русского текста в DOS-окно 413
ПРИЛОЖЕНИЕ 2. Список представленных в пособии процедур математической библиотеки IMSL 415
ЛИТЕРАТУРА 433
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 437

Предметный указатель

L
LU разложение
вычисление 150
применение 152
N
NaN См. Не число
Q
QR алгоритм 325
со сдвигом 327
QR разложение
в вырожденной задаче наименьших квадратов 358
в задаче наименьших квадратов 357
восстановление матрицы Q 292, 296
пересчет 298, 302
прямоугольной матрицы 294
с выбором главного столбца 358
симметрической трехдиагональной матрицы 290
А
Аккумулятор повышенной точности 101, 143
Алгоритм корректный 6
Б
Билинейная форма 135
В
Ведущее уравнение 147
Ведущий элемент 147
полный выбор 157
частичный выбор 148, 153
Вектор
Гаусса 150
единичный, нормированный 92
координаты 90
масштабирование 106
перестановок 155
столбец 76
строка 77
Хаусхолдера 275
Векторы
внешнее произведение 79
линейная комбинация 89
линейно зависимые 90
линейно независимые 90
скалярное произведение 78, 108
Вспомогательная процедура ru_doswin 413
Вспомогательный модуль text_transfer 413
Д
Двоичный порядок 21
З
Задача 16
И
Исчезновение порядка 15, 70
Итерационное уточнение решения 165, 385
К
Квадратичная форма 83, 136
Коэффициент роста 159, 167
Л
Лента матрицы
верхняя, нижняя 84
ширина 84
М
Мантисса 21
Матрица
g3 обратная 205
LU разложение 149
QR разложение 285
Адамара 194
балансировка 158

блочная 87
блочно столбцовая 77
блочно строковая 77
верхняя треугольная 84
виды разложений 274
вращения 88
вырожденная, невырожденная 83
Гивенса 89, 279
Гильберта 42
главные диагональные миноры 81
главные подматрицы 81
диагональная 82
диагонально доминирующая 158
единичная, тождественная 82
квадратичной формы 83
квазидиагональная 325
квазитреугольная 325
кодиагональ 84
ленточная 84, 95
ленточная симметрическая 95, 97
ленточная треугольная 97
ленточная эрмитова 96, 98
масштабирование 158
минор 81
неопределенная 176
неотрицательно определенная 171
нижняя треугольная 84
обобщенное обращение 204, 317
обратная 82
обращение 83
общего вида 82, 188
определитель 80
определитель произведения матриц 82
ортогональная 83
отражения 88
отрицательно определенная 168
персимметрическая 254
плохо обусловленная 161
подматрица 81
подобная 322
полная 94
полный столбцовый ранг 91
положительно определенная 167
преобразования Гаусса 150
преобразования подобия 322
произведение 80, 85, 126
простой структуры 323
прямоугольная 82
разложение Холецкого 168
разреженная 84, 99
ранг 90
расщепление 327
самоприсоединенная 378
симметрическая 83
сингулярное разложение 274
след 323
собственное значение 319
собственный вектор 319
степенной ряд 136
Теплица 254
трапециевидная 85
треугольная 201
умножение на вектор 86, 119, 137
унитарная 84
унитреугольная 85
характеристическая 319
характеристический определитель 319
характеристическое уравнение 319
Хаусхолдера 89, 275
Хессенберга 324
хорошо обусловленная 161
циклическая 257
число обусловленности 161, 189, 311
элементарные преобразования 80
эрмитова 83

Якоби 89
Машинная бесконечность 20
отрицательная, положительная 38
Машинная точность 24
Метод
наименьших квадратов 355
обратных итераций 328
отражений 276
с диагональным выбором 176
сопряженных градиентов 227
степенной 323
Хаусхолдера 289
Метод исключений Гаусса 146
вычисление обратной матрицы 159
вычисление определителя 160
обратный ход 146
прямой ход 146
устойчивость 159
Множитель k го шага 150
Модель 355
параметры 355
Н
Не число 22
сигнальный NaN 37
совершенный NaN 37
Невязка 147, 165
Норма вектора
1 , 2 , inf норма 91
2 норма, евклидова 15, 91,112
p норма 91
Норма матрицы
1 , 2 , inf норма 92, 140
p норма 92
Фробениуса 92
Нуль вектор 90
О
Обобщенное имя {G} 102
Обусловленность вычислительной задачи 16
Ошибка
завершающая 55
информационная 55
округления 14
тип или уровень 55
уровень серьезности 57
Ошибка округления 27
П
Параметры diag, side, trans, uplo 117
Параметры incx, incy 102
Переполнение 15, 70
Подстановки
обратные 152
прямые 152
Предотвращение конфликтов памяти 193
Преобразования
Гивенса 113,279
матриц 129
ортогональные 93
ранга 1 122
ранга 1,2, k и 2k 125
Преобразования Хаусхолдера 275
блочные 368
Проблема $Ax \approx b$
без ограничений 363
с Линейными ограничениями 372
Пространство
базис 90
нормированное 92
Процедура IMSL
второго уровня 10
главная 10
Р
Разложение неопределенной матрицы 176
Разложение Холецкого
вычисление 169
неотрицательно определенной матрицы 172
пересчет 302
С
Симметричный выбор 173
Сингулярное разложение 310
в задаче наименьших квадратов 361

Сингулярные векторы 310
Сингулярные значения 310
Система линейных уравнений 356, 145
    однородных 145
Собственные значения
    вычисление 324
    диагональной матрицы 323
    итерационные методы вычислений 322
    обобщенная проблема 321
    обычная проблема 321
    оценка точности вычисления 352
    полная проблема 321
    прямые методы вычислений 322
    свойства 323
    частичная проблема 321
Стандарт ШЕЕ 20, 37
Схема
    Горнера 12, 137
    частичного выбора 148
Т
Треугольники Холецкого 168
Ф
Флоп 11

ПРЕДИСЛОВИЕ

В настоящее время широко известны и распространены математические и статистические пакеты MATLAB, MATHCAD, STATISTICA и др., предоставляющие большой набор средств для решения различных математических задач. Однако специалисту, использующему в своей профессиональной деятельности Фортран, нет резона осваивать и переносить свои задачи в какой-либо иной пакет, поскольку в его распоряжении есть IMSL – математическая и статистическая библиотека фирмы Visual Numerics, Inc.

Библиотека IMSL является неотъемлемой частью профессиональных версий Fortran PowerStation (FPS) 4.0 и Compaq Visual Fortran (CVF). Фортран и IMSL, дополняя друг друга, образуют мощный инструмент для решения вычислительных задач и обработки данных. Процедуры IMSL, будучи реализованными на Фортране, естественным образом встраиваются в Фортран-приложения. Созданные с их присутствием программы можно употреблять для исследования моделей, выполнения расчетов, обработки и отображения данных, в автоматизированных системах проектирования и управления, т. е. в любой сфере, где проблема поддается формализации.

Большой объем библиотеки не позволяет дать ее описание в пределах одной книги. Поэтому в настоящем пособии рассматривается часть библиотеки – процедуры, позволяющие анализировать линейные модели (линейные системы, в том числе недоопределенные и переопределенные, собственные значения), выполнять векторные и матричные вычисления, пересчитывать матричные разложения и решать нелинейные уравнения и их системы. Кроме того, приводятся служебные процедуры IMSL и процедуры сортировки и поиска данных. Иные процедуры математической библиотеки будут изложены во второй части пособия.

Пособие содержит 3 блока материалов:

- сведения об особенностях машинной арифметики;
- основные понятия линейной алгебры и методы и алгоритмы, используемые процедурами IMSL;
- процедуры IMSL.

Назначение двух первых блоков – сделать пособие достаточным для восприятия процедур библиотеки и понимания заложенных в их основу механизмов. Большинство из приводимых в этих блоках алгоритмов сопровождаются примерами программной реализации. Кроме того, специалисту известно, что вычислительные задачи насыщены ловушками, преодоление которых без понимания численных методов невозможно. В то же время читателю, не имеющему достаточного опыта работы с линейными моделями, можно рекомендовать, например, учебные пособия Куроша А. Г., Вое-

водина В. В. или Голуба Дж. и Ван Лоана Ч. (см. список литературы в конце книги).

Поставляемая с CVF библиотека IMSL включает два вида процедур: отвечающие стандарту Фортран 77 (IMSL Fortran 77) и соответствующие стандарту Фортран 90 (IMSL Fortran 90 MP). В пособии приводятся процедуры и первого и второго вида. Более того, большая его часть отведена под процедуры IMSL Fortran 77. Это объясняется тем, что, во-первых, пользователям FPS 4.0 процедуры IMSL Fortran 90 MP недоступны (не входят в комплект поставки) и, во-вторых, библиотека IMSL Fortran 90 MP не обладает полнотой, присущей IMSL Fortran 77. Например, текущая версия IMSL Fortran 90 MP не содержит процедур для решения нелинейных уравнений и их систем, вычисления производных и квадратур, пересчета матричных разложений, в ней не реализованы многие базовые операции линейной алгебры, специальные математические функции и т. д. Тем не менее предпочтение следует отдать библиотеке IMSL Fortran 90 MP, если, конечно, задача решается имеющимися в ней средствами (более подробно о преимуществах IMSL Fortran 90 MP см. гл. 10).

Описание процедур IMSL максимально приближено к тексту, имеющемуся в поставляемой с Фортраном документации. Способы их употребления иллюстрируются примерами. Собственно процедурам IMSL предшествует обзор положенных в их основу алгоритмов. Полный перечень представленных в пособии процедур дан в прил. 2.

1. ХАРАКТЕРИСТИКИ ПРОЦЕДУР IMSL

1.1. СОСТАВ БИБЛИОТЕКИ

Библиотека IMSL Fortran 77 включает математические и статистические процедуры. Математическая библиотека разделена на две части: основную и содержащую специальные функции, не включенные в состав встроенных функций Фортрана. Описание процедур изложено в руководстве пользователя библиотеки.

Математическая библиотека (основная часть) содержит главы:

1) линейные системы;
2) анализ собственных значений;
3) интерполяция и аппроксимация;
4) интегрирование и дифференцирование;
5) преобразования;
6) нелинейные уравнения;
7) оптимизация;
8) базовые векторные и матричные операции;
9) утилиты.

Математическая библиотека (специальные функции):

1) элементарные, тригонометрические и гиперболические функции;
2) экспоненциальные интегралы и связанные с ними функции;
3) гамма-функции и связанные с ними функции;
4) функция ошибок и связанные с ней функции;
5) функции Бесселя, Кельвина, Эйри, Матье;
6) эллиптические интегралы;
7) эллиптические и связанные с ними функции;
8) функции распределения и обратные им функции;
9) различные функции.

Статистическая библиотека:

1) базовая статистика;
2) регрессия, корреляция, анализ дисперсии;
3) анализ категориальных и дискретных данных;
4) непараметрическая статистика;
5) тесты степени согласия и случайности;
6) анализ временных интервалов и предсказаний;
7) структуры ковариаций и факторный анализ;
8) дискриминантный анализ, кластерный анализ, выборки;
9) наработка на отказ, долговечность и надежность;
10) многомерное масштабирование;

11) оценка плотности и риска;
12) функции распределения вероятности и обратные им функции;
13) генерация случайных чисел;
14) утилиты, математическая поддержка, графика.

Замечание. Для компактности изложения вместо названия IMSL Fortran 77 будет употребляться аббревиатура IMSL. Исключение составляет 10-я глава пособия, содержащая описание процедур IMSL Fortran 90 MP. В ней для IMSL Fortran 77 применяется сокращение IMSL 77.

Большинство процедур имеют две модификации. Первая оперирует с данными одинарной, вторая – двойной точности. Процедуры ориентированы на программы с одной нитью. В случае использования нескольких нитей необходимо следить за тем, чтобы одновременно не выполнялись две или более процедуры IMSL. Активной может быть только одна процедура библиотеки. Техника создания многониточных приложений на Фортране изложена, например, в [3].

Процедуры библиотеки базируются на эффективных и корректных вычислительных алгоритмах. (Напомним, что алгоритм является корректным, если он удовлетворяет трем условиям: 1) позволяет за конечное число машинных операций преобразовать любые допустимые входные данные x в результат y; 2) результат y устойчив по отношению к малым возмущениям входных данных; 3) результат y обладает вычислительной устойчивостью. Обсуждение деталей, связанных с понятием корректности алгоритмов, можно найти в [1].) Каждый из применяемых алгоритмов тщательно исследован теоретически, а их программная реализация прошла всестороннюю проверку. В то же время при обнаружении неполадок соответствующее сообщение можно направить Visual Numerics по адресу
pcsupport@houston.vni.com

Дополнительную информацию о библиотеке IMSL можно получить по адресу
http://www.vni.com/tech/techsup.html

1.2. ВЫЗОВ ПРОЦЕДУР IMSL

Доступ к любой процедуре библиотеки IMSL (напомним, что речь идет о библиотеке IMSL Fortran 77) в среде FPS 4.0 и CVF станет возможным, если в программном компоненте, вызывающем библиотечную процедуру, выполнить ссылку

```
use msimsl
```

Файл msimsl.f90 содержит код с четырьмя ссылками:

```
module msimsl
  use msimslmd; use msimslms
  use msimslsd; use msimslss
end module msimsl
```

Модули, на которые ссылается файл msimsl.f90, содержат:

- msimslmd(s).f90 – интерфейсы процедур математической библиотеки, работающих с двойной (одинарной) точностью;
- msimslsd(s).f90 – интерфейсы процедур статистической библиотеки, работающих с двойной (одинарной) точностью.

Таким образом, если программа использует, например, только математические библиотеки с одинарной точностью, в ней достаточно разместить ссылку

```
use msimslms
```

Сокращение числа ссылок снизит время компиляции программы.

Каждый из перечисленных модулей в свою очередь имеет ссылку на модуль, содержащий интерфейсы общих для всех библиотек процедур (*use msimslc*). Модули msimslmd(s).f90 и msimslsd(s).f90 включают ссылку *use msimslcd(s)* на модуль, содержащий интерфейсы общих для всех библиотек процедур двойной (одинарной) точности. Кроме того, модуль msimslmd(s).f90 ссылается на модуль с интерфейсами общих математических процедур (*use msimslmc*), а модуль msimslsd(s).f90 – на модуль с интерфейсами общих статистических процедур (*use msimslsc*). Все перечисленные модули содержатся в директории INCLUDE папки MSDEV (в случае FPS) или в директории INCLUDE папки IMSL (в случае CVF).

Перечисленные файлы с интерфейсами включены в CVF для совместимости с FPS. Поэтому пользователю CVF более естественно вместо ссылки *use msimsl* употреблять ссылку

```
use dfimsl
```

Модуль dfimsl.f90 устроен по аналогии с модулем msimsl.f90, поэтому все сказанное относительно msimsl.f90 переносится на dfimsl.f90. В качестве альтернативы в CVF можно получить доступ к интерфейсам всех процедур IMSL, выполнив ссылку

```
use Numerical_Libraries
```

или

```
use imslf77
```

Файл Numerical_Libraries.f90 содержит интерфейсы всех процедур IMSL, написанные в стиле Фортрана 90. Например, интерфейс LSARG:

```
interface
 subroutine Lsarg(n, a, Lda, b, ipath, x)
  integer, intent(in) :: n, Lda
  real(kind(1e0)), dimension(Lda, *) :: a
  real(kind(1e0)), dimension(*) :: b, x
  integer :: ipath
 end subroutine
end interface
```

Применение встроенной функции KIND в операторах объявления обеспечивает переносимость процедур на любую платформу. Интерфейс той же процедуры в файле dfimslms.f90 или msimslms.f90

```
interface
  subroutine Lsarg(n, a, Lda, b, ipath, x)
  integer n, Lda, ipath
  real a(Lda, *), b(*), x(*)
 end subroutine
end interface
```

написан в стиле Фортрана 77.

Замечание. Для сохранения общности в приводимых примерах и программах используется ссылка *use msimsl*, которая работает и в FPS, и в CVF.

Кроме внесения в программу соответствующих ссылок на модули с интерфейсами процедур библиотеки IMSL, необходимо, чтобы компилятор видел директории, содержащие подключаемые модули. Как правило, эта видимость обеспечивается при установке Фортрана. Если же по какой-либо причине модули компилятором не обнаруживаются, то для обеспечения видимости в среде Microsoft Developer Visual Studio (VS) FPS 4.0 следует выполнить действия Tools – Options – Directories – в списке Show directories for выбрать Include files – New (рис. 1.1) – найти директорию с искомыми файлами – OK (рис. 1.2).

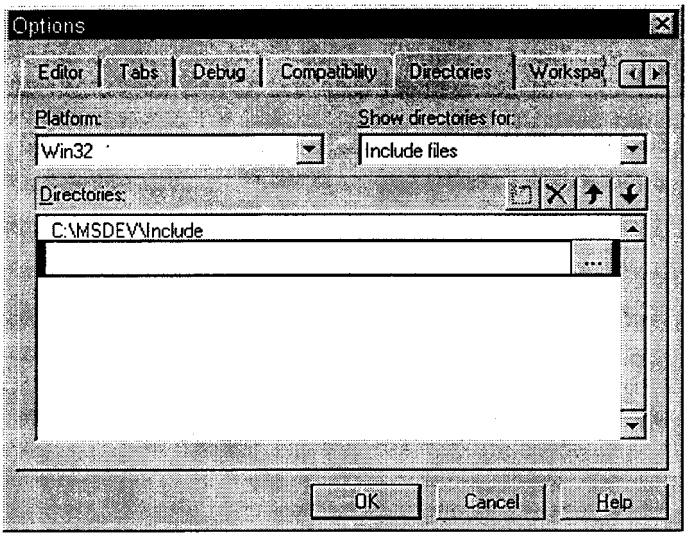

Рис. 1.1. Отображение директорий для INCLUDE-файлов

Также компилятор должен видеть файлы maths.lib, mathd.lib, stats.lib и statsd.lib, содержащие объектные коды процедур библиотеки. Связь с этими файлами устанавливается при инсталляции Фортрана (через переменную окружения LIB файла autoexec.bat). При необходимости директорию с библиотечными файлами можно явно указать в среде VS, применив цепочку Tools – Options – Directories – в списке Show directories for выбрать Library files – New – найти директорию с искомыми файлами – OK.

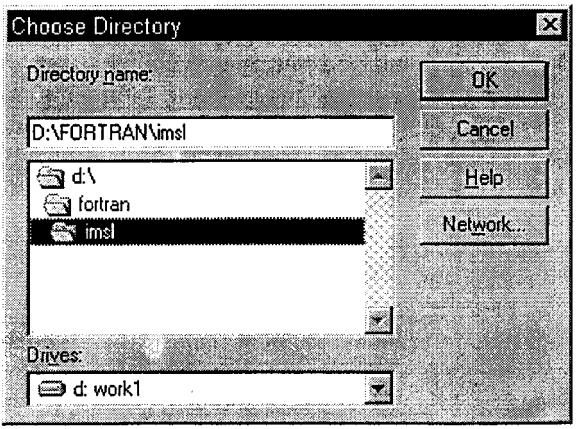

Рис. 1.2. Добавление новой директории с INCLUDE-файлами

Сами же библиотечные файлы можно затем явно указать построителю, применив цепочку Build – Settings – Link – выбрать тип проекта, например Release, – добавить, разделяя пробелами, имена библиотечных файлов в поле Object/library modules – OK.

Аналогичным образом организуется доступ к файлам библиотеки IMSL в среде CVF (см. также разд. 10.1).

1.3. ВЫДЕЛЕНИЕ ПАМЯТИ

Многим процедурам библиотеки IMSL для выполнения вычислений нужна дополнительная память, например для создания промежуточных данных. В IMSL выделение дополнительной памяти – одного или нескольких рабочих массивов – осуществляется в так называемой главной процедуре, например в LFCRG. Затем сформированные массивы передаются в процедуру 2-го уровня – L2CRG, в которой собственно и производятся вычисления. Перед выходом из LFCRG выделенная в ней память освобождается. Процедуру 2-го уровня можно вызвать непосредственно. Однако при этом в нее потребуется явно передать рабочие массивы. Так, L2CRG дополнительно по сравнению с LFCRG принимает рабочий вектор размера n. Если же вызывается процедура DL2CRG, оперирующая с двойной точностью, то размер рабочего вектора должен быть увеличен до $2n$.

Процедуры уровня 2, как правило, в пособии не приводятся. Исключения составляют случаи, когда процедуры второго уровня обладают дополнительными по сравнению с главными процедурами возможностями.

1.4. СОГЛАШЕНИЯ ОБ ИМЕНАХ

Имя процедуры IMSL состоит из корня и необязательного префикса. Имена процедур, оперирующих с вещественными данными одинарной точности и не входящих в базовые процедуры линейной алгебры, совпадают с их корнями, например LFCRG. В случае комплексных данных предпоследняя буква корня заменяется на C, например LFCCG. Имена процедур этого вида, работающих с двойной точностью, получаются путем добавления к корню префикса D, например DLFCRG или DLFCCG.

Целочисленные параметры процедур IMSL начинаются с букв, для которых в Фортране целый тип задается по умолчанию, т. е. с I, J, K, L, M и N. Остальные буквы используются для вещественных или комплексных параметров. Тип параметров зависит от типа данных, для обработки которых процедура предназначена. Существует определенная система именования параметров процедур IMSL. Например, имя LDA используется для задания ведущего измерения массива, представляющего матрицу A. Следует освоить

эту систему, с тем чтобы не присваивать присущие IMSL имена прочим объектам.

Порядок образования имен процедур базовой линейной алгебры и их параметров отражен в разд. 4.4.1.

Замечание. Работая с IMSL, для предотвращение конфликтов имен надо придерживаться очевидного правила: задавая имена констант, переменных, процедур, модулей и общих блоков, нельзя использовать имена процедур IMSL – как главных, так и процедур 2-го уровня. Правило легче выполнить, если отказаться от применения имен длиной в три или более символа, на второй или третьей позиции которых расположена цифра. Это позволит избежать конфликтов с именами процедур уровня 2.

1.5. ВЫЧИСЛИТЕЛЬНАЯ СЛОЖНОСТЬ АЛГОРИТМОВ

Оценку вычислительной сложности алгоритмов можно выполнять по числу арифметических операций, необходимых для получения результата. При работе с вещественными данными выполняются операции с плавающей точкой, каждую из которых принято называть *флопом* (от floating point operation). Таким образом, общая трудоемкость задачи оценивается числом выполненных при ее решении флопов.

Примеры:

1) сложение двух векторов размера n требует n флопов;
2) скалярное произведение двух векторов размера n требует $2n$ флопов (n умножений и n сложений);
3) в случае вложенных циклов суммируют обычно флопы наиболее глубоко вложенных операторов. Так, при умножении матриц ($\mathbf{R}^{m\times r}\times\mathbf{R}^{r\times n} \to \mathbf{R}^{m\times n}$) $C = AB \Rightarrow c_{ij} = \sum_{k=1}^{r} a_{ik}b_{ki}$ таковым является оператор $c(i,j) = c(i,j) + a(i,k) * b(k,j)$, который содержит 2 флопа и выполняется при умножении матриц общего вида mnr раз. То есть оценка трудоемкости такого умножения – $2mnr$ флопа. Оценка трудоемкости умножения треугольных $n\times n$-матриц – $n^3/3$ флопов.

Подсчет флопов, даже если он выполнен точно, даст достаточно грубую оценку эффективности алгоритма, поскольку не учитывает затраты на индексацию, обмены с памятью и другие возникающие при исполнении программы издержки. Кроме того, при параллельных вычислениях производительность программы существенно зависит от организации данных и модификации алгоритма, имеющие одинаковое число флопов, но разные структуры данных, могут серьезно различаться производительностью.

1.6. ОПТИМИЗАЦИЯ КОДА

Многие вычисления можно существенно ускорить и вдобавок повысить точность результата, если отказаться от вытекающей из постановки задачи последовательности операций. Пусть, например, для некоторого $x = x_0$ надо вычислить многочлен

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0.$$

Непосредственное вычисление многочлена требует выполнения $n^2 + n / 2$ умножений и n сложений. Кроме того, возведение в высокую степень может привести к переполнению или исчезновению порядка.

Число операций, необходимых для вычисления многочлена, можно снизить до n умножений и n сложений, применив схему Горнера:

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + xa_n) \ldots)).$$

Для ее реализации подойдет код

```
program Gorner
 integer(4), parameter :: ndeg = 4            ! Степень многочлена
 integer(4) :: i
 ! Массив coeff содержит коэффициенты многочлена P4(x) = x^4 - x^3 - 4x^2 + 34x - 120 = 0
 real(4) :: coeff(ndeg + 1) = (/ -120.0, 34.0, -4.0, -1.0, 1.0 /)
 ! x0 – значение аргумента, для которого вычисляется многочлен
 real(4) :: x0 = 1.2, p                       ! p – результат
 p = coeff(ndeg + 1)
 do i = ndeg, 1, -1
  p = coeff(i) + x0 * p
 end do
 print *, 'P(x0) = ', p                       ! P(x0) =     -84.614400
end program Gorner
```

Такая оптимизация не только снизит время вычислений, но и повысит точность результата, поскольку с ростом числа операций нередко за счет погрешностей округления увеличивается накапливаемая ошибка.

Большие возможности для оптимизации кода по быстродействию возникают при организации матричных вычислений. Процедуры IMSL реализуют эти возможности в полной мере, в частности за счет использования алгоритмов, реализованных в базовых процедурах линейной алгебры (гл. 4). Примеры оптимизации кода при выполнении матричных вычислений приводятся также в гл. 7 (см., например, разд. 7.2.1.3, 7.3.1 или 7.5.1).

1.7. УЧЕТ ОСОБЕННОСТЕЙ МАШИННОЙ АРИФМЕТИКИ

1.7.1. Разрывы между вещественными числами

Библиотечные процедуры разрабатываются с учетом особенностей машинной арифметики, которые связаны, во-первых, с неточностью представления вещественных и комплексных чисел и, во-вторых, с наличием ограниченных диапазонов их представления. Например, все представимые в ЭВМ вещественные числа типа REAL(4) расположены в диапазонах:

- отрицательные числа: от -3.402823E+38 до -1.175494E-38;
- число 0;
- положительные числа: от +1.175494E-38 до +3.402823E+38.

Таким образом, в ЭВМ нет чисел типа REAL(4), меньших -3.402823E+38 и больших +3.402823E+38. Также для этого типа нет чисел, если не считать число 0, между -1.175494E-38 и 1.175494E-38. Более того, нет чисел типа REAL(4) между представимыми в ЭВМ числами *val* и *val* + NEAREST(*val*, 1.0). Например, если *val* = 10000.0, то в ЭВМ нельзя представить числа между 10000.0 и 10000.0 + 9.765625E-04. Действительно, оператор

```
print *, nearest(val, 1.0) - val
```

напечатает 9.765625E-04.

Существующие разрывы между представимыми в ЭВМ вещественными числами можно трактовать так: для каждого числа *val* > 0 существует минимальное положительное число ε_{min}, которое в сумме с *val* даст число, большее *val*. Для положительных чисел, меньших ε_{min}, справедливо

$$val + \varepsilon_{min} = val.$$

Причем чем меньше *val*, тем меньше значение соответствующего ему ε_{min}. Пренебрежение этим обстоятельством может явиться причиной разнообразных ошибок, например зацикливания. Поиск ε_{min}, например при работе с двойной точностью, можно выполнить, применив код

```
program em
 real(8) :: val = 10000.0_8, emin
 emin = 1.0_8
 do while(val + emin > val)
   emin = emin / 2.0_8
 end do
 emin = emin * 2.0_8
 print *, emin                          ! 1.818989403545857E-012
 ! Тот же результат можно получить иначе:
 print *, nearest(val, 1.0_8) - val     ! 1.818989403545857E-012
end program em
```

Аналогичные предосторожности необходимы и при работе с отрицательными вещественными числами.

1.7.2. Ошибки округления

Ограничения, связанные с представлением вещественных и комплексных чисел, означают, что при выполнении операций с такими числами могут возникать *ошибки округления*. Пренебрежение ошибками округления может привести к драматическим последствиям. Рассмотрим, например, цикл

```
real(4) :: x = 0.1
do                                  ! Бесконечный цикл
 print *, x
 x = x + 0.1
 if(x == 1.0) exit                  ! EXIT – оператор выхода из цикла
end do
```

Он является бесконечным, поскольку *x* из-за ошибки округления не принимает значения, точно равного 1.0. Нормальное завершение цикла обеспечит код

```
real(4) :: x = 0.1, eps = 1.0e-6
do
 print *, x
 x = x + 0.1
 if(abs(x - 1.0) < eps) exit        ! x практически равен 1.0
end do                              ! ABS(x - 1.0) возвращает |x - 1.0|
```

Общий вывод из приведенного примера: нельзя сравнивать вещественные числа на предмет точного равенства или неравенства, а следует выполнять их сравнение с некоторой точностью.

Влияние ошибок округления можно снизить, правильно формируя порядок вычислений. Пусть, например, объявлены и инициализированы переменные *x*, *y* и *z*:

```
real(4) :: x = 1.0e+30, y = -1.0e+30, z = 5.0
```

Их сумма равна 5.0. Однако найдем и выведем их сумму так:

```
print *, x + (y + z)                !    0.000000E+00                (ошибка)
```

Результат ошибочен. Правильной является такая последовательность вычислений:

```
print *, (x + y) + z                !    5.000000                    (верно)
```

Другой пример взят из [10]. Пусть надо найти *xc* – середину отрезка [*a*, *b*]. Обычно *xc* вычисляют по формуле

$$xc = 0.5(xa + xb). \qquad (1.1)$$

Предположим, что $a = 0.982$, $b = 0.984$ и вычисления производятся на трехразрядной десятичной машине, выполняющей арифметические операции с плавающей точкой с округлением. Тогда сумма $xa + xb$ после округления будет равна 1.97, а формула (1.1) даст среднюю точку 0.985, которая находится вне отрезка $[xa, xb]$. Правильный результат на такой машине даст формула

$$xc = xa + 0.5(xb - xa) = 0.982 + 0.001 = 0.983. \qquad (1.2)$$

Поэтому библиотечные процедуры применяют в методе бисекций поиска корня нелинейного уравнения (разд. 10.1.2.1) формулу (1.2).

1.7.3. Устранение переполнения и исчезновения порядка

Код процедур IMSL устроен так, что вычисления происходят без *переполнения* и *исчезновения порядка*. Поясним эти понятия. Рассмотрим, например, задачу вычисления 2-нормы вектора x, содержащего n элементов типа REAL(4):

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}\ .$$

Очевидно, что при весьма малых, но представимых в ЭВМ значениях x_i после вычисления x_i^2 результат может оказаться меньше минимально представимого на ЭВМ положительного числа. В таких случаях говорят, что произошло *исчезновение порядка*, при котором результат заменяется нулем. Если исчезновение порядка наблюдается для всех элементов, то 2-норма окажется равной нулю. И наоборот, при сравнительно больших, но представимых в ЭВМ элементах вектора x_i^2 может получиться большим, чем наибольшее представимое на ЭВМ число. В таких случаях говорят, что произошло *переполнение*, при котором результатом вычислений будет машинная бесконечность. Как правило, при переполнении программа, если не принять специальных мер, досрочно прекратит работу. (Вопросы представления в ЭВМ вещественных и целых чисел подробно изложены в гл. 2.)

Пример:

```
real(4) :: r1, r2
r1 = huge(r1); r2 = tiny(r2)
r1 = r1 * r1; r2 = r2 * r2
print *, r1                    ! 1#INF........
print *, r2                    ! 0.000000E+00
```

Вернемся к вычислению 2-нормы. Чтобы избежать переполнения, следует найти наибольший элемент вектора x и произвести расчет по формуле

$$\|x\|_2 = |x_{\max}|\sqrt{\sum_{i=1}^{n}\left(\frac{x_i}{x_{\max}}\right)^2}.$$

Получение результата обеспечит код

```
program eNorm
 real(4) :: x2, xm, x(10)                    ! x2 – искомая 2-норма
 x = tiny(x)                                 ! Инициализация массива x
 xm = maxval(x)
 x2 = xm * sqrt(sum((x / xm)**2))
 print *, x2, tiny(x)                        ! 3.717240E-38   1.175494E-38
end program eNorm
```

Если в процедуре IMSL все же возникает переполнение, то его причиной являются неверные входные данные или ошибки в размерах массивов.

1.7.4. Оценка точности результата

Процедуры IMSL основаны на алгоритмах, обеспечивающих надлежащую для решаемого класса задач точность результатов. Скажем, системы линейных уравнений с матрицей общего вида решаются методом Гаусса, в котором для повышения точности используется частичный выбор ведущего элемента (разд. 5.3.1). Далее точность решения линейной системы можно повысить, применив итерационное уточнение корней (разд. 5.6.2).

Но точность определяется не только качеством алгоритма и применяемыми вычислительными средствами, но и обусловленностью вычислительной задачи. Рассмотрим эту важную характеристику более подробно.

Под *обусловленностью вычислительной задачи* понимают чувствительность ее решения к малым изменениям входных данных. Задачу называют *хорошо обусловленной*, если при малых изменениях входных данных результат также изменяется незначительно. Задача называется *плохо обусловленной*, если малые изменения входных данных могут привести к большим изменениям решения.

Часто входные данные являются результатом экспериментов и, следовательно, они лишь приблизительно отражают реальные физические значения. (Ошибки могут быть вызваны как неточностью измерений, так погрешностью эксперимента.) Поэтому в плохо обусловленной задаче вероятность получения точного решения незначительна. Причем эта возможная неточность связана с характером используемой для описания явления модели и не может быть устранена вычислительными методами.

Из сказанного следует, что специалисту нужно знать, какими качествами обладает его модель. В том числе он должен уметь количественно оценить ее обусловленность. И если задача окажется плохо обусловленной, то, скорее всего, следует предпринять усилия по изменению модели с целью улуч-

щения этой характеристики. В противном случае нельзя гарантировать точность получаемых в результате применения вычислительных методов решений.

Пример плохо обусловленной задачи. Пусть некоторая модель задается уравнением

$$x^4 - 4x^3 + 8x^2 - 16x + 16 = 0. \qquad (1.3)$$

Его левая часть является раскрытием выражения $(x - 2)^4$. Поэтому решение очевидно: $x_1 = x_2 = x_3 = x_4 = 2$.

Предположим, что свободный член исходного уравнения найден с ошибкой, равной 10^{-8}. Тогда уравнение примет вид

$$x^4 - 4x^3 + 8x^2 - 16x + 15.99999999 = 0, \qquad (1.4)$$

что эквивалентно $(x - 2)^4 = 10^{-8}$.

Корни этого уравнения: $x_1 = 2.001$, $x_2 = 1.99$, $x_3 = 2 + 0.01i$ и $x_4 = 2 - 0.01i$. Относительная ошибка решения равна 0.5%, что существенно выше относительного изменения свободного члена уравнения, равного 10^{-6}%, и указывает на плохую обусловленность задачи.

Найдем корни уравнения (1.3), обратившись к подпрограмме ZPORC библиотеки IMSL.

```
program zeros
 use msimsl
 integer(4), parameter :: ndeg = 4
 real(4) :: coeff(ndeg + 1) = (/ 16.0, -16.0, 8.0, -4.0, 1.0 /)
 complex(4) :: root(ndeg)
 call zporc(ndeg, coeff, root)
 print *, root
end program zeros
```

Результат:

```
(2.000097, 0.000000E+00)        (-1.396984E-09, 2.000000)
(-1.396984E-09, -2.000000)      (1.999903, 0.000000E+00)
```

Второй и третий корни оказались комплексными. То есть два действительных кратных корня найдены не были.

Для уравнения (1.4) ZPORC находит действительные корни 2.000096 и 1.999904. Относительная ошибка решения (ее можно вычислить, поскольку известно точное значение корня)

$$\delta = 100 * (2.01 - 2.000096)/2.01 = 0.493\%.$$

Заметим, что ZPORC не анализирует обусловленность задачи вычисления корней многочлена, и, следовательно, затруднительно оценить их точность. Однако для линейных моделей процедуры IMSL все же позволяют вычислять оценки точности результатов. Так, при решении систем линейных уравнений точность можно оценить по значению числа обусловленно-

сти матрицы. Такая оценка является априорной. Точность собственных значений матрицы оценивается после их вычисления, поэтому такая оценка уже является апостериорной. В тех же ситуациях, когда IMSL не выдает оценки точности решения, эта работа должна быть выполнена пользователем.

1.8. ОБРАБОТКА ОШИБОК

Процедуры IMSL, как правило, обнаруживают ошибки во входных данных и ошибки, возникающие в процессе вычислений. По умолчанию информация об обнаруженных ошибках выводится на экран. При необходимости вывод можно направить в файл (разд. 3.3.6). В случае серьезных ошибок IMSL прекращает вычисления. Умеренные ошибки сопровождаются предупреждениями; вычисления с такими ошибками не прерываются. Классификация ошибок IMSL приведена в разд. 3.4.1. Характер реагирования на ошибки может регулироваться пользователем (разд. 3.4.3).

1.9. СОГЛАШЕНИЯ, ДЕЙСТВУЮЩИЕ ПРИ ОПИСАНИИ ПРОЦЕДУР БИБЛИОТЕКИ IMSL

При описании процедуры IMSL указывается вызывающий ее оператор, например

CALL LFCRG(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *rcond*)

если процедура является подпрограммой, и

result = CONST(*name*)

если – функцией. В приводимом операторе имя процедуры записывается прописными буквами, имена параметров – курсивом и строчными буквами. Исключение составляют ведущие буквы L, которые пишутся как прописные. Так, вместо *lda* указываем *Lda*.

Описание каждого параметра начинается с красной строки, например:

name – символьная строка, содержащая имя константы, значение которой нужно узнать.

Таким же образом описывается результирующая переменная функции, имя которой совпадает с именем функции. Например:

CONST – значение константы *name*.

Для представления матриц используются двумерные массивы, имена которых совпадают с именами матриц, но в отличие от последних записываются строчными буквами. Например, матрицу *A* представляет массив *a*, матрицу *FAC* – массив *fac*.

В программах, содержащих вызовы процедур IMSL, все имена записываются прямыми строчными буквами, кроме имен, начинающихся с буквы L. Например:

```
call Lfcrg(n, a, Lda, fac, Ldfac, ipvt, rcond)
```

Имена переменных в сопровождающем программу комментарии пишутся курсивом и строчными буквами, а процедур – прямыми прописными, например:

```
! rj(j) – столбец с номером j единичной матрицы, поэтому подпрограмма LFIRG
! разместит столбец j обратной матрицы A^-1 в j-й столбец массива ainv
```

Написание имен процедур, не входящих в библиотеку IMSL, и их параметров может не подчиняться приведенным правилам.

1.10. ВСПОМОГАТЕЛЬНАЯ ФУНКЦИЯ *ru_doswin*

Для вывода русского текста в DOS-окно в пособии используется функция *ru_doswin*, код и порядок употребления которой приведены в прил. 1. Чтобы получить доступ к функции *ru_doswin,* в проекте нужно разместить код модуля *text_transfer*, содержащего *ru_doswin*, и выполнить в соответствующих программных компонентах ссылку

```
use text_transfer
```

2. ВЫЧИСЛЕНИЯ С ПЛАВАЮЩЕЙ ТОЧКОЙ

2.1. ДВОИЧНЫЕ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ

2.1.1. Представление целых чисел

Каждая разновидность целого и вещественного типа содержит конечное множество чисел. Так, тип INTEGER(2) представляет все целые числа из диапазона от -32,768 до +32,767. Каждое такое множество чисел может быть описано моделью. Данные о параметрах модели заданной разновидности типа и о конкретных характеристиках числа в задающей его модели позволяют получать встроенные числовые справочные и преобразовывающие функции, а также функции IMACH, AMACH и DMACH библиотеки IMSL.

Двоичное представление целого числа i задается формулой

$$i = \underbrace{(-1)^s}_{знак} b_0 b_1 \ldots b_{Q-1}, \qquad (2.1)$$

где s – это 0 или 1 (+ или -); b_i – двоичное число (0 или 1); Q – число цифр в целом числе по основанию 2.

2.1.2. Вещественные числа с плавающей точкой

Вещественные числа с плавающей точкой представляются в FPS в близком соответствии со стандартом IEEE для арифметики с плавающей точкой (ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, 1985). Фортран поддерживает форматы одинарной точности – тип REAL(4), двойной точности – тип REAL(8) и повышенной точности, используемой для выполнения промежуточных операций. Например, в следующем коде:

```
real(4) :: a, b, c, d, f                    ! Задана опция компилятора /Op
b = 0.0; c = 1.0e30; d = 1.0e30; f = 1.0e-30
a = (b + (c * d)) / 2.0 * f
print *, a                                  ! 5.000000E+29
```

промежуточные вычисления, если задана опция компилятора /Op, FPS выполнит с повышенной точностью. Если же при компиляции задана опция /Oxp, предусматривающая полную оптимизацию скорости вычислений и проверку ошибок, то на этапе компиляции возникнут предупреждения вида

```
warning F4756: overflow in constant arithmetic
```

(по причине переполнения в результате умножения c * d = 1.0e30 * 1.0e30), а результатом вычислений будет машинная бесконечность – 1#INF.....

Двоичное представление вещественного числа x с плавающей точкой задается формулой

$$x = \underbrace{(-1)^s}_{\text{знак}} \underbrace{b_0.b_1b_2\ldots b_{P-1}}_{\text{мантисса}} \times 2^E, \qquad (2.2)$$

где s – это 0 или 1 (+ или -); b_i – двоичное число (0 или 1); P – число цифр в мантиссе нормализованного представления вещественного числа по основанию 2; E – целое число, называемое (*двоичным*) *порядком*, из отрезка $E_{\min} \le E \le E_{\max}$. В табл. 2.1 приводятся значения параметров модели вещественных чисел для одинарной, двойной и повышенной точности.

Таблица 2.1. Параметры модели чисел стандарта IEEE Std 754

Параметр	*Точность*		
	Одинарная	*Двойная*	*Повышенная*
Число бит для знака	1	1	1
P	24	53	64
$E_{\max}$	+128	+1024	+16384
$E_{\min}$	-125	-1021	-16381
Смещение двоичного порядка	+126	+1022	+16382
Число бит для двоичного порядка	8	11	15
Число бит для числа	32	64	80

Стандарт требует, чтобы числа одинарной и двойной точности представлялись в нормализованном виде, поэтому всегда $b_0 = 1$ и, следовательно, для представления мантиссы чисел одинарной точности требуется 23 бита, а двойной – 52. Числа повышенной точности не нормализуются, поэтому для мантиссы требуется 64 бита. При записи порядка к нему с целью повышения скорости вычислений добавляется *смещение*, поэтому порядок всегда представляется в виде положительного числа *e*. Реально, однако, порядок E = *e – смещение*. Структура формата вещественных чисел двойной точности с плавающей точкой приведена на рис. 2.1.

1	11	52
Знак	Порядок + смещение	Мантисса

Рис. 2.1. Структура IEEE-формата вещественных чисел двойной точности

<u>*Замечания*</u>:

1. Множество представимых в компьютере чисел с плавающей точкой конечно. Так, для типа REAL(4) их число примерно равно 2^{31}.
2. FPS и CVF содержат программу BitViewer просмотра двоичного представления вещественных чисел одинарной и двойной точности.

2.1.3. Виды вещественных чисел с плавающей точкой

В дополнение к нормализованным стандарт IEEE Std 754 содержит 4 специальных ненормализованных вида чисел. Все предусмотренные IEEE Std 754 виды вещественных чисел с плавающей точкой приведены в табл. 2.2.

Таблица 2.2. Возможные виды вещественных чисел с плавающей точкой

Вид числа	*Значение*	*Порядок*	*Мантисса (sig)*
Нуль со знаком	± 0	$E = E_{\min} - 1$	$sig = 0$
Денормализованное число	$\pm 0.sig \times 2^{E_{\min}}$	$E = E_{\min} - 1$	$sig \neq 0$
Нормализованное число	$\pm 1.sig \times 2^{E}$	$E_{\min} \le E \le E_{\max}$	$b_0.b_1b_2 \mathrm{K}\ b_{P-1}$
Бесконечность со знаком	$\pm\infty$	$E = E_{\max} + 1$	$sig = 0$
Не число	NaN	$E = E_{\max} + 1$	$sig \neq 0$

Специальные виды чисел интерпретируются так:

- *нуль со знаком*. По умолчанию нуль рассматривается как число со знаком. Если используется встроенная функция SIGN, в которой вторым параметром является 0, то при выработке результата будет учитываться знак нуля. Однако при сравнении +0 равен -0. В большинстве задач знак нуля не имеет значения, но в некоторых приложениях численного анализа он полезен;
- *денормализованные числа*. Заполняют промежуток между наименьшим положительным и наибольшим отрицательными числами (между ними, если нет денормализованных чисел, находится только ±0). Денормализованные числа позволяют избежать исчезновения порядка при вычислении промежуточных результатов с повышенной точностью. Денормализацию можно обнаружить подпрограммой GETSTATUSFPQQ, использовав соответствующим образом при ее вызове параметр FPSW$DENORMAL;
- *бесконечность со знаком*. Бесконечность со знаком позволяет продолжать вычисления при возникновении переполнения. Знак бесконечности определяется знаком операндов, ее породивших. Исследуя статус контрольного слова, возвращаемого подпрограммой GETSTATUSFPQQ, можно различить бесконечность, возникшую в результате переполнения и деления на нуль. По умолчанию Фортран формирует бесконечность со знаком, имеющую значение ±1#INF;
- *не число*. Стандарт IEEE определяет NaN (Not a Number – не число) как результат некорректных операций, например 0/0.

Пример. Формируются машинная бесконечность и не число. Прежде, однако, подпрограммой SETCONTROLFPQQ изменяется контрольное слово

таким образом, что деление на нуль и некорректные результаты не будут вызывать завершения программы.

```
program inf
 use msflib; use msimsl
 use text_transfer                          ! Для вызова ru_doswin
 integer(2) :: control, newcontrol
 real(4) :: x = 0.0, y, z
 call getcontrolfpqq(control)               ! Читаем существующее управляющее слово
 ! Разрешаем деление на нуль
 newcontrol = control .or. fpcw$zerodivide .or. fpcw$invalid
 call setcontrolfpqq(newcontrol)            ! Устанавливаем новое управляющее слово
 y = 1.0 / x                                ! Деление на нуль
 print *, y                                 ! 1#INF........
 z = 0.0 / x                                ! Некорректная операция
 print *, z                                 ! 1#IND........
 if(ifnan(z)) then
  ! Для вывода русского текста в DOS-окне используем ru_doswin
  print *, trim(ru_doswin('z – это NaN', .false.))
 else
  print *, ' z = ', z
 end if
end program inf
```

Результат:

1#INF.....

1#IND.....

z – это NaN

2.1.4. характеристики модели представления чисел

Приводятся функции, возвращающие характеристики модели, в которой содержится параметр функции. Параметром функции может быть как скаляр, так и массив. Значение параметра может быть неопределенным.

DIGITS(x) – возвращает число двоичных значащих цифр в модели представления x (т. е. Q или P). Параметр x может быть целого или вещественного типа. Результат имеет стандартный целый тип.

Тип параметра x	*DIGITS(x)*
INTEGER(1)	7
INTEGER(2)	15
INTEGER(4)	31
REAL(4)	24
REAL(8)	53

EPSILON(x) – возвращает 2^{1-P}:

Тип параметра x	*EPSILON(x)*
REAL(8)	2.22044049250313E-016
REAL(4)	1.192093E-07

Замечание. Возвращаемое функцией и EPSILON число часто называют *машинной точностью* и обозначают $\varepsilon_м$.

HUGE(x) – для целого или вещественного x возвращает наибольшее значение x. Тип и параметр типа результата такие же, как у x. Значение равно $2^Q - 1$ – для целого x – и $\left(1 - 2^{-P}\right)2^{E_{\max}}$ – для вещественного x.

Тип параметра x	*HUGE(x)*
INTEGER(1)	127
INTEGER(2)	32,767
INTEGER(4)	2,147,483,647
REAL(4)	3.402823E+38
REAL(8)	1.797693134862316E+308

MAXEXPONENT(x) – для вещественного x возвращает максимальное значение порядка, т. е. E_{max}. Результат имеет стандартный целый тип.

Тип параметра x	*MAXEXPONENT(x)*
REAL(4)	128
REAL(8)	1024

MINEXPONENT(x) – для вещественного x возвращает минимальное значение порядка, т. е. $E_{\min}$. Результат функции имеет стандартный целый тип.

Тип параметра x	*MINEXPONENT(x)*
REAL(4)	-125
REAL(8)	-1021

PRECISION(x) – для вещественного или комплексного x возвращает число значащих цифр, следующих после десятичной точки, используемых для представления чисел с таким же параметром типа, как у x. Результат функции имеет стандартный целый тип.

Тип параметра x	*PRECISION(x)*
REAL(4) или COMPLEX(4)	6
REAL(8) или COMPLEX(8)	15

RADIX(x) – для целого или вещественного x возвращает стандартное целое, равное основанию системы счисления, используемой для представления чисел. Например:

```
print *, radix(0.0)                    ! 2
```

RANGE(x) – для целого, вещественного или комплексного x возвращает эквивалентный десятичный степенной диапазон значений x, т. е.

INT(LOG10(*huge*))

для целого и

INT(MIN(LOG10(*huge*), -LOG10(*tiny*)))

для вещественного или комплексного x, где *huge* и *tiny* – наибольшее и наименьшее числа в соответствующих разновидностях типа. Результат функции – стандартного целого типа.

Тип параметра x	*RANGE(x)*
INTEGER(1)	2
INTEGER(2)	4
INTEGER(4)	9
REAL(4) или COMPLEX(4)	37
REAL(8) или COMPLEX(8)	307

TINY(x) – для вещественного x возвращает наименьшее положительное значение x, т. е. $2^{E_{\min}}$. Тип и параметр типа результата такие же, как у x.

Тип параметра x	*TINY(x)*
REAL(4)	1.175494E-38
REAL(8)	2.225073858507201E-308

Следующие функции возвращают значения, связанные с компонентами двоичного представления аргумента.

EXPONENT(x) – возвращает степенную часть (т. е. порядок E) двоичного представления заданного вещественного x. Результат – стандартного целого типа. Результат равен нулю, если $x = 0$. Например:

```
real(4) :: r1 = 1.0
real(8) :: r2 = 123456789123456.7
write(*,*) exponent(r1)                ! 1
write(*,*) exponent(r2)                ! 47
```

FRACTION(x) – возвращает мантиссу – дробную часть двоичного представления x, т. е. $2^{-E}x$. Тип x – вещественный. Тип и разновидность типа результата такие же, как у x. Например:

```
print *, fraction(3.0)                 !    0.75
print *, 3.0*2.0**(-exponent(3.0))     !    0.75
print *, fraction(1024.0)              !    0.5
```

NEAREST(x, s) – возвращает вещественное значение с таким же параметром типа, как у x, равное ближайшему к x машинному числу, большему x, если $s > 0$, и меньшему x, если $s < 0$; s не может равняться нулю.

```
real(4) :: r1 = 3.0                          ! Пример для NEAREST
real(8) :: r2 = 3.0_8
! Используем для вывода шестнадцатеричную систему счисления
write(*, '(1x,z18)') nearest(r1, 2.0)        !                  40400001
write(*, '(1x,z18)') nearest(r2, -2.0_8)     !            4007FFFFFFFFFFFF
```

Замечание. Числа с плавающей точкой между нулем и HUGE(x) распределены неравномерно. В случае REAL(4) между каждыми соседними степенями двойки находится примерно 2^{22} чисел с плавающей точкой. Так, примерно 2^{22} числа находится между 2^{-125} и 2^{-124} и столько же между 2^{124} и 2^{125}. Простое сопоставление говорит о том, что числа с плавающей точкой гуще расположены вблизи нуля.

RRSPACING(x) – возвращает вещественное значение с таким же параметром типа, как у x, равное обратной величине относительного расстояния между числами в двоичном представлении x, в области, близкой к x, т. е. $2^P\left|2^{-E}x\right|$.

```
print *, rrspacing( 3.0_4)                   !     1.258291e+07
print *, rrspacing(-3.0_4)                   !     1.258291e+07
```

SCALE(x, i) – возвращает вещественное значение с таким же параметром типа, как у x, равное $2^i x$, где i – целое число.

```
print *, scale(5.2, 2)                       !     20.800000
```

SET_EXPONENT(x, i) – возвращает вещественное значение с таким же параметром типа, как у x, равное $2^{i-E}x$, где i – целое число, а E = EXPONENT(x).

SPACING(x) – возвращает вещественное значение с таким же параметром типа, как у x, равное абсолютному расстоянию между числами в двоичном представлении, в области, близкой к x, т. е. 2^{P-E}.

```
print *, spacing(3.0_4)                      !     2.384186e-07
```

2.1.5. Абсолютная и относительная ошибки

Пусть **F** – множество представимых в компьютере вещественных чисел. Определим множество

$$G = \{x \in \mathbf{R} : tiny \le |x| \le huge\} \cup \{0\},$$

где *tiny* и *huge* – соответственно наименьшее и наибольшее возможные значения x, и оператор *fl*: $G \to \mathbf{R}$, выполняющий округление вещественного

числа $x \in G$ до ближайшего вещественного числа с плавающей точкой $\hat{x} \in \mathbf{F} : \hat{x} = fl(x)$. Оператор fl удовлетворяет соотношению

$$fl(x) = x(1+\varepsilon), |\varepsilon| \le \varepsilon_м,$$

где $\varepsilon_м$ – машинная точность. Тогда относительная ошибка отдельной арифметической операции $a\ op\ b$, где op – знак операции,

$$\frac{|fl(a\ op\ b) - (a\ op\ b)|}{(a\ op\ b)} \le \varepsilon_м, a\ op\ b \ne 0 .$$

Следовательно, относительная погрешность отдельной арифметической операции мала. Однако при выполнении последовательности операций ошибки могут накапливаться и вырастать до неприемлемой величины. Таким образом, одна из задач проектирования вычислительного алгоритма – обеспечение последовательности операций, не приводящей в результате округлений к чрезмерному росту ошибки вычислений.

2.2. ИСКЛЮЧЕНИЯ

Если вещественное число x не равно точно одному из чисел, представляемых в компьютере, то x будет представляться ближайшим к нему числом с плавающей точкой $\hat{x}$ из модели (2), называемым *приближением* x. При этом возникнет *ошибка округления*, равная $|x - \hat{x}|$. Способ округления числа x устанавливается подпрограммой SETCONTROLFPQQ и зависит от примененного при ее вызове параметра.

Если результат арифметической операции не равен точно имеющемуся числу с плавающей точкой, то он тоже округляется. Также он может быть равен одному из специальных значений: нулю со знаком, денормализованному числу, бесконечности или не числу. Причем, если этот результат промежуточный, он будет участвовать в дальнейших вычислениях и выражение может вернуть пригодную для употребления величину. Результат, если он настолько мал или велик, что не может быть обработан процессором, приводит к *исчезновению порядка* или *переполнению*. По умолчанию в случае исчезновения результат заменяется нулем, а переполнения приводят к ошибке исполнения и аварийному завершению программы.

Приближения, специальные величины, переполнение и исчезновение порядка являются *исключениями с плавающей точкой*. Программист может выбрать способ обработки этих исключений, вызывая подпрограмму SETCONTROLFPQQ и задавая соответствующие значения ее параметра, называемого *управляющим словом вычислений с плавающей точкой* (в дальнейшем просто – *управляющее слово*)

2.3. УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

2.3.1. Значения статуса и управляющего слова

Процессор с плавающей точкой содержит два 16-битовых регистра, в один из которых записывается статус, а в другой – управляющее слово операций с плавающей точкой. Статус и управляющее слово являются результатом логических операций, таких, как .OR., специальных констант типа INTEGER(2) из фиксированного набора значений, определенных в файле msflib.f90 и приведенных в табл. 2.3 и 2.4.

Таблица 2.3. Константы статуса операций с плавающей точкой

Имя	*Значение*	*Описание*
FPSW$MSW_EM	#003F	Маска статуса; устанавливает во все биты число 1
FPSW$INVALID	#0001	Недопустимый результат, например 0/0
FPSW$DENORMAL	#0002	Результат является денормализованным числом
FPSW$ZERODIVIDE	#0004	Произошло деление на нуль
FPSW$OVERFLOW	#0008	Переполнение
FPSW$UNDERFLOW	#0010	Исчезновение порядка
FPSW$INEXACT	#0020	Приближение

Таблица 2.4. Константы управляющего слова

Имя	*Значение*	*Описание*
FPCW$MCW_IC	#1000	Управляющая маска машинной бесконечности
FPCW$AFFINE	#1000	Аффинная бесконечность
FPCW$PROJECTIVE	#0000	Проективная бесконечность
FPCW$MCW_PC	#0300	Управляющая маска точности
FPCW$64	#0300	Точность 64 бита
FPCW$53	#0200	Точность 53 бита
FPCW$24	#0000	Точность 24 бита
FPCW$MCW_RC	#0C00	Управляющая маска округления
FPCW$CHOP	#0C00	Отсечение
FPCW$UP	#0800	Округление до ближайшего большего
FPCW$DOWN	#0400	Округление до ближайшего меньшего
FPCW$NEAR	#0000	Округление по правилам арифметики
FPCW$MSW_EM	#003F	Маска исключений

FPCW$INVALID	#0001	Разрешены недопустимые числа, например 0/0
FPCW$DENORMAL	#0002	Разрешены денормализованные числа
FPCW$ZERODIVIDE	#0004	Разрешено деление на нуль
FPCW$OVERFLOW	#0008	Разрешены переполнения
FPCW$UNDERFLOW	#0010	Разрешены исчезновения порядка
FPCW$INEXACT	#0020	Разрешены приближения

По умолчанию управляющее слово равно

```
fpcw$53 .or. fpcw$near .or. fpcw$denormal .or. fpcw$underflow .or. fpcw$inexact
```

То есть промежуточные вычисления выполняются с повышенной точностью, округление производится до ближайшего числа, разрешены три вида исключений: денормализация, исчезновение порядка и приближение.

2.3.2. Чтение статуса

Записанный в процессор статус возвращается подпрограммой GETSTATUSFPQQ библиотеки MSFLIB. Чтобы узнать, какая из констант табл. 2.3 присутствует в статусе, или, иными словами, чтобы определить вид возникшего исключения, вместе с возвращенным статусом используется логическая операция .AND..

Пример. В первой операции 1.0/2.0 исключений не возникает, вторая операция 1.0/3.0 выполняется с приближением. Возникшие ситуации анализируются подпрограммой *except*. Операция с денормализацией также будет обнаружена подпрограммой.

```
program ex
 real(4) :: a, b, c, x
 a = 1.0; b = 2.0; c = 3.0
 x = a / b; call except( )                ! 1.0 / 2.0 – нет исключений
 x = a / c; call except( )                ! 1.0 / 3.0 – будет выполнено приближение
end program ex

subroutine except                         ! Анализ исключений
 use msflib; use text_transfer
 integer(2) status
 call getstatusfpqq(status)
 if((status .and. fpsw$inexact) > 0) then
   print *, ru_doswin("Приближение", .false.)
 else if((status .and. fpsw$denormal) > 0) then
   print *, ru_doswin("Денормализация", .false.)
 else
   print *, ru_doswin("Нет исключений", .false.)
 end if
end subroutine except
```

2.3.3. Изменение управляющего слова

Выполняется подпрограммой SETCONTROLFPQQ. Например, после выполнения операторов

```
integer(2) :: newcontrol
newcontrol = fpcw$inexact .or. fpcw$chop
call setcontrolfpqq(newcontrol)
```

все промежуточные вычисления будут выполняться с одинарной точностью, останется одно исключение – приближение, которое будет осуществляться простым отбрасыванием разрядов (отсечением). Управляющее слово не должно содержать исключающих параметров, например взаимоисключающими являются точности FPCW$64 и FPCW$53 или округления FPCW$UP и FPCW$DOWN и т. д.

Если же надо изменить лишь 1 бит управляющего слова, то лучше прежде, вызвав GETCONTROLFPQQ, прочитать существующее слово, затем при необходимости очистить все биты изменяемого раздела, применив с прочитанным словом подходящую управляющую маску, и внести в него соответствующие изменения.

Пример 1. Для промежуточных вычислений устанавливается точность, равная 64 битам.

```
use msflib
integer(2) :: control, clearcontrol, newcontrol
call getcontrolfpqq(control)              ! Читаем существующее управляющее слово
! Очищаем все его биты, отвечающие за точность
clearcontrol = control .and. (.not. fpcw$mcw_pc)
! Новое управляющее слово содержит точность в 64 бита
newcontrol = clearcontrol .or. fpcw$64
call setcontrolfpqq(newcontrol)           ! Устанавливаем новое управляющее слово
```

Пример 2. Изменяем способ приближения и разрешаем деление на нуль.

```
program cow
 use msflib
 integer(2) :: control, clearcontrol, newcontrol
 real(4) :: x = 0.0, y
 call getcontrolfpqq(control)             ! Читаем существующее управляющее слово
 ! Очищаем все его биты, отвечающие за приближение
 clearcontrol = control .and. (.not. fpcw$mcw_rc)
 ! Округление будет выполняться до ближайшего числа с плавающей точкой
 ! Деление на нуль не приведет к аварийному завершению
 newcontrol = clearcontrol .or. fpcw$up .or. fpcw$zerodivide
 call setcontrolfpqq(newcontrol)          ! Устанавливаем новое управляющее слово
 y = 1.0 / x;          print *, y         ! 1#INF........
 a = y / 2.0e30        print *, a         ! 1#INF........
end program cow
```

2.4. ОБРАБОТКА ИСКЛЮЧЕНИЙ

Исключения возникают при выполнении операций с плавающей точкой, например при делении на нуль, и при вызове математических функций – математические исключения, например извлечение корня из отрицательного числа.

2.4.1. Обработка исключений с плавающей точкой

При появлении исключения, если оно запрещено, возникает соответствующий ему один из приведенных в табл. 2.5 сигналов прерывания.

Таблица 2.5. Сигналы прерываний, возникающие при исключениях

Имя сигнала	*Значение*	*Исключение*
FPEW$INVALID	#81	Недопустимое число
FPEW$DENORMAL	#82	Денормализованное число
FPEW$ZERODIVIDE	#83	Деление на нуль
FPEW$OVERFLOW	#84	Переполнение
FPEW$UNDERFLOW	#85	Исчезновение порядка
FPEW$INEXACT	#86	Приближение

По умолчанию сигнал прерывания приведет к завершению программы. Можно заменить встроенный обработчик сигналов прерываний на свой, написав, во-первых, функцию, реагирующую на прерывания, и, во-вторых, зарегистрировав ее, применив SIGNALQQ в качестве процедуры, вызываемой при возникновении прерывания. Функция обработки прерываний должна иметь атрибут C. Возможна, например, такая реализация обработчика прерываний:

```
program ex2
 use msflib
 interface
  function hand_fpe(sigid, except)          ! Интерфейс функции для обработки прерываний
   !ms$attributes c :: hand_fpe             ! Обязательный атрибут C
   integer(4) :: hand_fpe
   integer(2) :: sigid, except
  end function hand_fpe
 end interface
 integer(4) :: result
 real(4) :: x = 0.0, y
 result = signalqq(sig$fpe, hand_fpe)       ! Регистрация функции как обработчика прерываний
 ! Выполняем деление на нуль; будет вызвана функция hand_fpe
 y = 2.0 / x                                ! Если функция hand_fpe не содержит оператор
 print *, y                                 ! STOP, то вычисления будут продолжены с y = 2.0
end program ex2
```

```
function hand_fpe(signum, excnum)          ! Обработчик прерываний
 !ms$attributes c :: hand_fpe
 use msflib; use text_transfer
 integer(2) :: signum, excnum
 character(40) :: fpe
 print *, trim(ru_doswin('Номер прерывания:', .false.)), signum
 select case(excnum)
 case(fpe$invalid)
  fpe = ru_doswin('Недопустимое число', .false.)
 case(fpe$denormal)
  fpe = ru_doswin('Денормализованное число', .false.)
 case(fpe$zerodivide)
  fpe = ru_doswin('Деление на нуль', .false.)
 case(fpe$overflow)
  fpe = ru_doswin('Переполнение', .false.)
 case(fpe$underflow)
  fpe = ru_doswin('Исчезновение порядка', .false.)
 case(fpe$inexact)
  fpe = ru_doswin('Приближение', .false.)
 case default
  fpe = ru_doswin('Не IEEE-тип', .false.)
 end select
 print *, fpe
 hand_fpe = 1
 stop                                      ! Если оператор отсутствует, то вычисления
end function hand_fpe                      ! будут продолжены
```

Результат:

```
Номер прерывания:      8
Деление на нуль
Stop – Program terminated.
```

Замечание. Пользователи CVF вместо *use msflib* могут указать *use dflib*.

Прерывания возникают при обнаружении неразрешенных исключений (соответствующий им в управляющем слове бит равен нулю). В результате прерывания, если оно обрабатывается встроенными средствами, процесс завершается и выдается сообщение, например такое:

```
run-time error M6103: MATH – floating-point error: divide by zero
```

Если обработчик прерываний создан программистом, то в нем можно предусмотреть, как это и сделано в приведенном выше примере, оператор останова STOP. Если же такой оператор отсутствует, то вычисления будут продолжены. Причем результатом операции $y = 2.0 / x$, где $x = 0.0$, будет 2.0, т. е. деление на x просто не будет выполнено. Однако поскольку такой результат операции далек от ожидаемого, продолжать вычисления не имеет

смысла. Кроме того, обработчик *hand_fpe* на новые прерывания реагировать не будет. Поэтому его следует снабдить оператором STOP.

Если все же есть необходимость продолжить вычисления при возникновении исключения, то нужно, применив SETCONTROLFPQQ, его разрешить. Разрешенное исключение прерываний не порождает, поэтому информацию о месте возникновения и характере исключения можно получить, анализируя результат, возвращенный GETSTATUSFPQQ. По результатам анализа может быть принято решение о завершении процесса.

2.4.2. Обработка математических исключений

Математические исключения с плавающей точкой, возникающие, например, при вызове функций SQRT или LOG, обрабатываются подпрограммой MATHERRQQ. По умолчанию она автоматически включается в код компоновщиком, и, если возникает математическое исключение, она реагирует заданным по умолчанию образом. Если нужно изменить поведение подпрограммы при обработке одного или нескольких математических исключений, необходимо написать свою версию MATHERRQQ. Причем созданная подпрограмма может после обработки исключения передать управление в процедуру, в которой исключение возникло.

С математическими исключениями связаны определенные в модуле msflib.f90 и приведенные в табл. 2.6 константы.

Таблица 2.6. Константы, связанные с математическими исключениями

Имя ошибки	*Значение*	*Описание исключения*
MTH$E_DOMAIN	1	Аргумент имеет недопустимое значение, например LOG(-1.0)
MTH$E_SINGULARITY	2	Вырожденное значение аргумента, например LOG(0.0)
MTH$E_OVERFLOW	3	Переполнение
MTH$E_UNDERFLOW	4	Исчезновение порядка
MTH$E_TLOSS	5	Полная потеря точности
MTH$E_PLOSS	6	Частичная потеря точности

Ошибка MTH$E_DOMAIN означает, что аргумент функции лежит за границами ее области определения, например SQRT(-1.0). Аргумент вырожден, если для него результат функции не определен, например LOG10(0.0). Переполнение и исчезновение имеют тот же смысл, что и соответствующие исключения с плавающей точкой. Полной потери точности соответствует денормализация, частичной – приближение.

Синтаксис создаваемого обработчика математических ошибок:

SUBROUTINE MATHERRQQ(*name*, *length*, *info*, *retcode*)

Входные данные: *name*, *length*, *info*.

Выходные данные: *retcode*.

name – имя встроенной математической функции, в которой возникло исключение.

length – длина имени *name* в символах; тип *length* – INTEGER(2).

info – переменная типа MTH$E_INFO.

retcode – возвращаемый MATHERRQQ код завершения; если он равен нулю, то произойдет завершение процесса, в котором возникла математическая ошибка, в противном случае вычисления будут продолжены. Тип *retcode* – INTEGER(2).

Тип MTH$E_INFO определен в модуле msflib.f90. Он позволяет подпрограмме MATHERRQQ распознать тип аргументов математической функции, в которой возникла ошибка, их число (1 или 2) и тип возвращаемого функцией результата.

```
STRUCTURE /MTH$E_INFO/          ! Структура, используемая в MATHERRQQ
 INTEGER*4 ERRCODE              ! Код ошибки (1-6)
 INTEGER*4 FTYPE                ! Тип результата
 UNION
  MAP
   REAL*4 R4ARG1                ! Входной: первый параметр функции
   REAL*4 R4ARG2                ! Входной: второй параметр функции (если есть)
   REAL*4 R4RES                 ! Выходной: тип результата
  END MAP
  MAP
   REAL*8 R8ARG1                ! Входной: первый параметр функции
   REAL*8 R8ARG2                ! Входной: второй параметр функции (если есть)
   REAL*8 R8RES                 ! Выходной: тип результата
  END MAP
  MAP
   COMPLEX*8 C8ARG1             ! Входной: первый параметр функции
   COMPLEX*8 C8ARG2             ! Входной: второй параметр функции (если есть)
   COMPLEX*8 C8RES              ! Выходной: тип результата
  END MAP
  MAP
   COMPLEX*16 C16ARG1           ! Входной: первый параметр функции
   COMPLEX*16 C16ARG2           ! Входной: второй параметр функции (если есть)
   COMPLEX*16 C16RES            ! Выходной: тип результата
  END MAP
 END UNION
END STRUCTURE
```

С типом математической функции связаны заданные в модуле msflib.f90 константы стандартного целого типа:

- TY$REAL4 = 1;

- TY$REAL8 = 2;
- TY$CMPLX8 = 3;
- TY$CMPLX16 = 4.

Компонент FTYPE структуры MTH$E_INFO может принимать значения одной из этих констант.

Компоненты присутствующего в структуре объединения (UNION), например R8ARG1, R8ARG2 и R8RES, передают в MATHERRQQ значения аргументов математической функции (R8ARG1 и R8ARG2), при которых возникло исключение, и позволяют передать в процедуру, в которой возникла математическая ошибка, результат, если он определяется в MATHERRQQ, неблагополучной математической функции.

Пример. Подпрограмма MATHERRQQ вызывается при попытке извлечь квадратный корень из отрицательного числа и при вызове DLOG(0.0_8).

```
program maer
 real(4) :: x = -1.0, y
 real(8) :: f = 0.0_8, z
 y = sqrt(x)                          ! Поскольку возникают математические исключения,
 z = real(y, kind = 8) * dlog(f)      ! будет вызвана подпрограмма MATHERRQQ
 print *, z
end program maer

subroutine matherrqq(name, length, info, retcode)
 use msflib; use text_transfer        ! Обработчик математических исключений
 integer(2) :: length, retcode
 character(length) :: name
 type(mth$e_info) info
 print *, trim(ru_doswin('Ошибка в функции', .false.)), ' ', name
 print *, trim(ru_doswin('Тип ошибки: ', .false.)), info%errcode
 select case(info%ftype)
 case(ty$real4 : ty$real8)
   call inform('REAL(8)'); read *, info%r8res
 case(ty$cmplx8 : ty$cmplx16)
   call inform('COMPLEX(8)'); read *, info%c16res
 end select
 retcode = 1

contains

 subroutine inform(typ)
  character(*) :: typ
  print *, trim(ru_doswin('Тип результата:', .false.)), ' ', typ
  print *, trim(ru_doswin('Введите желаемый результат функции:', .false.)), ' ', typ
 end subroutine inform
end subroutine matherrqq
```

3. ВСПОМОГАТЕЛЬНЫЕ ПРОЦЕДУРЫ

3.1. МАШИННЫЕ КОНСТАНТЫ

3.1.1. Целые машинные константы

Целочисленные машинные константы, определяющие используемую компьютером арифметику при работе с типом INTEGER(4) – параметры моделей целых и вещественных чисел, возвращаются функцией IMACH и встроенными числовыми функциями (ВЧФ) Фортрана (табл. 3.1). Напомним, что ВЧФ введены в Фортран стандартом 1990 г.

Таблица 3.1. Целые машинные константы

IMACH	*Возвращаемая константа*	*Значение*	*ВЧФ*
IMACH(1)	Число бит, используемых для представления целого числа типа INTEGER(4)	32	-
IMACH(2)	Число байт, используемых для представления целого числа типа INTEGER(4)	4	KIND
IMACH(3)	Основание системы счисления, используемой для представления целых чисел	2	RADIX
IMACH(4)	Q – число цифр в целом числе типа INTEGER(4) по основанию 2	31	DIGITS
IMACH(5)	$2^Q - 1$ – наибольшее целое типа INTEGER(4)	2147483647	HUGE
IMACH(6)	Основание системы счисления, используемой для представления вещественных чисел	2	RADIX
IMACH(7)	P – число цифр в мантиссе нормализованного вещественного числа типа REAL(4) по основанию 2	24	DIGITS
IMACH(8)	E_{min} – минимальный порядок в двоичном представлении вещественных чисел типа REAL(4)	-125	MINEXPO-NENT
IMACH(9)	E_{max} – максимальный порядок в двоичном представлении вещественных чисел типа REAL(4)	128	MAXEXPO-NENT
IMACH(10)	P – число цифр в мантиссе нормализованного вещественного числа типа REAL(8) по основанию 2	53	DIGITS

IMACH(11)	E_{min} – минимальный порядок в двоичном представлении вещественных чисел типа REAL(8)	-1021	MINEXPO-NENT
IMACH(12)	E_{max} – максимальный порядок в двоичном представлении вещественных чисел типа REAL(8)	1024	MAXEXPO-NENT

3.1.2. Вещественные машинные константы

Вещественные машинные константы арифметики с одинарной точностью возвращаются функцией AMACH, а двойной – DMACH.

Стандарт IEEE определяет *совершенный* NaN (Not a Number – не число) как результат некорректных операций, например 0/0. Вызов AMACH(6) возвращает *сигнальный* NaN. Если формат сигнального NaN не поддерживается, то AMACH(6) и DMACH(6) вернут совершенный NaN. Если компьютер не поддерживает NaN, то результатом AMACH(6) будет специальное значение, близкое к AMACH(2) или, если работа ведется с REAL(8), к DMACH(2). На компьютерах, не имеющих специального представления бесконечности, AMACH(7) и DMACH(7) возвращают те же значение, что и AMACH(2) и DMACH(2).

Возвращаемые функциями AMACH и DMACH величины перечислены в табл. 3.2. В ее последнем столбце приведены ВЧФ, вычисляющие те же машинные константы.

Таблица 3.2. Вещественные машинные константы

AMACH DMACH	*Возвращаемая константа*	*Значение*	*ВЧФ*
AMACH(1) DMACH(1)	$2^{E_{min}-1}$ – наименьшее нормализованное положительное вещественное число	1.175494E-38 2.225073858507201E-308	TINY
AMACH(2) DMACH(2)	$(1-2^{-P})b^{E_{max}}$ – наибольшее число	3.402823E+38 1.797693134862316E+308	HUGE
AMACH(3) DMACH(3)	2^{-P} – наименьшее относительное расстояние между числами	5.960464E-08 1.110223024625157E-016	-
AMACH(4) DMACH(4)	2^{1-P} – наибольшее относительное расстояние между числами (машинная точность $\varepsilon_м$)	1.192093E-07 2.220446049250313E-016	EPSILON
AMACH(5) DMACH(5)	LOG10(2)	3.010300E-01 3.010299956639812E-001	-

AMACH(6) DMACH(6)	NaN – сигнальное не число	1#QNAN..... 1#QNAN.....	-
AMACH(7) DMACH(7)	+∞ – положительная машинная бесконечность	1#INF..... 1#INF.....	-
AMACH(8) DMACH(8)	-∞ – отрицательная машинная бесконечность	1#INF..... 1#INF.....	-

Пример:

```
program amd
 use msimsl
 real(4) x4
 real(8) x8
 print *, 'amach(1) =', amach(1), '  tiny(x4) =', tiny(x4)
 print *, 'amach(2) =', amach(2), '  huge(x4) =', huge(x4)
 print *, 'amach(3) =', amach(3), '  1.0_4/rrspacing(huge(x4)) =', 1.0_4/rrspacing(huge(x4))
 print *, 'amach(4) =', amach(4), '  epsilon(x4) =', epsilon(x4)
 print *, 'dmach(1) =', dmach(1), '  tiny(x8) =', tiny(x8)
 print *, 'dmach(2) =', dmach(2), '  huge(x8) =', huge(x8)
 print *, 'dmach(3) =', dmach(3), '  1.0_8/rrspacing(huge(x8)) =', 1.0_8/rrspacing(huge(x8))
 print *, 'dmach(4) =', dmach(4), '  epsilon(x8) =', epsilon(x8)
 print *, 'amach(5) =', amach(5), '  dmach(5) =', dmach(5)
 print *, 'amach(6) =', amach(6), '  dmach(6) =', dmach(6)
 print *, 'amach(7) =', amach(7), '  dmach(7) =', dmach(7)
 print *, 'amach(8) =', amach(8), '  dmach(8) =', dmach(8)
end program amd
```

Результат:

```
amach(1) = 1.175494E-38                 tiny(x4) = 1.175494E-38
amach(2) = 3.402823E+38                 huge(x4) = 3.402823E+38
amach(3) = 5.960464E-08                 x4/rrspacing(huge(x4)) = 5.960465E-08
amach(4) = 1.192093E-07                 epsilon(x4) = 1.192093E-07
dmach(1) = 2.225073858507201E-308       tiny(x8) = 2.225073858507201E-308
dmach(2) = 1.797693134862316E+308       huge(x8) = 1.797693134862316E+308
dmach(3) = 1.110223024625157E-016       x8/rrspacing(huge(x8)) = 1.110223024625157E-016
dmach(4) = 2.220446049250313E-016       epsilon(x8) = 2.220446049250313E-016
amach(5) = 3.010300E-01                 dmach(5) =  3.010299956639812E-001
amach(6) = 1#QNAN.......                dmach(6) = 1#QNAN.................
amach(7) = 1#INF........                dmach(7) = 1#INF.................
amach(8) = 1#INF........                dmach(8) = 1#INF.................
```

3.1.3. Проверка на NAN

Логическая функция IFNAN (DIFNAN) возвращает .TRUE., если ее вещественный аргумент есть NaN (не число), и .FALSE. – в противном случае. Описана в [54].

Пример. Если x есть NaN, то будет выведено сообщение, иначе будет выведен x.

```
program ifna
 use msimsl
 use text_transfer                    ! Для вызова ru_doswin
 real(4) :: x
 x = amach(6)
 if(ifnan(x)) then
  ! Для вывода русского текста в DOS-окне используем ru_doswin
  print *, trim(ru_doswin('x – это NaN', .false.))
 else
  print *, 'x = ', x
 end if
end program ifna
```

Результат:

x – это NaN

3.2. ВЫВОД РЕЗУЛЬТАТОВ

3.2.1. Список, вызовы и параметры процедур

Приводимые в табл. 3.3 подпрограммы выводят на печать прямоугольную матрицу или вектор, а также управляют режимами их вывода. В табл. 3.4 и 3.5 даются соответственно синтаксис вызова и параметры подпрограмм из табл. 3.3 (кроме подпрограмм, управляющих выводом, параметры которых поясняются при их рассмотрении). Всегда указываются имена подпрограмм, принимающих параметры одинарной точности. Не указываются имена подпрограмм 2-го уровня и подпрограмм, вещественные параметры которых имеют точность REAL(8).

Таблица 3.3. Подпрограммы вывода векторов и матриц и управления выводом

Подпрограмма	*Назначение*
	Вещественная матрица
WRRRN	Вывод прямоугольной матрицы с нумерацией ее строк и столбцов
WRRRL	Вывод прямоугольной матрицы по заданному формату и с заданными именами ее строк и столбцов
	Целочисленная матрица
WRIRN	Делает то же, что и WRRRN
WRIRL	Делает то же, что и WRRRL

Комплексная матрица	
WRCRN	Делает то же, что и WRRRN
WRCRL	Делает то же, что и WRRRL и W2RRL
Подпрограммы, управляющие выводом	
WROPT	Устанавливает или запрашивает параметры, используемые при выводе матрицы
PGOPT	Устанавливает или определяет ширину и высоту страницы
UMACH	Устанавливает или запрашивает номер устройства В/В или передачи данных об ошибках

Замечание. Иногда для большей ясности изложения вместо понятия *строка матрицы* будет использоваться понятие *ряд матрицы*.

Таблица 3.4. Синтаксис вызова подпрограмм из табл. 3.3

Синтаксис вызова
Вещественная матрица
CALL WRRRN(*title, nra, nca, a, Lda, itring*)
CALL WRRRL(*title, nra, nca, a, Lda, itring, fmt, rlabel, clabel*)
Целочисленная матрица
CALL WRIRN(*title, nrmat, ncmat, mat, Ldmat, itring*)
CALL WRIRL(*title, nrmat, ncmat, mat, Ldmat, itring, fmt, rlabel, clabel*)
Комплексная матрица
CALL WRCRN(*title, nra, nca, a, Lda, itring*)
CALL WRCRL(*title, nra, nca, a, Lda, itring, fmt, rlabel, clabel*)
Подпрограммы, управляющие выводом
CALL WROPT(*iopt, iset, iscope*)
CALL PGOPT(*iopt, ipage*)
CALL UMACH(*n, nunit*)

Таблица 3.5. Параметры подпрограмм из табл. 3.4

Имена	*Смысл*	*Типы*
a	Массив формы (*Lda*, *n*), содержащий элементы матрицы *A*	REAL(4) или REAL(8); COMPLEX(4) или COMPLEX(8)
clabel	Вектор размера *nca* + 1 имен столбцов выводимой матрицы. Причем *clabel*(1) содержит имя столбца, хранящего имена строк матрицы, а *clabel*(1 + *j*) – имя столбца *j*. Размер вектора задается равным единице в двух случаях:	CHARACTER(*)

	если нужно пронумеровать столбцы цифрами 1, 2, ..., *nra*, то задается вектор *clabel*(1) = 'NUMBER'; если столбцы выводятся неименованными, то задается *clabel*(1) = 'NONE'	
fmt	Символьная строка, содержащая формат	CHARACTER(*)
itring	Параметр, задающий форму вывода матрицы	INTEGER(4)
Lda	Ведущий размер массива *a* по первому измерению	"
Ldmat	Ведущий размер массива *mat* по первому измерению	"
mat	Массив формы (*Ldmat*, *n*), содержащий элементы матрицы *MAT*	"
nca, *ncmat*	Число выводимых столбцов соответственно матриц *A* и *MAT*	"
nra, *nrmat*	Число выводимых строк (рядов) соответственно матриц *A* и *MAT*	"
rlabel	Вектор размера *nra* имен строк выводимой матрицы. Размер вектора задается равным единице в двух случаях: если нужно пронумеровать строки матрицы цифрами 1, 2, ..., *nra*, то задается вектор *rlabel*(1) = 'NUMBER'; если строки выводятся неименованными, то задается *rlabel*(1) = 'NONE'	CHARACTER(*)
title	Символьная строка, задающая заголовок таблицы. Задание *title* = ' ' подавляет вывод заголовка. Если в заголовке содержатся символы %/, то следующие за ними символы будут выводиться с новой строки. Заголовки, длина которых больше ширины страницы, переносятся автоматически	"

3.2.2. Вывод прямоугольной матрицы с нумерацией строк и столбцов

Выполняет подпрограмма WRRRN (DWRRRN):

CALL WRRRN(*title, nra, nca, a, Lda, itring*)

Входные данные: *title, nra, nca, a, Lda, itring*.

Выходные данные: нет.

Описание параметров, кроме *itring*, дано в табл. 3.5.

itring – параметр, задающий форму вывода матрицы; поясняется в табл. 3.6.

Таблица 3.6. Действие параметра itring

itring	*Что выводится*
0	Вся *nra*×*nca* -матрица
1	Верхняя треугольная часть матрицы с диагональю
2	Верхняя треугольная часть матрицы без диагонали
-1	Нижняя треугольная часть матрицы с диагональю
-2	Нижняя треугольная часть матрицы без диагонали

Обычно *itring* ≠ 0 задается для симметрических матриц.

Замечания:

1. Вывод выполняется под управлением формата. Автоматически для вывода выбирается дескриптор преобразований D, E или F, с тем чтобы вывести 4 значащих цифры для наибольшего по абсолютной величине элемента матрицы *A*. Для изменения по умолчанию установленного формата используется подпрограмма WROPT.
2. Горизонтальное центрирование, способы вывода матрицы и NaN, нумерация страниц, вывод заголовка таблицы на каждой странице и другие параметры вывода задаются WROPT.
3. По умолчанию используется страница шириной в 78 символов. Ширину страницы можно изменить, вызвав PGOPT.
4. Номер устройства вывода передается или возвращается подпрограммой UMACH.
5. Одномерный массив (вектор) может быть выведен в столбце, если задать параметр *nra*, равным длине вектора, а *nca* = 1. Если же задать *nra* = 1, а *nca* равным длине вектора, то вывод вектора будет выполнен построчно. В обоих случаях *Lda* = *nra* и *itring* = 0.
6. Вывод матриц типа REAL(8) выполняется DWRRRN.
7. Матрицы типа INTEGER(4) выводятся WRIRN, которая использует дескриптор преобразований I.
8. При выводе строки и столбцы матрицы нумеруются цифрами 1, 2, 3 и т. д.

Пример. Выводится нижняя треугольная часть симметрической 4×4-матрицы Гильберта, в которой $a_{ij} = 1/(i + j - 1)$.

```
program wrr1
 use msimsl; use text_transfer
 integer(4), parameter :: n = 4, Lda = n
 real(4) :: a(Lda, n)
 character(80) :: title = 'Нижняя треугольная часть %/ матрицы Гильберта'
 do j = 1, n                                  ! Задаем матрицу Гильберта
   a(:n, j) = (/ (1.0/real(i + j - 1), i = 1, n) /)
 end do
 ! Для вывода русского заголовка используем ru_doswin
 call wrrrn(ru_doswin(title, .false.), Lda, n, a, Lda, -1)
end program wrr1
```

Результат:

Нижняя треугольная часть матрицы Гильберта

	1	2	3	4
1	1.000			
2	0.500	0.333		
3	0.333	0.250	0.200	
4	0.250	0.200	0.167	0.143

3.2.3. Вывод прямоугольной матрицы по заданному формату и с заданными именами строк и столбцов

3.2.3.1. Вывод вещественной матрицы

Выполняет подпрограмма WRRRL (DWRRRL):

CALL WRRRL(*title, nra, nca, a, Lda, itring, fmt, rlabel, clabel*)

Все параметры являются *входными*.

Описание параметров, кроме *itring* и *fmt*, дано в табл. 3.5. Параметр *itring* описан в табл. 3.6.

fmt – символьная строка, содержащая формат вывода. Если *fmt* является пустой строкой или состоит из пробелов, то используется формат, заданный WROPT. Иначе строка *fmt* должна содержать один или более дескрипторов преобразования данных, обрамленных круглыми скобками. Например, '(F10.3)' задает формат для всей матрицы; '(2E10.3, 3F10.3)' задает дескриптор E для столбцов 1 и 2 и дескриптор F для столбцов 3, 4 и 5. Если число столбцов больше пяти, то вновь дважды используется дескриптор E10.3, а затем трижды F10.3 и т. д. Даже если матрица *A* является вещественной, можно для вывода целых частей ее элементов употребить дескриптор I. Для автоматического выбора формата используйте в строке *fmt* символ V или W. Задание *fmt* равным '(V10.4)' или '(W10.4)' означает, что автоматически будет выбираться дескриптор D, E или F (или I в случае W) с длиной поля в 10 символов и с четырьмя цифрами после десятичной точки. Причем V выводит конечные нули, а W – нет. Параметр *fmt* может содержать только дескрипторы данных D, E, F, G, I, V и W и не может включать, например, дескриптор управления X.

Форма вывода, например 2×3-матрицы, такова:

title

clabel(1)	clabel(2)	clabel(3)	clabel(4)
rlabel(1)	xxxxx	xxxxx	xxxxx
rlabel(2)	xxxxx	xxxxx	xxxxx

Наличие в заголовке и именах рядов и столбцов символов %/ вызывает перенос текста на новую строку страницы. Длинные заголовки и имена, не содержащие символов %/, переносятся автоматически.

Использование дескрипторов V и W обеспечивает автоматический выбор одного из дескрипторов Фортрана D, E, F или I. Дескриптор V или W задается в виде V*n.d* или W*n.d*, где *n* – длина поля вывода, а *d* – число выводимых десятичных знаков. Допустимые значения: для *n* – это 3, 4, ..., 40, для *d* – 1, 2, ..., *n* – 2. Если *fmt* содержит один дескриптор V или W, то выбор формата вывода выполняется в результате анализа всех элементов матрицы. Если в *fmt* число дескрипторов V и/или W больше единицы, то выбор формата вывода выполняется для каждого дескриптора V (или W) отдельно.

Замечания:

1. Справедливы замечания 2-5 из разд. 3.3.2.
2. Вывод матриц типа REAL(8) выполняется DWRRRL.
3. Матрицы типа INTEGER(4) выводятся WRIRL, которая использует дескриптор преобразований I.

Пример. Выводится 3×4-матрица A, в которой $a_{ij} = (i + j/10) * 10^{j-3}$.

```
program wrr2
 use msimsl; use text_transfer
 integer(4), parameter :: Lda = 10, nca = 4, nra = 3, itring = 0
 integer(4) :: i, j
 real(4) :: a(Lda, nca)
 character(10) :: clabel(nca + 1), rlabel(nra), fmt*8 = '(w10.6)'
 clabel = (/ '     ', 'col 1', 'col 2', 'col 3', 'col 4' /)
 rlabel = (/ 'row 1', 'row 2', 'row 3' /)
 do i=1, nra
   a(i, :) = (/ ((i + j * 0.1) * 10.0**(j – 3), j = 1, nca) /)
 end do
 ! Для вывода русских заголовка и имен столбцов и строк матрицы используем ru_doswin
 clabel = (/ 'Номер ряда', 'Столбец 1 ', 'Столбец 2 ', 'Столбец 3 ', 'Столбец 4 ' /)
 rlabel = (/ 'Ряд 1', 'Ряд 2', 'Ряд 3' /)
 do i = 1, nca + 1;          clabel(i) = ru_doswin(clabel(i), .false.); end do
 do i = 1, nra;              rlabel(i) = ru_doswin(rlabel(i), .false.); end do
 call wrrrl(ru_doswin('Вещественная матрица A', .false.), nra, nca, a, Lda, itring, fmt, rlabel, clabel)
end program wrr2
```

Результат:

Вещественная матрица А

Номер ряда	Столбец 1	Столбец 2	Столбец 3	Столбец 4
Ряд 1	0.011	0.120	1.300	14.000
Ряд 2	0.021	0.220	2.300	24.000
Ряд 3	0.031	0.320	3,300	34.000

Замечание. Если элементы матрицы вычислять по формуле $a_{ij} = (i + j/10)*10^{j-3}$, то при выводе будет выбран дескриптор E и результат будет таким:

Вещественная матрица А

Номер ряда	Столбец 1	Столбец 2	Столбец 3	Столбец 4
Ряд 1	1.100E-04	1.200E-05	1.300E-06	1.400E-07
Ряд 2	2.100E-04	2.200E-05	2.300E-06	2.400E-07
Ряд 3	3.100E-04	3.200E-05	3.300E-06	3.400E-07

3.2.3.2. Вывод комплексной матрицы

Выполняет подпрограмма WRCRL (DWRCRL):

CALL WRCRL(*title, nra, nca, a, Lda, itring, fmt, rlabel, clabel*)

Все приведенные при описании параметров WRRRL сведения применимы и для параметров WRCRL. Относительно параметра *fmt* дополнительно следует сообщить, что для комплексного числа *fmt* может содержать два дескриптора, например '(E10.3, F10.3)'. Тогда действительная часть числа будет использовать дескриптор E, а мнимая – F. Однако если *fmt* содержит один дескриптор, например '(F10.3)', то он будет использован при выводе и действительной и мнимой частей комплексного числа. Так же как и в случае вещественной матрицы, для вывода целой части действительного и/или мнимого компонента комплексного числа можно использовать дескриптор I.

Пример. Выводится комплексная 3×4-матрица *A*.

```
program wrc1
 use msimsl; use text_transfer
 integer(4), parameter :: Lda = 10, nca = 4, nra = 3, itring = 0
 integer(4) :: i, j
 complex(4) :: a(Lda, nca)
 character(10) :: clabel(nca + 1), rlabel(nra), fmt = '(w12.6)'
 do i=1, nra                                  ! Задаем элементы комплексной матрицы
   a(i, :) = (/ (cmplx(i, j) + 0.123456, j = 1, nca) /)
 end do
 ! Для вывода русских заголовка и имен столбцов и строк матрицы используем ru_doswin
 clabel = (/ 'Номер ряда', 'Столбец 1 ', 'Столбец 2 ', 'Столбец 3 ', 'Столбец 4 ' /)
 rlabel = (/ 'Ряд 1', 'Ряд 2', 'Ряд 3' /)
 do i = 1, nca + 1;          clabel(i) = ru_doswin(clabel(i), .false.); end do
 do i = 1, nra;              rlabel(i) = ru_doswin(rlabel(i), .false.); end do
 call wrcrl(ru_doswin('Комплексная матрица A', .false.), nra, nca, a, Lda, itring, fmt, rlabel, clabel)
end program wrc1
```

Результат:

Комплексная матрица A

Номер ряда	Столбец 1	Столбец 2
Ряд 1	(1.12346, 1.00000)	(1.12346, 2.00000)
Ряд 2	(2.12346, 1.00000)	(2.12346, 2.00000)
Ряд 3	(3.12346, 1.00000)	(3.12346, 2.00000)
	Столбец 3	Столбец 4
Ряд 1	(1.12346, 3.00000)	(1.12346, 4.00000)
Ряд 2	(2.12346, 3.00000)	(2.12346, 4.00000)
Ряд 3	(3.12346, 3.00000)	(3.12346, 4.00000)

3.2.3.3. Вывод целочисленной матрицы

Выполняет подпрограмма WRIRL

CALL WRIRL(*title, nrmat, ncmat, mat, Ldmat, itring, fmt, rlabel, clabel*)

Все параметры являются *входными*.

Все приведенные при описании параметров WRRRL сведения применимы и для WRIRL. Относительно параметра *fmt* дополнительно следует сообщить, что если *fmt* является пустой строкой или содержит лишь пробелы, то используется дескриптор I с шириной поля, определяемой по наибольшему элементу матрицы. Если, например, *fmt* равен '(I10)', то дескриптор I10 используется для всей матрицы. Если *fmt* равен '(2I10, 3I5)', то для столбцов 1 и 2 используется дескриптор I10, а для столбцов 3, 4 и 5 - I5. Если число столбцов больше пяти и равно, например, 10, то вновь дважды используется дескриптор I10, а затем трижды I5 и т. д. Параметр *fmt* может иметь только дескриптор I.

Пример. Вывести вектор из восьми элементов, применив для элементов с нечетными индексами дескриптор I5, а с четными – дескриптор I7.

```
program wri1
 use msimsl; use text_transfer
 integer(4), parameter :: Lda = 1, nca = 8, nra = 1, itring = 0
 integer(4) :: ia(nca)
 character(5) :: clabel(nca + 1), rlabel(1), fmt*8 = '(i5, i7)'
 ia = (/ (i, i = 1, nca) /)                    ! Задаем элементы вектора
 rlabel = 'none'                               ! Отказываемся от вывода имен строк
 ! Для вывода русских заголовка и имен элементов используем ru_doswin
 clabel = (/ '     ', 'Эл. 1', 'Эл. 2', 'Эл. 3', 'Эл. 4', 'Эл. 5', 'Эл. 6', 'Эл. 7', 'Эл. 8' /)
 do i = 1, nca + 1;          clabel(i) = ru_doswin(clabel(i), .false.); end do
 call wrirl(ru_doswin('Целочисленный вектор', .false.), nra, nca, ia, Lda, itring, fmt, rlabel, clabel)
end program wri1
```

Результат:

Целочисленный вектор

Эл. 1	Эл. 2	Эл. 3	Эл. 4	Эл. 5	Эл. 6	Эл. 7	Эл. 8
1	2	3	4	5	6	7	8

3.2.4. Управление выводом

Установка и чтение параметров вывода выполняется подпрограммой WROPT:

CALL WROPT(*iopt, iset, iscope*)

Входные данные: *iopt, iscope*.

Входные/выходные данные: *iset*.

iopt – вид регулируемого параметра вывода. Если *iopt* < 0, то значения *iset* и *iscope* присваиваются внутренним переменным WROPT. Если *iopt* > 0, *iset* является выходным параметром и возвращает, если *iscope* = 0, локальные значения, соответствующие *iopt*, или глобальные, если *iscope* = 1.

iset – устанавливает или возвращает значение параметра, заданного *iopt*; *iset* является входным, если *iopt* < 0, выходным, если *iopt* > 0, не адресуется, если *iopt* = 0.

iscope – область действия параметра, заданного *iopt* и *iset*. (Если *iopt* = 0, то параметр *iscope* не адресуется.) Если *iscope* = 0, то установка имеет действие только на следующий вызов WR*** (локальная установка); если *iscope* = 1, то установка активна до тех пор, пока не будет изменена следующим вызовом WROPT с *iscope* = 1 (глобальная установка).

Замечание. Вызов WROPT избыточен, если подходят заданные по умолчанию параметры печати.

Расшифровка *iopt* дана в табл. 3.7. Варианты задания *iset* для различных значений *iopt* приведены в табл. 3.8. Заданные по умолчанию значения *iset* перечислены в табл. 3.9.

Таблица 3.7. Значения и смысл параметра iopt

iopt	*Что регулируется*
±1	Выравнивание матрицы (по центру страницы или по ее левой границе)
±2	Способ вывода больших матриц
±3	Нумерация страниц
±4	Способ вывода NaN, отрицательной и положительной машинной бесконечности
±5	Способ вывода заголовка таблицы
±6	Задаваемый по умолчанию формат вывода вещественных и комплексных матриц
±7	Расстояние между столбцами
±8	Максимальный горизонтальный размер поля для вывода имен рядов матрицы
±9	Отступ текста при переносе имени ряда матрицы на строку продолжения
±10	Ширина в символах зоны, в которой может быть выполнен перенос имени ряда матрицы
±11	Максимальный горизонтальный размер поля для выводя имен столбцов
±12	Ширина в символах зоны, в которой может быть выполнен перенос имени столбца
±13	Ширина в символах зоны, в которой может быть выполнен перенос заголовка таблицы
±14	Надпись в левом верхнем углу таблицы, в которую выводится матрица; оказывает действие только при вызове WR**N
±15	Параметр, определяющий, будет ли выводиться пустая строка перед заголовком таблицы или нет; действует при вызове WR**N

±16	Если имя ряда матрицы занимает более одной строки страницы, то вывод элементов соответствующего ряда матрицы можно выполнить в первой строке, занимаемой именем ряда, или в его последней строке
0	Восстанавливает текущие значения параметров, присваивая им величины, которые они имели во время последнего вызова WROPT с *iscope* = 1

Таблица 3.8. Значения параметра iset

iopt	*iset*	*Что задается*
±1	0 1	Матрица выравнивается по левому краю матрица выводится по центру страницы
±2	0 *m*	Текущий ряд выводится полностью, затем с новой строки страницы начинается вывод следующего ряда; при необходимости при выводе ряда осуществляется переход на следующую строку страницы Пусть $n1$ – число столбцов, начиная от столбца с номером 1, которое может разместиться по ширине страницы. Тогда будут выведены столбцы с номерами от 1 до $n1$ для рядов с номерами от 1 до m (m – положительное целое число). Пусть теперь $n2$ – число столбцов, начиная от столбца с номером $n1 + 1$, которое может разместиться по ширине страницы. Тогда будут выведены столбцы с номерами от $n1 + 1$ до $n1 + n2$ для рядов с номерами от 1 до m. Далее по таким правилам выводятся все оставшиеся строки матрицы
±3	-2 -1 0 *k*	Вывод начинается со следующей строки страницы. Страницы не нумеруются Страницы нумеруются. При каждом новом вызове подпрограммы WR*** вывод начинается с новой страницы Страницы нумеруются. После первого вызова WR*** вывод начинается с новой страницы. Работа WR*** завершается вызовом WROPT и записью в *iset* значения k, равного числу строк, выведенных на последней странице. И если *iscope* = 1 и не изменено значение *iset*, то новый вызов WR*** продолжит вывод с текущей страницы Страницы нумеруются. Если k меньше длины страницы *ipage* (см. PGOPT) и на текущей странице выведено k строк, то $ipage - k$ строк будет выведено, прежде чем начнется новая страница. Если $k \geq ipage$, то после первого вызова WR*** выполнится переход на новую страницу Работа WR*** завершается вызовом WROPT и записью в *iset* значения k, равного числу строк, выведенных на последней странице
±4	0 1 2	NaN выводится как последовательность точек; отрицательная машинная бесконечность – как последовательность минусов, а положительная машинная бесконечность – как последовательность плюсов NaN выводится как последовательность пробелов; отрицательная и положительная машинные бесконечности выводятся, как и при *iset* = 0 NaN выводится как NaN; отрицательная машинная бесконечность – как -Inf; а положительная машинная бесконечность – как Inf

	3	NaN выводится как последовательность пробелов; отрицательная и положительная машинные бесконечности выводятся, как и при *iset* = 2
±5	0 1	Заголовок таблицы появляется только на первой странице Заголовок таблиц появляется на всех страницах
±6	0 1 2 3 4 5	Формат W10.4 Формат W12.6 Формат 1PE12.5 Формат V*n*.4 (способ определения *n* пояснен в разд. 3.3.3.1) Формат V*n*.6 Формат 1PE*n.d*, где $n = d + 7$, а $d + 1$ – максимальное число значащих цифр
±7	*k*1	Число пробелов слева от столбца; $0 \le k1 \le 5$
±8	*k*2	Максимальный размер поля под имена строк матрицы; если *k*2 = 0, то действует размер, задаваемый по умолчанию
±9	*k*3	Число пробелов, после которых начинается текст на строке продолжения имени ряда матрицы; $0 \le k3 \le 10$ (перенос текста в имени ряда вызывают, в частности, символы %/; также перенос выполняется автоматически в случае длинных имен строк)
±10	*k*4	Ширина в символах зоны, в которой может быть выполнен перенос имени ряда матрицы на другую строку страницы; перенос выполняется либо если в этой зоне встречен пробел, либо если длина имени без пробелов превысила допустимое значение $0 \le k4 \le 50$. Если *k*4 = 0, то используется значение по умолчанию
±11	*k*5	Максимальный горизонтальный размер поля для вывода имен столбцов. Не уместившиеся символы переносятся на другую строку страницы. Если *k*5 = 0, то используется значение по умолчанию
±12	*k*6	Ширина в символах зоны, в которой может быть выполнен перенос имени столбца матрицы на другую строку страницы; перенос выполняется либо если в этой зоне встречен пробел, либо если длина имени без пробелов превысила допустимое значение $0 \le k6 \le 50$. Если *k*6 = 0, то используется значение по умолчанию
±13	*k*7	Ширина в символах зоны, в которой может быть выполнен перенос заголовка таблицы; $1 \le k7 \le 50$
-14	0 1	Надпись в левом верхнем углу таблицы, в которую отображается матрица, не выводится Выводится надпись Component, если печатается вектор (построчно или в столбик); выводится надпись Row/Column, если печатается матрица, в которой и число рядов, и число столбцов больше единицы
-15	0 1	Если вывод продолжается на текущей странице, то перед заголовком таблицы перед каждым вызовом WR**N выводится пустая строка Перед заголовком таблицы перед каждым вызовом WR**N пустая строка не выводится

±16	0	Элементы матрицы выводятся в случае переноса имени ряда матрицы в последней строке имени, если для *iopt* = 2 возвращается *iset* > 0;
	1	Элементы матрицы выводятся в первой строке имени ряда матрицы

Таблица 3.9. Заданные по умолчанию значения iset

iopt	*iset*	*Смысл значения, заданного по умолчанию*
1	0	Матрица выравнивается по левому краю
2	10^6	Число подряд выводимых рядов матрицы равно 1000000
3	-2	Страницы не нумеруются
4	2	NaN выводится как NaN; отрицательная машинная бесконечность – как -Inf; а положительная машинная бесконечность – как Inf
5	0	Заголовок таблицы появляется только на первой странице
6	3	Формат V*n*.4
7	2	Число пробелов между столбцами *k*1 = 2
8	0	По умолчанию максимальный размер поля под имена строк матрицы *maxrlw* = 2 * *ipagew*/3, если матрица имеет один столбец, и *maxrlw* = = *ipagew*/4 – в противном случае, где *ipagew* – ширина страницы в символах (текущая ширина страницы *ipagew* задается PGOPT)
9	3	Число пробелов, после которых начинается текст на строке продолжения имени ряда матрицы *k*3 = 3
10	0	Ширина зоны, в которой может быть выполнен перенос имени ряда матрицы на другую строку страницы *k*4 = *maxrlw*/3
11	0	По умолчанию максимальный горизонтальный размер поля для вывода имен столбцов *maxclw* = min{max(*nw* + *nw*/2, 15), 40} для целочисленных и вещественных матриц, где *nw* – ширина поля, определяемая форматом, используемым при выводе конкретного столбца матрицы; *maxclw* = min{max(*nw* + *nw*/2, 15), 83} для комплексных матриц, где *nw* – ширина двух полей, определяемая форматом, используемым при выводе конкретного столбца матрицы, плюс 3
±12	0	Ширина зоны, в которой может быть выполнен перенос имени столбца матрицы на другую строку страницы; *k*6 = *maxclw*/3 символов
13	10	Ширина в символах зоны, в которой может быть выполнен перенос заголовка таблицы; *k*7 = 10
14	0	Надпись в левом верхнем углу таблицы, в которую отображается матрица, не выводbтся
15	0	Перед заголовком таблицы перед каждым вызовом WR**N выводится пустая строка
16	0	Элементы матрицы выводятся в случае переноса имени ее ряда в последней строке имени

Пример 1. Выводится 3×4-матрица, в которой $a_{ij} = i + j/10$. Первоначально матрица выводится по центру листа, затем в результате локального действия *iopt* = -1 и *iset* = 0 выравнивается по левому краю, а вслед вновь выводится по центру.

```
program wop1
 use msimsl, nouse => iset
 use text_transfer
 integer(4), parameter :: Lda = 10, nca = 4, nra = 2, itring = 0
 integer(4) :: i, iopt, iscope, iset, j
 real(4) :: a(Lda, nca)
 do j = 1, nca                          ! Задаем матрицу A
   a(:nra, j) = (/ (real(i) + real(j)/10.0, i = 1, nra) /)
 end do
 ! Для вывода русских заголовков в DOS-окне используем ru_doswin
 iopt = -1; iset = 1                    ! Вывод по центру
 iscope = 1                             ! Область действия – вся программа
 call wropt(iopt, iset, iscope)
 call wrrrn(ru_doswin('Вывод по центру', .false.), nra, nca, a, Lda, itring)
 iopt = -1; iset = 0            ! Выравнивание по левому краю
 iscope = 0                     ! Область действия локальная (на один вызов WRRRN)
 call wropt(iopt, iset, iscope)
 call wrrrn(ru_doswin('Выравнивание по левому краю', .false.), nra, nca, a, Lda, itring)
 call wrrrn(ru_doswin('Вновь вывод по центру', .false.), nra, nca, a, Lda, itring)
end program wop1
```

Результат:

Вывод по центру

	1	2	3	4
1	1.100	1.200	1.300	1.400
2	2.100	2.200	2.300	2.400

Выравнивание по левому краю

	1	2	3	4
1	1.100	1.200	1.300	1.400
2	2.100	2.200	2.300	2.400

Вновь вывод по центру

	1	2	3	4
1	1.100	1.200	1.300	1.400
2	2.100	2.200	2.300	2.400

Замечание. В ссылке на модуль msimsl.mod присутствует переименование *nouse* => *iset*. Оно позволяет избежать повторного использования имени *iset*, поскольку это же имя содержит и названный модуль.

Пример 2. Вывести 40×9-матрицу, в которой $a_{ij} = i + j/10$.

```
program wop2
 use msimsl, nouse => iset
 integer(4), parameter :: Lda = 40, nca = 9, nra = Lda, itring = 0
 integer(4) :: i, iopt, iscope, iset, j
 real(4) :: a(Lda, nca)
 character(80) :: clabel(nca + 1), rlabel(nra), fmt = '(w12.6)', title, st*2
 title = 'Имя таблицы'
 rlabel = 'Длинное имя очередной строки'
 clabel = 'Имя столб.'; clabel(1) = "`
 do i = 1, nca                          ! Формируем имена столбцов
   write(st, '(i2)') i
   clabel(i + 1) = trim(clabel(i + 1)) // st
 end do
 do j = 1, nca                          ! Формируем матрицу A
   a(:nra, j) = (/ (real(i) + real(j)/10.0, i = 1, nra) /)
 end do
 open(1, file = 'a.txt')                ! Подсоединяем файл к устройству 1
 call umach(-2, 1)                      ! Вывод выполняется в файл a.txt
 ! Выводим по 3 ряда, а далее вновь выводим заголовки столбцов
 iopt = -2; iset = 2; iscope = 1; call wropt(iopt, iset, iscope)
 ! Страницы нумеруются
 iopt = -3; iset = 0; iscope = 1; call wropt(iopt, iset, iscope)
 ! Заголовок таблицы выводится на каждой странице
 iopt = -5; iset = 1; iscope = 1; call wropt(iopt, iset, iscope)
 ! Число пробелов между столбцами равно трем
 iopt = -7; iset = 3; iscope = 1; call wropt(iopt, iset, iscope)
 ! Текст имени ряда на строке продолжения начинается без отступов
 iopt = -9; iset = 0; iscope = 1; call wropt(iopt, iset, iscope)
 ! Ширина зоны переноса имени ряда равна 10
 iopt = -10; iset = 10; iscope = 1; call wropt(iopt, iset, iscope)
 ! Элементы матрицы выводятся в первом ряду имени ряда
 iopt = -16; iset = 1; iscope = 1; call wropt(iopt, iset, iscope)
 ! Вывод матрицы
 call wrrrl(title, nra, nca, a, Lda, itring, fmt, rlabel, clabel)
end program wop2
```

Часть файла a.txt:

```
1
                                Имя таблицы
                  Имя столб. 1   Имя столб. 2   Имя столб. 3   Имя столб. 4
 Длинное имя          1.1            1.2            1.3            1.4
 очередной строки
 Длинное имя          2.1            2.2            2.3            2.4
 очередной строки
```

	Имя столб. 5	Имя столб. 6	Имя столб. 7	Имя столб. 8
Длинное имя очередной строки	1.5	1.6	1.7	1.8
Длинное имя очередной строки	2.5	2.6	2.7	2.8

3.2.5. Управление размером страницы

Установка и чтение ширины и длины страницы выполняются подпрограммой PGOPT:

CALL PGOPT(*iopt, ipage*)

Входные данные: *iopt*.

Входные/выходные данные: *ipage*.

iopt – вид регулируемого параметра страницы. Если *iopt* = ±1, то устанавливается или запрашивается ширина страницы; если *iopt* = ±2, то – ее высота.

ipage – ширина страницы в символах или ее высота в строках. Параметр *ipage* входной, если *iopt* < 0, и выходной, если *iopt* > 0; принимает значения 10, 11, ...

Пример. Вывести заданные по умолчанию ширину и высоту страницы, а затем распечатать 3×4-матрицу, в которой $a_{ij} = i + j/10$, на странице шириной в 20 символов.

```
program pgo1
 use msimsl
 integer(4), parameter :: Lda = 10, nca = 4, nra = 3, itring = 0
 integer(4) :: i, iopt, ipage, j, ipw, iph
 real(4) :: a(Lda, nca)
 do j = 1, nca                          ! Задаем матрицу A
   a(:nra, j) = (/ (real(i) + real(j)/10.0, i = 1, nra) /)
 end do
 ! Узнаем заданные по умолчанию ширину и высоту страницы
 call pgopt(1, ipw)                     ! Ширина
 call pgopt(2, iph)                     ! Высота
 print *, 'ipw = ', ipw, '; iph = ', iph
 iopt = -1; ipage = 20                  ! Ширина страницы – 20 символов
 call pgopt(iopt, ipage)
 call wrrrn ('A', nra, nca, a, Lda, itring)   ! Вывод матрицы A
end program pgo1
```

Результат:

```
ipw =      78; iph =      60
                A
                1          2
   1        1.100      1.200
   2        2.100      2.200
   3        3.100      3.200
                3          4
   1        1.300      1.400
   2        2.300      2.400
   3        3.300      3.400
```

3.2.6. Задание номера устройства В/В

Номер устройства В/В, с которым работают подпрограммы WR***, задается или опрашивается подпрограммой UMACH. Также UMACH задает устройство, на которое направляются сообщения о возникших при вызовах процедур IMSL ошибках. Если UMACH направляет В/В на устройство, отличное от заданного по умолчанию, то к нему необходимо подсоединить внешний файл подходящим оператором OPEN. Вызов UMACH:

CALL UMACH(*n, nunit*)

Входные данные: *n*.

Входные/выходные данные: *nunit*.

n – вид задаваемого или опрашиваемого устройства В/В.

nunit – номер устройства; является входным, если $n > 0$, и выходным, если $n < 0$.

Параметр *n* влечет, если он равен:

- 1, запись в *nunit* номера устройства ввода данных;
- 2, запись в *nunit* номера устройства вывода данных;
- 3, запись в *nunit* номера устройства ввода ошибок.

Параметр *n* определяет *nunit*, если он равен:

- -1, как устройство ввода данных;
- -2, как устройство вывода данных;
- -3, как устройство вывода ошибок.

Пример. Вывести в файл a.txt заданные по умолчанию номера устройств В/В. Определить также устройство 10 как устройство вывода ошибок, подсоединив к нему файл error.txt. Вывести затем в него ошибку, возникающую при неверном задании аргумента функции AMACH.

```
program uma1
 use msimsl
```

```
integer(4) :: input, iout, ierr
real(4) :: x
! Узнаем заданные по умолчанию номера устройства В/В и вывода ошибок
call umach(1, input)                    ! Номер устройства ввода
call umach(2, iout)                     ! Номер устройства вывода данных
call umach(3, ierr)                     ! Номер устройства вывода ошибок
open(10, file = 'a.txt')
call umach(-2, 10)                      ! Направим вывод данных в файл a.txt
write(10, '(1x, 3(a, i2))') 'input =', input, '; iout =', iout, '; ierr =', ierr
open(11, file = 'error.txt')
call umach(-3, 11)                      ! Направим вывод ошибок в файл error.txt
x = amach(0)                            ! Ошибка при вызове AMACH
end program uma1
```

Состав файла a.txt:

```
input = 5; iout = 6; ierr = 6
```

Состав файла error.txt:

```
*** TERMINAL ERROR 5 from AMACH. The argument must be between 1 and 8
***        inclusive. N = 0
```

3.3. ОБРАБОТКА ОШИБОК

3.3.1. Виды ошибок IMSL

Возникающие при выполнении процедур IMSL ошибки различаются на *завершающие* и *информационные*. Каждой информационной ошибке соответствует тип и код. *Тип,* или *уровень,* ошибки – это число от единицы до четырех. Причем чем серьезнее ошибка, тем выше ее тип. Завершающие ошибки относятся к 5-му типу. Кроме того, процедуры IMSL могут генерировать сообщения об ошибках, которые не снабжены кодами и не относятся к какому-либо типу.

Завершающие ошибки часто связаны с бессмысленными входными данными. Например, такая ошибка возникнет, если передать в процедуру IMSL параметр $n = -1$, где n – число уравнений. Поскольку работа с таким числом уравнений невозможна, то эта ошибка, как и любая иная завершающая ошибка, вызовет останов программы, предваряемый одним или несколькими сообщениями. Впрочем, применив ERSET, можно попытаться продолжить вычисления даже при возникновении завершающей ошибки.

После возникновения информационной ошибки программа генерирует ее код, который можно прочитать и программно обработать. Это позволяет не завершать программу даже в случае серьезной информационной ошибки, а продолжить вычисления по ветви, в которой, с одной стороны, подобного рода ошибка не возникнет, а с другой – будет получен требуемый результат.

Разумеется, подобные ветви должны быть предусмотрены программистом. Коды информационных ошибок печатаются при выводе их сопровождающих сообщений.

3.3.2. Примеры ошибок IMSL

Пример 1. Отключается останов программы при ошибках IMSL любого уровня и задаются данные, при которых в LSARG возникает ошибка уровня 4, а в LINRG – уровня 5.

```
program ex1
 use msimsl
 integer(4), parameter :: n = 2
 real(4) :: a(n, n), ainv(n, n), b(n), x(n)
 ! a – массив, задающий матрицу A
 a = reshape((/          2.0,    -3.0,        &
                         2.0,    -3.0 /),     shape = (/ n, n /))
 b = (/ 1.0, 2.0 /)
 ! Включаем печать и отключаем останов программы при ошибках любого уровня
 call erset(0, 1, 0)
 ! Возникнет ошибка уровня 4; ее причина – вырожденность матрицы A
 call Lsarg(n, a, n, b, 1, x)
 ! Внесем ошибку, задав ведущий размер массива a, равным -1
 call Linrg(-1, a, n, ainv, n)                ! Возникнет ошибка уровня 5
 ! Этот оператор будет выполнен, несмотря на ранее возникшую ошибку уровня 5
 print *, 'Last program line.'
end program ex1
```

Результат:

```
*** FATAL ERROR 2 from LSARG. The input matrix is singular.  Some of
***       the diagonal elements of the upper triangular matrix U of the
***       LU factorization are close to zero.

*** TERMINAL ERROR 1 from LINRG. The order of the matrix must be positive
***       while N = – 1 is given.
Last program line.
```

Пример 2. При вычислении двойного интеграла

$$\int_0^1\int_0^1 (x+y)dxdy = \int_0^1 g(y)dy\text{, где } g(y) = \int_0^1 (x+y)dx,$$

процедурами QDAG и QDAGS возникает ошибка (из-за неверного задания *errrel*) и выполняется ее трассировка. Приводимые при трассировке программы перечисляются в порядке, обратном их вызову.

```
program ex2
 use msimsl
 integer(4), parameter :: n = 2
```

```
 real(4) :: a, b, errabs, errest, errrel, result
 real(4), external :: g
 a = 0.0; b = 1.0                          ! Параметры интегрирования
 errabs = 0.0; errrel = 0.001
 ! Внешний интеграл
 call qdags(g, a, b, errabs, errrel, result, errest)
 print *, result, errest
end program ex2

real(4) function g(argy)
 real(4) :: argy, c, d, errabs, errest, errrel, y
 real(4), external :: f
 integer(4) :: irule
 common /comy/ y
 y = argy; c = 0.0; d = 1.0                ! Параметры интегрирования
 errabs = 0.0; errrel = -0.001             ! Вносим ошибку, задавая errrel < 0
 irule = 1
 ! Внутренний интеграл; g – результирующая переменная
 call qdag(f, c, d, errabs, errrel, irule, g, errest)
end

real(4) function f(x)
 real(4) :: x, y
 common /comy/ y
 f = x + y
end function f
```

Результат:

```
*** TERMINAL ERROR 4 from Q2AG. The relative error desired ERRREL =
***        -1.000000E-03. It must be at least zero.
Here is a traceback of subprogram calls in reverse order:
Routine name              Error type Error code
---------------           ---------- ----------
Q2AG                           5     4                (Called internally)
QDAG                           0     0
Q2AGS                          0     0                (Called internally)
QDAGS                          0     0
USER                           0     0
```

3.3.3. Изменение характера реагирования на информационные ошибки

Информационные ошибки процедур IMSL по своей серьезности разделяются на 5 уровней (типов). Каждому уровню соответствуют приведенные в табл. 3.10 заданные по умолчанию действия по выводу сообщений и остановке программы.

Таблица 3.10. Уровни серьезности информационных ошибок

Уровень	*Вид*	*Описание*	*Сообщения*	*Останов*
1	Замечание	Тривиальная ошибка или информация о ходе вычислений	Нет	Нет
2	Внимание	Сообщения о происходящих в процессе вычислений событиях	"	"
3	Предупреждение	Означает, что возникли условия, которые, возможно, требуют вмешательства пользователя с целью нормализации вычислений. Например, точность одного из результатов недостаточна, но точность других возвращаемых данных удовлетворительна. Впрочем, подчас вмешательства пользователя не требуется и вычисления могут быть продолжены	Да	"
4	Критическая	Возникшие условия таковы, что без их изменения вычисления невозможны	"	Да
5	Завершающая	Ошибки входных данных или ошибки, которые нельзя точно классифицировать средствами Фортрана. Наиболее верное средство исправления – проверка параметров процедур на их соответствие приведенным в документации требованиям. Особое внимание следует уделить проверке порядка следования параметров и их типов	"	"

Пользователь может изменить заданные по умолчанию действия, связанные с выводом сообщений и остановом программы, при возникновении ошибки заданного уровня (типа). Для этих целей служит подпрограмма

CALL ERSET(*iersvr, ipact, isact*)

Все ее параметры являются *входными* и имеют тип INTEGER(4).

iersvr – задание уровня ошибки. Если *iersvr* = 0, то установки ERSET распространяются на уровни 1-5. Если *iersvr* – число между 1 и 5, установки ERSET окажут воздействие на заданный *iersvr* уровень серьезности ошибки.

ipact – задает действия, связанные с выводом сообщений об ошибках. Если *ipact* равен

- -1, то текущие установки не меняются;
- 0, то сообщения об ошибках не выводятся;
- 1, то сообщения об ошибках выводятся;
- 2, то восстанавливаются заданные по умолчанию установки.

isact – задает действия, связанные с остановкой программы. Если *isact* равен

- -1, то текущие установки не меняются;
- 0, то программа при возникновении ошибки продолжает работу;
- 1, то программа при возникновении ошибки прекращает работу;
- 2, то восстанавливаются заданные по умолчанию установки.

Установки ERSET распространяются только на процедуры библиотеки IMSL. Дополнительную информацию о процедуре ERSET см. в [20].

Пример:

```
call erset(3, 0, -1)      ! Отключаем вывод предупреждений
call erset(3, -1, 1)      ! Программа при возникновении предупреждения прекратит работу
call erset(0, 1, -1)      ! Будут выводиться сообщения о всех ошибках
call erset(0, 2, 2)       ! Восстанавливаем все заданные по умолчанию установки
```

3.3.4. Использование системы обработки ошибок

Функция типа INTEGER(4)

icode = IERCD( )

возвращает код последней, возникшей при вызове процедур IMSL информационной ошибки.

Функция типа INTEGER(4)

itype = N1RTY(1)

возвращает уровень серьезности последней, возникшей при вызове процедур IMSL ошибки.

Значения *itype*, равные

- 1, 2, 4 и 5, означают уровень серьезности ошибки;
- 3 и 6, соответствуют предупреждениям; при этом уровень серьезности ошибки равен трем. Если *itype* = 3, то ошибка является информационной (IERCD( ) /= 0); если *itype* = 6, то ошибка – неинформационная (IERCD( ) == 0).

Дополнительно о процедурах обработки ошибок IMSL см. в [20].

Пример. Если в результате R^TR-разложения в подпрограмме LFTDS выяснится, что симметрическая матрица не является положительно определенной, то можно, применив IERCD, изменить характер вычислений.

```
...
call Lftds(n, a, Lda, fac, Ldfac)
if(iercd( ) == 2) then
  ! Если матрица A не является положительно определенной, то вызываем подпрограмму
  ! LFTSF, которая работает с неопределенными симметрическими матрицами
  call Lftsf(n, a, Lda, fac, Ldfac, ipvt)
  ...
end if
```

3.4. ПРОЦЕДУРЫ ДАТЫ И ВРЕМЕНИ

Процедуры IMSL, возвращающие дату и время, перечислены в табл. 3.11; их параметры – в табл. 3.12, а возможная информационная ошибка – в табл. 3.13.

Таблица 3.11. Процедуры, вычисляющие дату и время

Вызов/Тип	*Что вычисляет*
result = IDYWK(*iday, month, iyear*) Функция типа INTEGER(4)	День недели (число в диапазоне 1-7) для заданной даты. Причем 1 соответствует Воскресенью, а 7 – Субботе
result = NDAYS(*iday, month, iyear*) Функция типа INTEGER(4)	Число дней от 1 января 1900 г. до заданной даты. Результат, меньший нуля, означает число дней до 1 января 1900 г.
CALL NDYIN(*ndays, iday, month, iyear*)	Дату, отстоящую от 1 января 1900 г. на заданное число дней
CALL TDATE(*iday, month, iyear*)	Текущую дату, которую показывают встроенные часы
CALL TIMDY(*ihour, minute, isec*)	Час, минуты и секунды, которые показывают встроенные часы
result = CPSEC( ) Функция типа REAL(4)	Процессорное время, прошедшее с момента первого вызова CPSEC. Первый вызов CPSEC вернет 0.0

Таблица 3.12. Параметры процедур из табл. 3.11

Имена	*Смысл*	*Вид*
iday	Число месяца, например 15	Входной для IDYWK и NDAYS и выходной для NDYIN и TDATE
month	Месяц, например 2	То же
iyear	Год, например 2001	"
ndays	Число дней от 1 января 1900 г.	Выходной
ihour, *minute* и *isec*	Соответственно час, минуты и секунды, которые показывают встроенные часы	Выходные

Замечание. Тип всех параметров – INTEGER(4).

Таблица 3.13. Возможная ошибка подпрограмм из табл. 3.11

Тип	*Код*	*Причина ошибки*	*Где возникает*
1	1	Юлианский календарь вошел в обиход с 45 г. до рождества Христова. Однако поскольку до этого года не было универсально использовавшихся календарей, при расчетах предполагается, что и до 45 г. до рождества Христова также действовал Юлианский календарь	IDYWK, NDAYS

Замечания:

1. Григорианский календарь берет отсчет от 4 октября 1582 г. по юлианскому календарю. Причем этот день юлианского календаря стал 15 октября 1582 г. в григорианском. Поэтому дат между 5 октября 1582 г. и 14 октября 1582 г. включительно не существует и в функциях IDYWK и NDAYS при обращении к ним с датой из этого диапазона возникает завершающая ошибка. Также нельзя задавать даты нулевого года, поскольку такого года не существует.
2. В функциях IDYWK и NDAYS для задания дат до рождества Христова используется отрицательный год.
3. Число дней между двумя датами вычисляется как разность между двумя обращениями к NDAYS с соответствующими датами.
4. Задание отрицательного параметра *ndays* в подпрограмме NDYIN приведет к вычислению даты до рождества Христова.

Пример для IDYWK. Вычисляется день недели 6 февраля 2000 г.

```
program dt1
 use msimsl; use text_transfer              ! Для вывода русского текста
 integer(4) :: iday, iyear, month
 character(10) :: date                      ! Строка, содержащая дату
 iday  = 6; month = 2; iyear = 2000
 write(date, '(2(i2, a), i4)') iday, '.', month, '.', iyear
 print *, trim(ru_doswin('День недели', .false.)), ' ', date, ':', idywk(iday, month, iyear)
end program dt1
```

Результат:

```
День недели  6. 2.2000:       1
```

Пример для NDAYS. Вычисляется число дней от 6 февраля 500 г. до рождества Христова до 25 июля 2000 г.

```
program dt2
 use msimsl; use text_transfer
 integer(4) :: iday, iyear, month, nday0, nday1
 character(10) :: date0, date1              ! Строки, содержащие даты
 iday = 6; month = 2; iyear = -500
```

```
 write(date0, '(2(i2, a), i4)') iday, '.', month, '.', iyear
 nday0 = ndays(iday, month, iyear)
 iday = 25; month = 7; iyear = 2000
 write(date1, '(2(i2, a), i4)') iday, '.', month, '.', iyear
 nday1 = ndays(iday, month, iyear)
 print *, trim(ru_doswin('Число дней между', .false.)), ' ', date0, ' и ', date1, ' =', nday1 - nday0
end program dt2
```

Результат:

Число дней между 6.10.-500 и 25. 7.2000 = 912673

Пример для NDYIN. Вычисляется дата, соответствующая 32-му дню 2000 г.; предварительно вычисляется число дней от 1 января 1900 г. до 31 декабря 1999 г.

```
program dt3
 use msimsl; use text_transfer
 integer(4) :: iday, iyear, month, nday0
 character(10) :: date                    ! Строка, содержащая дату
 nday0 = ndays(31, 12, 1999)
 call ndyin(nday0 + 32, iday, month, iyear)
 write(date, '(2(i2, a), i4)') iday, '.', month, '.', iyear
 print *, trim(ru_doswin('Дата, соответствующая 32-му дню 2000 г.', .false.)), ' - ', date
end program dt3
```

Результат:

Дата, соответствующая 32-му дню 2000 г. – 1. 2.2000

Пример для TDATE. Вычисляется текущая дата.

```
program dt4
 use msimsl; use text_transfer
 integer(4) :: iday, iyear, month
 character(10) :: date                    ! Строка, содержащая дату
 call tdate(iday, month, iyear)
 write(date, '(2(i2, a), i4)') iday, '.', month, '.', iyear
 print *, trim(ru_doswin('Сегодня', .false.)), ' ', date
end program dt4
```

Результат:

Сегодня 6. 2.2000

Замечания:

1. Стандартом 90-го года в Фортран введена встроенная подпрограмма DATE_AND_TIME, возвращающая дату и время, которые показывают встроенные в компьютер часы, а также вычисляющая разницу во времени в минутах по отношению ко времени по Гринвичу. Данные возвращаются как в символьном, так и в цифровом виде. Поэтому вместо TDATE и TIMDY лучше использовать DATE_AND_TIME, например:

```
program dt5
 use text_transfer
 integer(4) :: val(8)
 character(10) :: date                    ! Строка, содержащая дату
 call date_and_time(values = val)         ! Запрашиваем только цифровые данные
 write(date, '(2(i2, a), i4)') val(3), '.', val(2), '.', val(1)
 print *, trim(ru_doswin('Сегодня', .false.)), ' ', date
 print '(1x, a, 4(i3, a))', trim(ru_doswin('Час - минуты - секунды - миллисекунды:', .false.)), &
                    val(5), ' -', val(6), ' -', val(7), ' - ', val(8)
 print *, trim(ru_doswin('Разница по Гринвичу (в минутах)', .false.)), ' - ', val(4)
end program dt5
```

Результат:

Сегодня 6. 2.2000

Час – минуты – секунды – миллисекунды: 12 - 52 - 17 - 0

Разница по Гринвичу (в минутах) – 180

2. Стандартом 95-го года в Фортран введена встроенная подпрограмма CPU_TIME(*time*) возвращающая процессорное время *time*, тип которого – REAL(4). Единицы измерения времени – секунды; после десятичной точки *time* содержит две значащих цифры. Пользователи, имеющие компилятор, поддерживающий Фортран 95, могут использовать CPU_TIME для оценки продолжительности вычислений, например:

```
real(4) :: start_time, finish_time
call cpu_time(start_time)
<Вычисления...>
call cpu_time(finish_time)
print *, 'Calculation time = ', finish_time - start_time
```

Для тех же целей можно использовать функцию CPSEC библиотеки IMSL или стандартную подпрограмму DATE_AND_TIME, например:

```
program find_time
 real(8) :: start_time, finish_time        ! Время начала и завершения вычислений
 ! Функция timer возвращает процессорное время в миллисекундах
 real(8) :: timer
 start_time = timer( )                     ! Время начала вычислений
 <вычисления...>
 finish_time = timer( )                    ! Время конца вычислений
 print *, 'Calculation time = ', finish_time - start_time
end program find_time
real(8) function timer( )                  ! Определяет процессорное время timer
 integer(4) :: ival(8)                     ! в миллисекундах
 call date_and_time(values = ival)
 timer = dble(ival(8))*0.001_8+dble(ival(7))+dble(ival(6))*60.0_8+dble(ival(5))*3600.0_8
end function timer
```

3.5. ЗАДАНИЕ И ЧТЕНИЕ НАСТРОЕК ПРОЦЕДУР IMSL

Целочисленные и вещественные настройки некоторых процедур математической и статистической библиотек IMSL задаются и/или возвращаются подпрограммами IUMAG, SUMAG и DUMAG. Параметры подпрограмм приведены в табл. 3.14. В ее последнем столбце названы подпрограммы, использующие поясняемый параметр.

Таблица 3.14. Параметры подпрограмм IUMAG, SUMAG и DUMAG

Имя	*Смысл*	*Тип*	*Подпрограмма*
prodnm	Имя библиотеки; принимает значения "MATH" или "STAT"	CHARACTER(4)	IUMAG, SUMAG и DUMAG
ichp	Номер главы в фирменной документации, в которой описана процедура, использующая задаваемые IUMAG величины	INTEGER(4)	То же
iact	Равен единице, если необходимо прочитать настройки, и двум, если они задаются	То же	"
numopt	Размер массива *iopts*	"	"
iopts	Массив, содержащий целочисленные имена возвращаемых или задаваемых настроек	"	"
ivals	Массив, содержащий значения возвращаемых или задаваемых настроек. Эти значения являются элементами массивов, соответствующих отдельным именам в *iopts*. Размер *ivals* равен сумме размеров отдельных массивов, соответствующих отдельным именам в *iopts*	"	IUMAG
svals	Имеет то же назначение и аналогично определяемый размер, что и *ivals*	REAL(4)	SUMAG
dvals	То же	REAL(8)	DUMAG

<u>*Замечания*</u>:

1. В математической библиотеке услугами IUMAG, SUMAG и DUMAG пользуются процедуры, применяемые для вычисления корней систем линейных уравнений, определения собственных и сингулярных значений и решения дифференциальных уравнений.
2. На первых этапах применения процедур, использующих подпрограммы IUMAG, SUMAG и DUMAG, пользователь может не знакомиться с этими подпрограммами.
3. Подробное описание подпрограмм можно найти в [24].

4. Распределение материала в руководстве пользователя библиотеки IMSL по главам приведено в разд. 1.1.

Вызов подпрограмм задания и чтения настроек процедур IMSL:

CALL IUMAG(*prodnm, ichp, iact, numopt, iopts, ivals*)

CALL SUMAG(*prodnm, ichp, iact, numopt, iopts, svals*)

CALL DUMAG(*prodnm, ichp, iact, numopt, iopts, dvals*)

Входные данные: *prodnm, ichp, iact, numopt, iopts*.

Входные/выходные данные: *ivals, svals, dvals*.

Пояснения:

1. В *iopts* можно записывать как положительные, так и отрицательные значения.
2. Если *iopts*(i) < 0 ($1 \le i \le numopt$), то IUMAG (SUMAG, DUMAG) возвращает в *ivals* (*svals, dvals*) в режиме чтения принятые по умолчанию значения, а в режиме задания восстанавливает значения, принятые для *iopts*(i) по умолчанию.

Пример 1. Прочитать заданные по умолчанию целочисленные параметры математической библиотеки для процедур, описанных в гл. 1, 2 и 10.

```
program getPar
 use msimsl
 integer(4) :: ichp, i, chapter(4) = (/ 1, 2, 5, 10 /), iact = 1
 integer(4), allocatable :: iopts(:), ivals(:)
 do i = 1, 4
  ichp = chapter(i)
  select case(ichp)
  case(1)                                  ! Глава 1
   numopt = 9                              ! Число регулируемых настроек
   allocate(iopts(numopt), ivals(14))
   iopts = (/ 5, 6, 10, 11, 13, 14, 16, 17, 21 /)
  case(2)                                  ! Глава 2
   numopt = 1
   allocate(iopts(numopt), ivals(14))
   iopts = (/ 1 /)
  case(10)                                 ! Глава 10
   numopt = 3
   allocate(iopts(numopt), ivals(3))
   iopts = (/ 1, 2, 3 /)
  end select
  call iumag('MATH', ichp, iact, numopt, iopts, ivals)
  print *, ivals
  if(allocated(iopts)) deallocate(iopts, ivals)
 end do
end program getPar
```

Результат:

```
Chapter =  1
     1   2147483647   2147483647        1        1        0        0
     1            16            0        1        1        2        1

Chapter =  2
     1       16       0       1       1       16        0
     1       16       0       1       1        2  7867604

Chapter = 10
     0        2       3
```

Пример 2. Демонстрируется техника изменения настроек подпрограммы LCLSQ, вычисляющей, используя метод наименьших квадратов, значения неизвестных x_1, x_2 и x_3 переопределенной системы линейных уравнений

$$\begin{aligned} 3x_1 + 2x_2 + x_3 &= 3.3, \\ 4x_1 + 2x_2 + x_3 &= 2.3, \\ 2x_1 + 2x_2 + x_3 &= 1.3, \\ x_1 + x_2 + x_3 &= 1.0, \end{aligned}$$

связанных ограничениями

$$\begin{aligned} &x_1 + x_2 + x_3 \le 1, \\ &0 \le x_1 \le 0.5, \\ &0 \le x_2 \le 0.5, \\ &0 \le x_3 \le 0.5. \end{aligned}$$

Подпрограмма LCLSQ вызывает процедуру 2-го уровня L2LSQ. До вызова LCLSQ изменяется предусмотренное в ней по умолчанию число итераций с max(*nra*, *nca*) на 6. Второй параметр равен единице, поскольку LCLSQ описана в первой главе документации по процедурам математической библиотеки. Параметр *iact* = 2, что соответствует режиму изменения настроек, используемых LCLSQ. Имя задаваемой целочисленной настройки, хранимое в *iopt*, – 14, а ее новое значение – 6. Регулируется одна настройка, поэтому *numopt* = 1.

Помимо изменения числа итераций SUMAG меняет допуск, используемый при определении ранга. Значение допуска по умолчанию равно SQRT(AMACH(4)), где AMACH(4) – машинная точность $\varepsilon_м$ при вычислениях с параметром разновидности KIND = 4. Устанавливаемое значение – 0.01. Имя вещественной настройки, хранимое в *iopt*, – 2.

Заданные по умолчанию число итераций и допуск после вызова LCLSQ восстанавливаются. Заметим, что изменения не имеют эффекта на решение, а используются для иллюстрации техники обращения с IUMAG и SUMAG.

```
program useIS
 use text_transfer; use msimsl
 integer(4), parameter :: ichp = 1, iact = 2, mcon = 1,                    &
                numopt = 1, nca = 3, nra = 4, Lda = nra, Ldc = mcon
 integer(4) :: iopts(numopt), irtype(mcon), ivals(1)
 real(4) :: a(Lda, nca), b(nra), bc(mcon), c(Ldc, nca)
 real(4) :: res(nra), resnrm, xlb(nca), xsol(nca), xub(nca), svals(1)
 a = reshape((/      3.0,     2.0,     1.0,                  &
                     4.0,     2.0,     1.0,                  &
                     2.0,     2.0,     1.0,                  &
                     1.0,     1.0,      1.0  /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
 b = (/ 3.3, 2.3, 1.3, 1.0 /); c = 1.0; bc = 1.0; irtype = 1; xlb = 0.0; xub = 0.5
 ! Изменим число итераций, используемых в LCLSQ
 iopts(1) = 14; ivals(1) = 6                  ! Новое значение числа итераций
 call iumag('MATH', ichp, iact, numopt, iopts, ivals)
 iopts(1) = 2; svals(1) = 0.01                ! Новое значение допуска
 call sumag('MATH', ichp, iact, numopt, iopts, svals)
 ! Находим оценки неизвестных
 call Lclsq(nra, nca, mcon, a, Lda, b, c, Ldc, bc, bc, irtype, xlb, xub, xsol, res)
 resnrm = snrm2(nra, res, 1)                  ! Вычисляем 2-норму невязки
 print '(1x, a, 3f9.4, /, 1x, a, 4f9.4, /, 1x, a, f9.4)',                 &
       trim(ru_doswin('Решение:', .false.)), xsol,                        &
       trim(ru_doswin('Невязка:', .false.)), res,                         &
       trim(ru_doswin('2-норма:', .false.)), resnrm
 ! Хорошим стилем программирования является восстановление
 !  заданных по умолчанию значений. Заменяем iopt на -iopt
 iopts(1) = -2;     call sumag('MATH', ichp, iact, numopt, iopts, svals)
 iopts(1) = -14;    call iumag('MATH', ichp, iact, numopt, iopts, ivals)
end program useIS
```

Результат:

```
Решение:      0.5000    0.3000    0.2000
Невязка:     -1.0000    0.5000    0.5000    0.0000
2-норма:      1.2247
```

3.6. НЕКОТОРЫЕ СПЕЦИАЛЬНЫЕ ПРОЦЕДУРЫ

3.6.1. Разложение числа на простые множители

Выполняет подпрограмма

CALL PRIME(*n, npf, ipf, iexp, ipw*)

Входные данные: *n*.

Выходные данные: *npf, ipf, iexp, ipw*.

n – целое число, простые множители которого вычисляются.

npf – число неповторяющихся простых множителей числа ABS(*n*). Если *n* равен -1, или 0, или 1, то *npf* получит значение 0. Значение *npf* не может быть больше 13. Этого вполне достаточно, что разложить на простые множители любое представимое в компьютере целое число.

ipf – целый вектор размера 13; *ipf*(*i*) содержит абсолютное значение простого множителя числа *n*, для *i* = 1, ..., *npf*. Оставшиеся 13 – *npf* позиций вектора *ipf* не используются.

iexp – целый вектор размера 13; *iexp*(*i*) содержит число повторений простого множителя *ipf*(*i*), для *i* = 1, ..., *npf*.

ipw – целый вектор размера 13; *ipw*(*i*) = *ipf*(*i*)****iexp*(*i*), для *i* = 1, ..., *npf*.

Возвращенный PRIME результат интерпретируется так:

ABS(*n*) = *ipf*(1)****iexp*(1) * .... * *ipf*(*npf*)****iexp*(*npf*).

Подпрограмма основана на процедуре, опубликованной в [29].

Пример. Определяются простые множители числа 46080 = $2^{10}*3^2*5^1$.

```
program pri
 use msimsl; use text_transfer                ! Для вывода русского текста
 integer(4) :: n = 46080
 integer(4) :: iexp(13), ipf(13), ipw(13), npf
 call prime (n, npf, ipf, iexp, ipw)          ! Простые множители числа 46080
 print '(1x, a, 1x, i6)', trim(ru_doswin('Простые множители числа', .false.)), n
 print '(1x, a,10i5)', 'ipf = ', ipf(1:npf)
 print '(1x, a,10i5)', 'iexp = ', iexp(1:npf)
 print '(1x, a,10i5)', 'ipw = ', ipw(1:npf)
 print '(1x, a, i5)', 'npf = ', npf
end program pri
```

Результат:

```
Простые множители числа  46080
ipf =          2     3     5
iexp =        10     2     1
ipw =       1024     9     5
npf =          3
```

3.6.2. Дополнительные символьные функции

В дополнение к встроенным IMSL содержит 6 символьных функций. Две из них – ACHAR и IACHAR совпадают с одноименными встроенными функциями и поэтому не рассматриваются.

Преобразование строки, состоящей из цифр, в целое число выполняет подпрограмма

CALL CVTSI(*string, number*)

Входные данные: *string*.

Выходные данные: *number*.

string – символьная строка, состоящая из цифр.

number – целое число типа INTEGER(4), содержащееся в строке *string*.

При преобразовании строка – целое число ведущие и завершающие пробелы игнорируются. Если строка содержит символы, отличные от цифр, то возникнет ошибка выполнения. Исключение составляют символы – и +, один из которых может быть размещен перед первой цифрой строки. Также ошибка возникнет, если строка содержит целое число, большее максимально представимой в компьютере целой величины, возвращаемой функциями IMACH и HUGE.

Замечание. Более широкими возможностями обладает оператор READ, в котором имя строки используется в качестве номера устройства ввода.

Пример. Строки '4598' и '-100' преобразовываются в целые числа.

```
use msimsl
integer(4) :: n1, n2, n3
character(10) :: string1 = '4598', string2 = '-100'
call cvtsi(string1, n1)
read(string1, '(i4)') n2
call cvtsi(string2, n3)
print *, n1, n2, n3                    !    4598    4598    -100
```

ASCII-значение символа без учета регистра возвращает функция

result = ICASE(*ch*)

Входные данные: *ch*.

Выходные данные: ICASE.

ch – символ, который преобразовывается функцией ICASE. Тип *ch* – CHARACTER(1).

ICASE – функция типа INTEGER(4), возвращающая ASCII-значение символа *ch*. Причем если *ch* – строчная буква, то ICASE вернет ASCII-значение прописной буквы. Иными словами, ICASE возвращает те же значения, что встроенная функция IACHAR, кроме тех случаев, когда *ch* – строчная буква. На русские буквы действие ICASE не распространяется.

```
use msimsl
print *, icase('g'), icase('G'), iachar('g')    !    71    71    103
```

Сравнение строк без учета регистра выполняет функция

result = IICSR(*str*1, *str*2)

Входные данные: *str*1, *str*2.

Выходные данные: IICSR.

*str*1, *str*2 – сравниваемые строки.

IICSR – функция типа INTEGER(4), возвращающая:

- -1, если *ustr*1 < *ustr*2;

- 0, если *ustr*1 = *ustr*2;
- 1, если *ustr*1 > *ustr*2,

где *ustr*1 и *ustr*2 – переведенные в верхний регистр строки *str*1 и *str*2.

Сравнение строк выполняется после перевода строчных букв в прописные. Фактически сравниваются последовательности ASCII-кодов символов, составляющих строки. Если строки имеют разную длину, то перед сравнением их длина выравнивается за счет прибавления завершающих пробелов к более короткой строке. Функция IICSR не умеет переводить строчные русские буквы в прописные при сравнении строк, содержащих буквы русского алфавита.

```
use msimsl
character(6) :: str1, str2
str1 = 'Abc 2';  str2 = 'abc 1'          ! str1 > str2
print '(i3)', iicsr(str1,str2)           ! 1
str1 = 'abC'; str2 = 'ABc'               ! str1 = str2
print '(i3)', iicsr(str1,str2)           ! 0
str1 = 'aBc'; str2 = 'Abc 1'             ! str1 < str2
print '(i3)', iicsr(str1,str2)           ! -1
```

Позиция подстроки в строке без учета регистра возвращается функцией

result = IIDEX(*chrstr, key*)

Входные данные: *chrstr, key*.

Выходные данные: IIDEX.

chrstr – строка, в которой ищется подстрока *key*.

IIDEX – функция типа INTEGER(4), возвращающая позицию, с которой подстрока *key* начинается в строке *chrstr*. Если *chrstr* не содержит *key*, то IIDEX вернет нуль. При поиске английские строчные буквы строк *chrstr* и *key* переводятся в прописные. Те же функции, но с учетом регистра, выполняет встроенная функция INDEX, которая вдобавок может искать не только с начала, но и с конца строки.

Пример:

```
use msimsl
character  key*5, string*10
string = 'a1b2c3d4e5'
key    = 'C3d4E'
print *, iidex(string, key)              ! 5
print *, iidex(string, 'Ff')             ! 0
```

3.6.3. Вычисление SQRT(*a***2 + *b***2) без исчезновения порядка и переполнения

Выполняет функция HYPOT (DHYPOT)

result = HYPOT(*a, b*)

Входные данные: *a, b*.

Выходные данные: HYPOT= SQRT(*a***2 + *b***2).

Тип входных параметров и результата в случае одинарной точности – REAL(4), в случае двойной – REAL(8).

Функция HYPOT основана на процедуре PYTHAG пакета EISPACK 3 и является модификацией варианта, изложенного в [42].

Пример. Вычисляется без переполнения $\sqrt{a^2+b^2}$, где $a = 10^{20}$, а $b = 20^{20}$.

```
use msimsl
real(4) :: a = 1.0e20, b = 2.0e20
print '(1x, a, 1pe10.4)', 'sqrt(a**2 + b**2) = ', hypot(a, b)
```

Результат:

```
sqrt(a**2 + b**2) = 2.2361E+20
```

3.6.4. Математические и физические константы

Возвращает функция CONST (DCONST)

result = CONST(*name*)

Входные данные: *name*.

Выходные данные: CONST.

Тип результата в зависимости от точности REAL(4) или REAL(8).

name – символьная строка, содержащая имя константы, значение которой нужно узнать.

CONST – значение константы *name*.

Замечания:

1. Буквы строки *name* могут быть как прописными, так и строчными, т. е. имена "PI", "Pi", "pI" и "pi" равнозначны.
2. Список символьных констант, значения которых возвращает CONST, дан в табл. 3.15. Значения констант даны в системе единиц СИ. Константы, после значения которых стоит (E), возвращаются с машинной точностью $\varepsilon_м$.

Таблица 3.15. Константы, значения которых возвращает CONST

Имя	*Описание*	*Значение*	*Ссылка*
AMU	Атомная единица массы	1.6605402E-27 *кг*	[32]
ATM	Стандартное атмосферное давление	1.01325E+5 Н/$м^2$ (E)	[56]
AU	Астрономическая единица	1.496E+11 *м*	[32]
Avogadro	Число Авогадро, N_A	6.0221367E+23 *моль*$^{-1}$	[32]
Boltzman	Постоянная Больцмана, *k*	1.380658E-23 Дж/*град*	[32]
C	Скорость света в вакууме, *c*	2.997924580E+8 *м/с* (E)	[32]

Catalan	Постоянная Каталана	0.915965 ... (E)	-
E	Основание натурального логарифма, *e*	2.718 ... (E)	-
ElectronCharge	Заряд электрона, *e*	1.60217733E-19 К	[32]
ElectronMass	Масса покоя электрона, m_e	9.1093897E-31 *кг*	[32]
ElectronVolt	Электронвольт	1.60217733E-19 Дж	[32]
Euler	Постоянная Эйлера	0.577 ... (E)	-
Faraday	Постоянная Фарадея, *F*	9.6485309E+4 К/*моль*	[32]
FineStructure	Тонкая структура	7.29735308E-3	[32]
Gamma	Постоянная Эйлера	0.577 ... (E)	-
Gas	Универсальная газовая постоянная, *R*	8.314510 Дж/*моль***град*	[32]
Gravity	Постоянная тяготения, γ	6.67259E-11 Н*$м^2$/$кг^2$	[32]
Hbar	Постоянная Планка, $h/2\pi$	1.05457266E-34 Дж**с*	[32]
PerfectGas-Volume	Объем 1 *моля* идеального газа при нормальных условиях, V_0	2.241383E-2 $м^3$	[32]
Pi	Число, π	3.141 ... (E)	-
Planck	Постоянная Планка, *h*	6.6260755E-34 Дж**с*	[32]
ProtonMass	Масса покоя протона, m_p	1.6726231E-27 *кг*	[32]
Rydberg	Постоянная Ридберга, R_∞	1.0973731534E+7 $м^{-1}$	[32]
SpeedLight	Скорость света в вакууме, *c*	2.997924580E+8 *м/с* (E)	[32]
StandardGravity	Ускорение свободного падения, *g*	9.80665 $м/с^2$ (E)	[56]
StandardPressure	Стандартное атмосферное давление	1.01325E + 5 Н/$м^2$ (E)	"
StefanBoltzmann	Постоянная Стефана – Больцмана, σ	5.67051E-8 Вт/ $м^2$**град*^{4}	[32]
WaterTriple	Термодинамическая температура тройной точки воды	2.7316E+2 *град* (E)	[56]

Перевод из одной системы единиц в другую выполняет CUNIT.

Пример. Функцией CONST возвращается постоянная Эйлера, вычисляемая по формуле $\gamma = \lim_{n\to\infty}\left(\sum_{k=1}^{n-1}\frac{1}{k} - \ln n\right)$.

```
use msimsl
print *, 'gamma = ', const('gamma')
```

Результат:
gamma = 5.772157E-01

3.6.5. Перевод величин из одной системы единиц в другую

Выполняется подпрограммой

CALL CUNIT(*x, xunits, y, yunits*)

Входные данные: *x, xunits, yunits*.

Выходные данные: *y*.

x – значение, подлежащее пересчету в другие единицы.

xunits – символьная строка, содержащая имя единиц измерения *x*.

y – значение в *yunits*, соответствующее величине *x* в *xunits*.

yunits – символьная строка, содержащая имя единиц измерения *y*.

Замечания:

1. Символьные строки *xunits* и *yunits* имеют вид $u_1*u_2*...*u_m/v_1*v_2*...v_n$, где u_i и v_i – имена базовых единиц (возможно, возведенных в степень). Например, "m*kg/s", "meter*kilogram/second", "meter" или "m/kg2".
2. Буквы символьных строк *xunits* и *yunits* могут быть как прописными, так и строчными, т. е. строки "METER", "Meter" и "meter" равнозначны.
3. Если *xunits* = "SI", то *x* рассматривается как величина в системе СИ. И наоборот, если *yunits* = "SI", то уже *y* рассматривается как величина в системе СИ.
4. Входные и выходные единицы проверяются на совместимость за исключением ситуации, когда *xunits* = "SI".
5. Базовые имена единиц измерения приведены в табл. 3.16; префиксы, которые можно использовать с базовыми имена единиц измерения, – в табл. 3.17.

Таблица 3.16. Базовые имена единиц измерения

Величина	*Базовые имена единиц измерения*
Время	day, hour = hr, min = minute, s = sec = second, year
Частота	Hertz = Hz
Масса	AMU, g = gram, lb = pound, ounce = oz, slug
Расстояние	Angstrom, AU, feet = foot = ft, in = inch, m = meter = metre, micron, mile, mill, parsec, yard
Площадь	acre
Объем	l = liter = litre
Сила	dyne, N = Newton, poundal
Энергия	BTU(thermochemical), Erg, J = Joule
Работа	W = watt

Давление	ATM = atomosphere, bar, Pascal
Температура	degC = Celsius, degF = Fahrenheit, degK = Kelvin
Вязкость	poise, stoke
Заряд	Abcoulomb, C = Coulomb, statcoulomb
Ток	A = ampere, abampere, statampere
Напряжение	Abvolt, V = volt
Магнитная индукция	T = Tesla, Wb = Weber
Другие	1, farad, mole, Gauss, Henry, Maxwell, Ohm

Таблица 3.17. Префиксы, используемые с имена из табл. 3.16

Префикс	*Наименование*	*Множитель*
A	атто	10^{-18}
F	фемпто	10^{-15}
P	пико	10^{-12}
N	нано	10^{-9}
U	микро	10^{-6}
M	милли	10^{-3}
C	санти	10^{-2}
D	деци	10^{-1}
DK	дека	10^{+2}
K	кило	10^{+3}
Myria	мириа	10^{+4}
Mega	мега	10^{+6}
G	гига	10^{+9}
T	тера	10^{+12}

При работе с CUNIT может возникнуть *информационная ошибка* типа 3 с кодом 8: "Для совместимости величин необходимо преобразовать единицы массы в единицы силы".

Пример. Проверяется равенство $3{,}6 \cdot 10^3$ Дж = 1 ватт-час.

```
use msimsl
real(4) :: en1 = 3.6e3, en2 = 3.6, wh1_1, wh1_2, wh2
call cunit(en1, 'Joule', wh1_1, 'watt*hour')  ! Вызов без префикса
call cunit(en1, 'Si', wh1_2, 'watt*hour')     ! То же, но вместо 'Joule' употребим 'Si'
call cunit(en2, 'kJ', wh2, 'w*hr')            ! Используем префикс k
print *, wh1_1, wh1_2, wh2                    !     1.000000      1.000000      1.000000
```

3.7. ВЕРСИЯ БИБЛИОТЕКИ IMSL

Версию библиотеки IMSL, названия операционной системы и компилятора Фортрана, к которым приспособлена библиотека, и ее серийный номер возвращает функция

result = VERML(*iselct*)

Функция вернет:

- версию библиотеки IMSL, если *iselct* = 1;
- название операционной системы, если *iselct* = 2;
- название компилятора Фортрана, если *iselct* = 3;
- серийный номер библиотеки IMSL, если *iselct* = 4.

```
program ver
 use msimsl
 integer(4) :: iselct
 character(50), dimension(4) :: string
 string(1) = 'IMSL math/library version number: '
 string(2) = 'Operating system ID number: '
 string(3) = 'Fortran compiler version number: '
 string(4) = 'IMSL MATH/LIBRARY serial number: '
 do iselct=1, 4
   print '(1x, 2a)', string(iselct), verml(iselct)
 end do
end program ver
```

4. БАЗОВЫЕ ПРОЦЕДУРЫ ЛИНЕЙНОЙ АЛГЕБРЫ

4.1. ВВЕДЕНИЕ

Базовые процедуры линейной алгебры (БПЛА) библиотеки IMSL выполняют простые операции над векторами и матрицами, такие, как вычисление скалярного произведения векторов, умножение матриц и др. БПЛА содержат 3 уровня. Процедуры 1-го уровня оперируют преимущественно векторами. Процедуры, оперирующие векторами и матрицами, образуют 2-й и 3-й уровни.

БПЛА IMSL написаны на Фортране. После появления Фортрана 90 некоторые из них могут быть заменены встроенными процедурами для массивов или присваиваниями массивов. Поэтому в главе наряду с описанием БПЛА приводятся, по возможности, и заменяющие их встроенные процедуры. Не всегда, однако, возможные замены целесообразны, поскольку могут привести к росту времени вычислений.

В то же время большинство БПЛА сохраняют актуальность, поскольку (это прежде всего относится к БПЛА уровней 2 и 3) они с некоторыми видами матриц работают быстрее, чем заменяющая их встроенная функция MATMUL, используют в некоторых случаях при вычислениях аккумуляторы повышенной точности, учитывают особенности машинной арифметики и оперируют, в отличие от встроенных функций для массивов, не только матрицами общего вида, но и ленточными, треугольными, симметрическими и другими матрицами.

Помимо базовых IMSL содержит иные, не входящие в состав БПЛА процедуры, выполняющие операции над векторами и матрицами. Эти процедуры рассмотрены в последнем разделе этой главы.

Изложению процедур БПЛА предшествуют необходимые для восприятия этой и последующих глав сведения о векторах и матрицах и о способах их представления в процедурах IMSL.

4.2. НЕКОТОРЫЕ СВЕДЕНИЯ О ВЕКТОРАХ И МАТРИЦАХ

4.2.1. Обозначение векторов и матриц

Вектор $x \in \mathbf{R}^n$ (или $\mathbf{R}^{n\times 1}$), где $\mathbf{R}$ – множество всех вещественных чисел, а $\mathbf{R}^n$ – линейное векторное пространство всех вещественных векторов, является *вектор-столбцом*

$$x \in \mathbf{R}^n \Leftrightarrow x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \; x_i \in \mathbf{R}, \; i = 1, \ldots, n .$$

Вектор $y = x^T$ является вектор-строкой.

Пространство $\mathbf{R}^{1\times n}$ состоит из *вектор-строк*:

$$x \in \mathbf{R}^{1\times n} \Leftrightarrow x = (x_1, \ldots, x_n).$$

Прямоугольная $m\times n$-матрица $A \in \mathbf{R}^{m\times n}$, где $\mathbf{R}^{m\times n}$ – множество всех вещественных $m\times n$-матриц, представляется в виде прямоугольной таблицы из $m\times n$ элементов:

$$A \in R^{m\times n} \Leftrightarrow A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}, \; a_{ij} \in R ,$$

здесь a_{ij} – элемент матрицы, расположенный на пересечении ее i-й строки и j-го столбца.

Матрицу можно представить как совокупность вектор-строк

$$A = \begin{pmatrix} a_{1:}^T \\ a_{2:}^T \\ \vdots \\ a_{m:}^T \end{pmatrix} \in R^{m\times n}, \; a_{i:} = (a_{i1}, a_{i2}, \ldots, a_{in})^T \in R^n$$

или вектор-столбцов

$$A = (a_{:1}, a_{:2}, \ldots, a_{:n}) \in R^{m\times n}, \; a_{:j} = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix} \in R^m .$$

Первое представление называется *блочно-строковым*, второе – *блочно-столбцовым*.

Смысл двоеточия в индексе вектор-строки (столбца) такой же, как и у соответствующего сечения двумерного массива, например:

```
real(4) :: a(4, 5)
read *, a(2, :)          ! Ввод элементов второй строки массива a
read *, a(:, 4)          ! Ввод элементов четвертого стролбца массива a
```

В программах, выполняющих матричные вычисления, имена массивов, содержащих элементы матриц, будем задавать строчными буквами, совпадающими с именами матриц, например:

```
integer(4), parameter :: Lda = 5, n = 4
real(4) :: a(Lda, n)                    ! Массив, представляющий матрицу A
```

4.2.2. Операции над векторами

Над векторами выполняются операции (x, y и $z \in \mathbf{R}^n$; α и $\omega \in \mathbf{R}$):

- умножение на число α: $z = \alpha x \Rightarrow z_i = \alpha x_i$;
- сложение векторов: $z = x + y \Rightarrow z_i = x_i + y_i$;
- скалярное произведение векторов – это число $\omega = x^T y = (x, y) = \sum_{i=1}^{n} x_i y_i$;
- покомпонентное произведение векторов: $z = x*y \Rightarrow z_i = x_i * y_i$.

Скалярное произведение $(x, x) = \sum_{i=1}^{n} x_i^2$ неотрицательно и равно нулю, когда $x = 0$.

В случае комплексных векторов (x и $y \in \mathbf{C}^n$; $\omega \in \mathbf{C}$, где $\mathbf{C}$ – множество всех комплексных чисел, а $\mathbf{C}^n$ – линейное векторное пространство всех комплексных векторов) скалярное произведение

$$\omega = x^T y = (x, y) = \sum_{i=1}^{n} x_i \bar{y}_i ,$$

где $\bar{y}_i$ – величина, сопряженная y_i, обладает эрмитовой симметрией:

$(y, x) = \overline{(x, y)}$.

В Фортране все перечисленные операции можно выполнить, не прибегая к циклам, а используя выражения с массивами. Векторы – операнды выражений и вектор, в который записывается результат, должны быть одного размера. Например:

```
integer(4) :: i
real(4) :: x(10) = (/ 1.0, 2.0, 3.0, 4.0, 5.0, -1.0, -2.0, -3.0, -4.0, -5.0 /)
real(4) :: y(10) = (/ 1.1, 2.1, 3.1, 4.1, 5.1, -1.1, -2.1, -3.1, -4.1, -5.1 /)
real(4) :: z(10), alpha = 4.0, c
z = alpha * x                           ! Умножение на скаляр
z = x + y                               ! Сложение векторов
c = dot_product(x, y)                   ! Скалярное произведение векторов
z = x * y                               ! Покомпонентное произведение векторов
! Скалярное произведение c векторов x и y является суммой
c = 0.0
do i = 1, size(x)                       ! SIZE(x) – возвращает размер вектора x
  c = c + x(i) * y(i)
end do                                  ! Ту же сумму вернет встроенная функция SUM:
c = sum(x * y)
```

Помимо приведенных операций полезно помнить, что произведением вектор-столбца x размера m и вектор-строки y^T размера n (x и $y \in \mathbf{R}^n$) является $m \times n$-матрица A с элементами $a_{ij} = x_i * y_j$:

$$A=\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_m \end{pmatrix}(y_1 \;\; y_2 \dots y_m) = xy^T \in \mathbf{R}^{m\times n}.$$

В Фортране ее вернет встроенная функция MATMUL. Правда, для обеспечения ее работоспособности потребуется преобразовать вектор x в массив формы (/ m, 1 /), а вектор y^T – в массив формы (/ 1, n /):

```
integer(4), parameter :: m = 3, n = 5
integer(4) :: i, j
real(4) :: x(m) = (/ 1.0, 2.0, 3.0 /)
real(4) :: y(n) = (/ 5.0, 6.0, 7.0, 8.0, 9.0 /)
real(4) :: a(m, n) = 0.0
a = matmul(reshape(x, shape = (/ m, 1 /)), reshape(y, shape = (/ 1, n /)))
do i = 1, m                                  ! Вывод результата
  print '(1x, 10f6.2)', a(i, :)
end do
! Тот же результат, но гораздо быстрее, даст вложенный цикл
do j = 1, n
 do i = 1, m
  a(i, j) = x(i) * y(j)
 end do
end do
```

Результат:

5.00	6.00	7.00	8.00	9.00
10.00	12.00	14.00	16.00	18.00
15.00	18.00	21.00	24.00	27.00

Замечание. Рассмотренное умножение вектор-столбца x на вектор-строку y^T называют *внешним произведением*. Ранг внешнего произведения, когда x и y – действительные векторы, не выше единицы.

4.2.3. Операции над матрицами

Основные операции над матрицами:

- транспонирование ($\mathbf{R}^{m\times n} \to \mathbf{R}^{n\times m}$):

  $C = A^T \Rightarrow c_{ij} = a_{ji}$ (строка с номером i в A^T является столбцом с тем же номером в A);

- сложение ($\mathbf{R}^{m\times n} + \mathbf{R}^{m\times n} \to \mathbf{R}^{m\times n}$):

  $C = A + B \Rightarrow c_{ij} = a_{ij} + b_{ij}$;

- умножение матрицы на число ($\mathbf{R}\times\mathbf{R}^{m\times n} \to \mathbf{R}^{m\times n}$):

  $C = \alpha A \Rightarrow c_{ij} = \alpha a_{ij}$;

- умножение матрицы на матрицу ($\mathbf{R}^{m\times r}\times\mathbf{R}^{r\times n}\to\mathbf{R}^{m\times n}$):

$$C = AB \Rightarrow c_{ij} = \sum_{k=1}^{r} a_{ik} b_{ki}\,.$$

Все приведенные операции реализуются в Фортране без применения циклов:

```
integer(4), parameter :: m = 3, r = 4, n = 5
real(4) :: a(m, r), b(r, n), c1(r, m), c2(m, r), c3(m, n), alpha = 0.5
a = reshape((/      5.0,    6.0,    7.0,    8.0,                    &
                   10.0,   12.0,   14.0,   16.0,                    &
                   15.0,   18.0,   21.0,   24.0 /), shape = (/ m, r /), order = (/ 2, 1 /))
b = reshape((/      5.0,    6.0,    7.0,    8.0,    9.0,            &
                   10.0,   12.0,   14.0,   16.0,   10.0,            &
                  -11.0,  -10.0,  -15.0,  -17.0,   -5.0,            &
                   15.0,   18.0,   21.0,   24.0,   -8.0 /),         &
                  shape = (/ r, n /), order = (/ 2, 1 /))
c1 = transpose(a)                    ! C1 = A^T
c2 = alpha * a                       ! C2 = αA
c2 = a + c2                          ! C2 = A + C2
c3 = matmul(a, b)                    ! C = AB
```

4.2.4. Элементарные преобразования

Элементарными преобразованиями матрицы A называются:

- перестановка двух строк или двух столбцов матрицы;
- умножение строки (или столбца) на произвольное отличное от нуля число;
- прибавление к одной строке (или столбцу) другой строки (столбца), умноженной на некоторое число.

Элементарные преобразования не меняют ранга матрицы.

4.2.5. Определитель и миноры матрицы

С квадратной $n\times n$-матрицей A связан определитель

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}, \det A \in \mathbf{R}\,.$$

Пример матрицы и ее определителя:

$$A = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 4 \\ 2 & 4 & 14 \end{pmatrix}, \qquad \det A = \begin{vmatrix} 1 & -1 & 2 \\ -1 & 5 & 4 \\ 2 & 4 & 14 \end{vmatrix} = 4\,.$$

Определитель $n\times n$-матрицы A вычисляется через определители n-1×n-1-матриц, например:

$$\det A=\sum_{j=1}^{n}(-1)^{j+1}a_{1j}\det A_{1j}\ ,$$

где A_{1j} – это n-1×n-1-матрица, получаемая из A вычеркиванием первой строки и j-го столбца.

Пример:

$$\det A=\begin{vmatrix}1 & -1 & 2\\ -1 & 5 & 4\\ 2 & 4 & 14\end{vmatrix}=1\begin{vmatrix}5 & 4\\ 4 & 14\end{vmatrix}+1\begin{vmatrix}-1 & 4\\ 2 & 14\end{vmatrix}+2\begin{vmatrix}-1 & 5\\ 2 & 4\end{vmatrix}=$$

$$=(70-16)+(-14-8)+2(-4-10)=54-22-28=4.$$

Рассмотрим, например, 3×4-матрицу

$$B=\begin{pmatrix}1 & -1 & 2 & 3\\ -1 & 5 & 4 & 2\\ 2 & 4 & 14 & 1\end{pmatrix}.$$

Выделим в ней любые k строк и столбцов ($k\le 3$). Определитель k-го порядка, составленный из элементов матрицы, стоящих на пересечении выделенных строк и столбцов, называется *минором* этой матрицы. Всего для 3×4-матрицы можно составить $C_4^1\cdot C_3^1=12$ миноров 1-го порядка – это сами элементы матрицы B; $C_4^2\cdot C_3^2=18$ миноров 2-го порядка и $C_4^3\cdot C_3^3=4$ миноров 3-го порядка.

Пример миноров 2-го и 3-го порядка матрицы B:

$$\begin{vmatrix}2 & 3\\ 2 & 4\end{vmatrix},\quad \begin{vmatrix}5 & 4\\ 4 & 14\end{vmatrix},\quad \begin{vmatrix}-1 & 2 & 3\\ 5 & 4 & 2\\ 4 & 14 & 1\end{vmatrix}.$$

Минор квадратной матрицы называется *главным диагональным*, если он образован путем выделения k подряд расположенных строк и столбцов матрицы, причем выделение начинается с первой строки и первого столбца матрицы. Получаемые в результате такого выделения подматрицы называются *главными подматрицами*.

Пример. Главными диагональными минорами матрицы

$$A=\begin{pmatrix}1 & -1 & 2 & 3\\ -1 & 5 & 4 & 2\\ 2 & 4 & 14 & 1\\ 3 & 5 & 7 & 4\end{pmatrix}\ \text{являются}\ 1,\begin{vmatrix}1 & -1\\ -1 & 5\end{vmatrix},\begin{vmatrix}1 & -1 & 2\\ -1 & 5 & 4\\ 2 & 4 & 14\end{vmatrix},\begin{vmatrix}1 & -1 & 2 & 3\\ -1 & 5 & 4 & 2\\ 2 & 4 & 14 & 1\\ 3 & 5 & 7 & 4\end{vmatrix}.$$

Определитель произведения нескольких матриц равен произведению определителей этих матриц.

4.2.6. Виды матриц

Матрица, в которой число строк равно числу столбцов, называется *квадратной*. В противном случае матрица называется *прямоугольной*.

Матрица $A \in \mathbf{R}^{m\times n}$, не имеющая каких-либо специфических черт, называется матрицей *общего вида*.

Матрица $D \in \mathbf{R}^{m\times n}$ называется *диагональной*, если все ее элементы, не лежащие на главной диагонали, равны нулю. Заметим, что всякую матрицу можно элементарными преобразованиями привести к диагональной форме.

Если диагональные элементы диагональной квадратной $n\times n$-матрицы равны единице, то она называется *единичной* и обозначается I_n.

Пример диагональной и единичной матриц:

$$D=\begin{pmatrix}3&0&0&0\\0&4&0&0\\0&0&5&0\end{pmatrix},\qquad I_3=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}.$$

Компактная форма записи диагональной матрицы:

$$D = \mathrm{diag}(d_1, d_2, ..., d_q),\ q = \min(m, n).$$

Единичную матрицу также называют *тождественной*, поскольку

$$AI_n = A,\ A \in \mathbf{R}^{n\times n}.$$

Единичную $n\times n$-матрицу I_n можно задать блочно-столбцовым представлением

$$I_n = (e^{(1)}, ..., e^{(n)}),$$

где $e^{(k)}$ – k-й канонический вектор, $e^{(k)} = (\underbrace{0, ..., 0}_{k-1}, 1, \underbrace{0, ..., 0}_{n-k})^T$.

Матрица X является *обратной* по отношению к матрице A, если

$$AX = I.$$

Для обратной матрицы X используется обозначение A^{-1}. Легко показать, что для данной невырожденной матрицы A справедливо

$$AA^{-1} = A^{-1}A = I.$$

Определитель матрицы A^{-1} равен $1/\det(A)$; обратной для нее служит матрица A. Поскольку

$$(A^T)^{-1} = (A^{-1})^T,$$

обратная транспонированная матрица обозначается как A^{-T}.

Легко вычисляется обратная матрица D^{-1} диагональной матрицы D, диагональные элементы которой отличны от нуля, например:

$$D=\begin{pmatrix} d_{11} & & \\ & d_{22} & \\ & & d_{33} \end{pmatrix}, \qquad D^{-1}=\begin{pmatrix} 1/d_{11} & & \\ & 1/d_{22} & \\ & & 1/d_{33} \end{pmatrix}.$$

Нахождение A^{-1} называется *обращением* A. Если обращение A возможно, т. е. если существует A^{-1}, матрица A называется *невырожденной*, в противном случае A называется *вырожденной*. Определитель вырожденной матрицы равен нулю.

Квадратная матрица A называется *симметрической*, если $A^T = A$, например:

$$A=\begin{pmatrix} 3 & 2 & 7 \\ 2 & 5 & 9 \\ 7 & 9 & 8 \end{pmatrix}.$$

Легко показать: если A – вещественная общего вида квадратная матрица, то матрицы AA^T и $A + A^T$ будут симметрическими. Вещественная симметрическая матрица преобразованиями подобия приводится к диагональной матрице.

Для любой симметрической $n\times n$-матрицы A можно указать *квадратичную форму*

$$f=\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j\,, \tag{4.1}$$

имеющую элементы матрицы A своими коэффициентами. Причем квадратичная форма (4.1) может быть записана в виде произведения

$$f = x^T A x,$$

где x – вектор-столбец, составленный их неизвестных x_i: $x = (x_1, x_2, ..., x_n)^T$. Справедливо и обратное: из коэффициентов a_{ij} квадратичной формы (4.1) можно составить симметрическую $n\times n$-матрицу $A = (a_{ij})$. Она называется *матрицей квадратичной формы* f.

Квадратная матрица Q называется *ортогональной*, если она удовлетворяет любому из следующих условий:

1) строки матрицы Q образуют ортонормированную систему векторов;
2) $QQ^T = I$;
3) $Q^TQ = I$;
4) $Q^T = Q^{-1}$.

Произведение двух ортогональных матриц также является ортогональный матрицей.

Квадратная матрица H с комплексными элементами называется *эрмитовой*, если $\overline{H}^T = H$, где $\overline{H}$ - комплексно-сопряженная с H матрица. Например:

$$H=\begin{pmatrix} 3.0+0.0i & 1.0-1.0i & 4.0-0.0i \\ 1.0+1.0i & 2.0+0.0i & -5.0+1.0i \\ 4.0+0.0i & -5.0-1.0i & -2.0+0.0i \end{pmatrix}.$$

Квадратная комплексная матрица A называется *унитарной*, если она удовлетворяет любому из следующих условий:

$$\bar{A}A=I,\ A\bar{A}=I,\ \bar{A}=A^{-1}.$$

Определитель унитарной матрицы есть комплексное число, модуль которого равен единице.

Прямоугольная или квадратная матрица называется *ленточной*, если все ее элементы a_{ij}, для которых $i - j > nlca$ или $j - i > nuca$, равны нулю, где $nlca$ и $nuca$ – соответственно *нижняя* и *верхняя ширина ленты*. Например в матрице

$$A=\begin{pmatrix} 3 & 2 & 7 & 0 & 0 & 0 \\ 2 & 5 & 9 & 4 & 0 & 0 \\ 0 & 9 & 8 & 3 & 5 & 0 \\ 0 & 0 & 4 & 2 & 6 & 8 \\ 0 & 0 & 0 & 2 & 9 & 1 \end{pmatrix}$$

$nlca = 1$ и $nuca = 2$. Число $m = nlca + nuca + 1$ называется *шириной ленты*. Диагонали, отличные от главной, называются *кодиагоналями*. Используя это понятие, ленточную матрицу можно охарактеризовать по-другому – это матрица, все ненулевые элементы которой расположены на $nlca$ нижних и $nuca$ верхних кодиагоналях, примыкающих к главной диагонали. Частным случаем ленточной является трехдиагональная матрица, в которой $nlca = nuca = 1$.

Матрица называется *разреженной*, если число ее ненулевых элементов существенно больше общего числа элементов матрицы.

Квадратная матрица является *треугольной*, если ее элементы, стоящие выше (ниже) главной диагонали, равны нулю.

Квадратная матрица L называется *нижней треугольной*, если нулю равны элементы, расположенные выше главной диагонали; квадратная матрица U называется *верхней треугольной*, если нулю равны элементы, расположенные ниже главной диагонали.

Пример нижней и верхней треугольных матриц:

$$L=\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1.5 & 1 \end{pmatrix},\qquad U=\begin{pmatrix} 1 & -1 & 2 \\ 0 & 4 & 6 \\ 0 & 0 & 1 \end{pmatrix}.$$

Треугольная матрица, диагональные элементы которой равны единице, называется *унитреугольной*. Так, нижней унитреугольной является приведенная выше матрица L.

Аналогично определяются *нижняя* (L) *и верхняя* (U) *трапециевидные* прямоугольные матрицы.

Пример нижней и верхней трапециевидных матриц:

$$L=\begin{pmatrix}1 & 2 & 6 & 5\\ 0 & 3 & 7 & 4\\ 0 & 0 & 4 & 9\end{pmatrix},\qquad U=\begin{pmatrix}1 & 3 & 6\\ 0 & 2 & 5\\ 0 & 0 & 4\\ 0 & 0 & 0\end{pmatrix}.$$

4.2.7. Умножение матриц

Произведением $m\times r$-матрицы A и $r\times n$-матрицы B является $m\times n$-матрица C ($C=AB$, $A\in\mathbf{R}^{m\times r}$, $B\in\mathbf{R}^{r\times n}$ и $C\in\mathbf{R}^{m\times n}$).

Рассмотрим прежде, как вычисляется произведение общего вида матриц A и B. Если представить матрицы A и B соответственно в виде наборов вектор-строк и вектор-столбцов (каждый вектор из этих наборов имеет размер r), т. е. в блочно-строчном или блочно-столбцовом виде

$$A=\begin{pmatrix}a_{1:}^T\\ \vdots\\ a_{m:}^T\end{pmatrix}\in\mathbf{R}^{m\times r}\ ,\ a_{i:}\in R^r;\ B=\left(b_{:1},\ \ldots,\ b_{:n}\right)\in\mathbf{R}^{r\times n}\ ,\ b_{:j}\in R^r\ ,$$

то значение каждого элемента c_{ij} матрицы $C=AB$ можно записать в виде скалярного произведения соответствующих векторов матриц A и B:

```
c = 0.0
do j = 1, n                          ! Индекс столбца меняем во внешнем цикле,
 do i = 1, m                         ! а индекс строки – во внутреннем
  c(i, j) = dot_product(a(i, :), b(:, j))
 end do
end do
```

<u>*Замечание*</u>. Поскольку в Фортране элементы матрицы расположены в памяти по столбцам, то индекс столбца следует изменять во внешнем цикле, а индекс строки – во внутреннем. Это обеспечит естественный и, следовательно, более быстрый доступ к элементам массива.

Произведение матриц A и B имеет смысл, когда число столбцов матрицы A равно числу строк матрицы B. При этом число строк матрицы AB равно числу строк матрицы A, а число столбцов AB равно числу столбцов B.

Произведением A^TA ($A\in\mathbf{R}^{m\times n}$) является симметрическая матрица $C\in\mathbf{R}^{n\times n}$.

В некоторых случаях, когда матрицы имеют специальный вид, например являются треугольными, вместо встроенной функции MATMUL целесообразнее с позиции снижения временных затрат вычислять произведение $C = AB$ с использованием циклов.

При умножении верхних треугольных $n \times n$-матриц A и B результатом является также верхняя треугольная $n \times n$-матрица, поэтому число выполняемых операций можно снизить в 6 раз по сравнению с умножением $n \times n$-матриц A и B общего вида:

```
c = 0.0
do j = 1, n                          ! Индекс столбца меняем во внешнем цикле
 do i = 1, j                         ! Индекс строки меняем от единицы до j
  c(i, j) = dot_product(a(i, i:j), b(i:j, j))
 end do
end do
```

Умножение треугольных матриц требует примерно $n^3/3$ флопов.

Еще меньше операций требуется для вычисления произведения диагональных $n \times n$-матриц:

```
c = 0.0
do i = 1, n
 c(i, i) = a(i, i) * b(i, i)         ! C – также является диагональной матрицей
end do
```

Очевидно, что $AI_n = I_m A = A \in \mathbf{R}^{m \times n}$.

Умножение матриц некоммутативно, т. е. в общем случае $AB \neq BA$. Однако оно ассоциативно, т. е. $(AB)C = A(BC)$.

4.2.8. Умножение матрицы на вектор

Результатом произведения $m \times n$-матрицы A ($A \in \mathbf{R}^{m \times n}$) на вектор x размера n ($x \in \mathbf{R}^n$) является вектор $z = Ax$ размера m ($z \in \mathbf{R}^m$). Каждый элемент вектора

$$z_i = \sum_{j=1}^{n} a_{ij} x_i^T = x_i^T \sum_{j=1}^{n} a_{ij} .$$

```
integer(4), parameter :: m = 3, n = 5
real(4) :: x(n) = (/ 5.0, 6.0, 7.0, 8.0, 9.0 /), z(m) = 0.0
real(4) :: a(m, n)
a = reshape((/      5.0,    6.0,    7.0,    8.0,    9.0,                &
                   10.0,   12.0,   14.0,   16.0,   18.0,                &
                   15.0,   18.0,   21.0,   24.0,   27.0 /), shape = (/ m, n /), order = (/ 2, 1 /))
z = matmul(a, x)
print '(1x, 10f9.2)', z                    ! 255.00  510.00  765.00
```

Если же матрицу A представить в виде набора вектор-строк

$$A = \begin{pmatrix} a_{1:}^T \\ \vdots \\ a_{m:}^T \end{pmatrix} \in \mathbf{R}^{m \times n},$$

где $a_{i:}^T = (a_{i1}, \ldots, a_{in})$ – i-я вектор-строка матрицы A, то $z = Ax$ можно записать с использованием в цикле скалярного произведения:

```
z = 0.0                                  ! Цикл вернет то же, что и MATMUL(a, x)
do i = 1, m
  z(i) = z(i) + dot_product(a(i, :), x)
end do
```

Можно также умножить $m \times n$-матрицу A на вектор x размера m слева. Результатом такого произведения будет являться вектор $z = xA$ размера n. Каждый элемент вектора

$$z_j = x_j \sum_{i=1}^{m} a_{ij} .$$

```
integer(4), parameter :: m = 3, n = 5
real(4) :: x(m) = (/ 1.0, 2.0, 3.0 /), z(n) = 0.0
real(4) :: a(m, n)
a = reshape((/      5.0,    6.0,    7.0,    8.0,    9.0,                &
                   10.0,   12.0,   14.0,   16.0,   18.0,                &
                   15.0,   18.0,   21.0,   24.0,   27.0 /), shape = (/ m, n /), order = (/ 2, 1 /))
z = matmul(x, a)
print '(1x, 10f9.2)', z                  ! 70.00   84.00   98.00  112.00  126.00
```

Произведение xA, так же как и Ax, можно найти, применив в цикле скалярное произведение $(x, a_{:j})$, где $a_{:j}$ – вектор-столбец $m \times n$-матрицы $A = (a_{:1}, \ldots, a_{:n})$:

```
z = 0.0                                  ! Цикл вернет то же, что и MATMUL(x, a)
do j = 1, n
  z(j) = z(j) + dot_product(x, a(:, j))
end do
```

Если $A = I_n$, то очевидно, что $xI_n = I_n x = x$, где $x \in \mathbf{R}^n$.

4.2.9. Блочные матрицы

Частным случаем блочных матриц являются приведенные выше блочно-столбцовые и блочно-строчные матрицы. Общее представление $m \times n$-матрицы A в виде блоков:

$$A=\begin{array}{cc}\begin{pmatrix} A_{11} & \cdots & A_{1q} \\ \vdots & & \vdots \\ A_{p1} & \cdots & A_{pq} \end{pmatrix} & \begin{matrix} m_1 \\ \\ m_p \end{matrix} \\ \begin{matrix} n_1 & & n_q \end{matrix} & \end{array}$$

где блок A_{ij} – это подматрица размерности $m_i \times n_j$, $m_1 + ... + m_p = m$, а $n_1 + ... + n_q = n$. Представленная в таком виде матрица A называется *$p\times q$-блочной матрицей*.

Говорят, что $m\times n$-матрица B разбита на блоки *согласованно* с A, если B также есть $p\times q$-блочная матрица и размерности соответствующих блоков A и B одинаковы.

Так же как и обычные, согласованные блочные матрицы можно умножать на число и складывать. Суммой согласованных $p\times q$-блочных матриц A и B также является $p\times q$-блочная матрица $C = A + B$, где подматрица $C_{ij} = A_{ij} + B_{ij}$.

Важными частными случаями являются произведение 2×2-блочных матриц $A \in \mathbf{R}^{m\times r}$ и $B \in \mathbf{R}^{r\times n}$

$$A=\begin{array}{cc}\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} & \begin{matrix} m_1 \\ m_2 \end{matrix} \\ \begin{matrix} r_1 & r_2 \end{matrix} & \end{array}, \quad B=\begin{array}{cc}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} & \begin{matrix} r_1 \\ r_2 \end{matrix} \\ \begin{matrix} n_1 & n_2 \end{matrix} & \end{array}$$

и умножение 2×2-блочной матрицы A на двумерный блочный вектор x.

Произведение 2×2-блочных матриц

$$AB = C=\begin{array}{cc}\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} & \begin{matrix} m_1 \\ m_2 \end{matrix} \\ \begin{matrix} n_1 & n_2 \end{matrix} & \end{array},$$

где $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$, $i = 1, 2$ и $j = 1, 2$.

Результатом произведения 2×2-блочной матрицы A на двумерный блочный вектор x будет двумерный блочный вектор z:

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix}=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}=\begin{pmatrix} A_{11}x_1 & + & A_{12}x_2 \\ A_{21}x_1 & + & A_{22}x_2 \end{pmatrix}.$$

4.2.10. Двумерные матрицы вращения и отражения

В численных методах матрицы вращения и отражения часто используются для обнуления отдельных элементов векторов и матриц. В двумерном случае матрица вращения, поворачивающая вектор против часовой стрелки на угол θ, равна

$$G=\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

Получаемый в результате вращения вектор $y = G^T x$. Пусть $x = (x_1, x_2)^T$. Вычислим угол θ, при котором второй компонент вектора $y = G^T x$ будет равен нулю. Поскольку

$$\begin{aligned} y_1 &= \cos\theta x_1 - \sin\theta x_2, \\ y_2 &= \sin\theta x_1 + \cos\theta x_2, \end{aligned}$$

то y_2 обратится в нуль, если

$$\cos\theta = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \text{ и } \sin\theta = \frac{-x_2}{\sqrt{x_1^2 + x_2^2}}.$$

Пример. Выполняется поворот вектора $x = (1, \sqrt{3})^T$, в результате которого второй элемент преобразованного вектора оказывается равным нулю.

```
program rot
 real(4) :: x(2), y(2), c, s, r
 x = (/ 1.0, sqrt(3.0) /)
 r = sqrt(x(1) * x(1) + x(2) * x(2))
 c = x(1) / r;  s = -x(2) / r                  ! Элементы матрицы вращения G
 print *, asind(s)                             !  60.0   (Угол поворота)
 ! Вычисляем y = G^T x (выполняем поворот на угол -60°)
 y = (/        (c * x(1) - s * x(2)),                    &
               (s * x(1) + c * x(2)) /)
 print *, y                                    !  2.0   0.0
end program rot
```

Угол между векторами $x = (1, \sqrt{3})^T$ и e_1 равен θ = 60°. Поэтому вектор $y = (2,\ 0)$ можно получить, выполнив отражение x относительно прямой, имеющей с e_1 угол θ/2 = 30°. Матрица отражения

$$H = \begin{pmatrix} \sin 30 & \cos 30 \\ \cos 30 & -\sin 30 \end{pmatrix}.$$

Таким образом, $y = Hx$.

Матрицы вращения и отражения играют большую роль в матричных вычислениях, позволяя в результате обнуления элементов векторов и матриц приводить их к простому виду.

Замечание. Матрица вращения называется *матрицей Гивенса* или *матрицей Якоби*, а матрица отражения – *матрицей Хаусхолдера*.

4.2.11. Линейная независимость и базис

Линейной комбинацией векторов $x \in \mathbf{R}^n$ и $y \in \mathbf{R}^n$ является вектор $z \in \mathbf{R}^n$

$$z = \alpha x + \beta y,$$

где α и β – коэффициенты линейной комбинации, α и β ∈ **R**.

Векторы $x^{(1)}, x^{(2)}, ..., x^{(m)}$, где $x^{(i)} \in \mathbf{R}^n$, называются *линейно зависимыми*, если существуют числа $\alpha_1, \alpha_2, ..., \alpha_m$, не все равные нулю и такие, что

$$\alpha_1 x^{(1)} + \alpha_2 x^{(2)} + ... + \alpha_m x^{(m)} = 0. \qquad (4.2)$$

Из приведенного определения следует, что данные векторы линейно зависимы, когда один из них является линейной комбинацией других. Действительно, пусть $\alpha_m \neq 0$, тогда

$$x^{(m)} = \frac{\alpha_1}{\alpha_m} x^{(1)} + \frac{\alpha_2}{\alpha_m} x^{(2)} + ... + \frac{\alpha_{m-1}}{\alpha_m} x^{(m-1)}.$$

Если же равенство (4.2) выполняется только тогда, когда $\alpha_1 = \alpha_2 = ... = \alpha_m = 0$, то векторы $x^{(1)}, x^{(1)}, ..., x^{(m)}$ называются *линейно независимыми*. Иными словами, векторы линейно независимы, когда никакая их линейная комбинация, не все коэффициенты которой равны нулю, не является нуль-вектором, т. е. вектором, все компоненты которого равны нулю.

Базисом пространства называется линейно независимая система векторов, через которые линейно выражается каждый вектор пространства. В линейном пространстве $\mathbf{R}^n$ любая линейно независимая система из n векторов образует базис, а любая система из $n + 1$ векторов линейна зависима. Это означает, что любой вектор $x \in \mathbf{R}^n$ может быть представлен в виде линейной комбинации

$$x = \alpha_1 e_1 + \alpha_2 e_2 + ... + \alpha_n e_n,$$

где $e_1, e_2, ..., e_n$ – базис $\mathbf{R}^n$, $e_i \in \mathbf{R}^n$, а числа $\alpha_1, \alpha_2 ..., \alpha_n \in \mathbf{R}$ называются *координатами вектора* x относительно этого базиса. Разложение вектора x по заданному базису единственно.

4.2.12. Ортогональность векторов

Векторы x, y называются *ортогональными*, если $(x, y) = 0$. Система векторов евклидова пространства называется *ортогональной*, если либо она состоит из одного вектора, либо ее векторы попарно ортогональны. Нормированная ортогональная система называется *ортонормированной*. Такая система является линейно независимой.

Базис, векторы которого образуют ортонормированную систему, называется *ортонормированным*. В линейной алгебре доказывается, что в любом конечномерном евклидовом пространстве такой базис существует, см. [5].

4.2.13. Ранг матрицы

Ранг матрицы $A \in \mathbf{R}^{m \times n}$ – обозначается $rank(A)$ – равен максимальному числу линейно независимых ее строк (или столбцов). Причем ранг системы строк матрицы равен рангу системы ее столбцов. Отсюда, $rank(A) = rank(A^T)$. Ранг прямоугольной матрицы не может быть больше $\min(m, n)$.

Также справедливо: наивысший порядок отличных от нуля миноров матрицы A равен рангу этой матрицы. Легко видеть, что ранг диагональной матрицы равен числу неравных нулю диагональных элементов. Ранг произведения AB матриц A и B не выше ранга сомножителей.

Пример. Ранг 5×4-матрицы

$$A=\left(a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}\right)=\begin{pmatrix} 1 & -2 & 4 & 3 \\ 3 & -4 & 8 & 7 \\ 2 & -3 & 9 & 8 \\ 5 & -6 & 7 & 6 \\ 4 & -5 & 3 & 2 \end{pmatrix}$$

равен трем, поскольку $a^{(4)}=a^{(1)}+a^{(2)}+a^{(3)}$.

Говорят, что матрица $A \in \mathbf{R}^{m\times n}$ имеет *полный столбцовый ранг*, если все ее столбцы линейно независимы.

Простой способ определения ранга матрицы – это приведение ее элементарными преобразованиями (разд. 4.2.4) к диагональной форме. Например, матрица

$$A=\begin{pmatrix} 0 & -1 & 3 & 0 & 2 \\ 2 & -4 & 1 & 5 & 3 \\ -4 & 5 & 7 & -10 & 0 \end{pmatrix}$$

элементарными преобразованиями приводится к виду

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Следовательно, $rank(A) = 2$.

4.2.14. Векторные нормы

В процедурах библиотеки IMSL используются *p-нормы*, определяемые как

$$\| x \|_p=\left(| x_1|^p+\ldots+| x_n|^p\right)^{1/p},\ p\ge 1.$$

Причем используются 1-, 2- и ∞-нормы:

$$\| x \|_1=| x_1|+\ldots+| x_n|,$$

$$\| x \|_2=\left(| x_1|^2+\ldots+| x_n|^2\right)^{1/2}=(x^Tx)^{1/2},$$

$$\| x \|_\infty=\max_{1\le i\le n}| x_i|.$$

<u>*Замечание*</u>. Норму $\| x\|_2$ называют также *евклидовой нормой*.

Норма $\|x\|$ является вещественным числом и в вещественных ($x \in \mathbf{R}^n$) и в комплексных ($x \in \mathbf{C}^n$) пространствах.

Для норм любых векторов справедливо:

1) $\|x\| > 0$, если $x \neq 0$, и $\|x\| = 0$, если $x = 0$;
2) $\|\lambda x\| = |\lambda|\|x\|$, где λ – любое число;
3) $\|x + y\| \leq \|x\| + \|y\|$.

Вектор x называется *нормированным* по отношению к норме $\|\cdot\|$, если $\|x\| = 1$. Также такой вектор называют *единичным*.

Линейные пространства, в которых каждому вектору поставлена в соответствие норма, называются *нормированными*.

Нормы в векторных пространствах служат для измерения расстояний (ρ) между векторами:

$$\rho(x, y) = \|x - y\|.$$

Равенство $\rho(x, y) = 0$ означает, что $x = y$. Заметим, что $\|x\| = \rho(x, 0)$.

В евклидовом пространстве $\rho(x, y)$ есть расстояние между точками x и y, а $\rho(x, 0)$ – длина вектора x.

4.2.15. Матричные нормы

В процедурах IMSL используются матричная *норма Фробениуса*

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\left|a_{ij}\right|^2}, \quad A \in \mathbf{R}^{m\times n}, \ \|A\|_F \in \mathbf{R}$$

и *p-нормы*

$$\|A\|_p = \max_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p}, \quad A \in \mathbf{R}^{m\times n}, \ x \in \mathbf{R}^n, \ \|A\|_p \in \mathbf{R}. \tag{4.3}$$

Норма $\|A\|_p$ является p-нормой наибольшего вектора, полученного действием A на нормированные (единичные) в p-норме векторы:

$$\|A\|_p = \max_{\|x_p\|=1}\|Ax\|_p.$$

Матричные 1-, 2- и ∞-нормы ($A \in \mathbf{R}^{m\times n}$):

$$\|A\|_1 = \max_{1\leq j\leq n}\sum_{i=1}^{m}\left|a_{ij}\right|,$$

$$\|A\|_2 = \max_{x\neq 0}\frac{\|Ax\|_2}{\|x\|_2} = \max_{x\neq 0}\sqrt{\frac{x^T A^T Ax}{x^T x}} = \max_{1\leq j\leq n}\left(\lambda_j(A^T A)\right)^{1/2},$$

где $\lambda_j(A^T A)$ – собственные значения матрицы $A^T A$,

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}| .$$

Значения норм $\|A\|_1$ и $\|A\|_\infty$ вычисляются просто. Найдем их, например, для матрицы

$$A = \begin{pmatrix} 1 & -1 & 2 & 3 \\ -1 & 5 & 4 & 2 \\ 2 & 4 & 14 & 1 \end{pmatrix}.$$

$$\|A\|_1 = \max(1+1+2,\ 1+5+4,\ 2+4+14,\ 3+2+1) = \max(4,\ 10,\ 20,\ 6) = 20,$$

$$\|A\|_\infty = \max(1+1+2+3,\ 1+5+4+2,\ 2+4+14+1) = \max(7,\ 12,\ 21) = 21.$$

Вычисление 2-нормы матрицы, связанное с поиском собственных значений симметрической матрицы A^TA, существенно сложнее.

Норме (4.3) можно дать такую геометрическую интерпретацию. Будем рассматривать $\|x\|$ как длину вектора x. Результатом произведения Ax также является вектор, длина которого $\|Ax\|$. Тогда отношение $\|Ax\|/\|x\|$ есть коэффициент растяжения вектора x под действием матрицы A. В такой интерпретации норма (4.3) есть максимальный коэффициент растяжения ненулевых векторов.

Как и векторные, матричные нормы используются для оценки расстояний в пространстве матриц, например для выяснения, насколько матрица близка к вырожденной.

Замечание. Приведенные формулы для вычисления матричных норм справедливы и для комплексных пространств ($A \in \mathbf{C}^{m\times n}, x \in \mathbf{C}^n$). Сама же норма является вещественным числом – $\|A\| \in \mathbf{R}$.

4.2.16. Нормы и ортогональные преобразования

Преобразования, выполняемые под действием ортогональных матриц, называются *ортогональными*. При таких преобразованиях 2-норма вектора сохраняет свое значение. Действительно, если Q – ортогональная матрица ($Q^TQ = I$), то $\|Qx\|_2^2 = x^TQ^TQx = x^Tx = \|x\|_2^2$. То же справедливо и для матричных 2-нормы и нормы Фробениуса: $\|QAP\|_2 = \|A\|_2$ и $\|QAP\|_F = \|A\|_F$, где Q и P – ортогональные матрицы подходящих размеров.

4.3. ПРЕДСТАВЛЕНИЕ МАТРИЦ В ПРОГРАММАХ

На практике часто вместо матрицы общего вида встречаются трехдиагональные, разреженные, ленточные матрицы, в которых большое число эле-

ментов равно нулю. Для подобных матриц есть более компактные формы представления и, как правило, более эффективные связанные с такого рода матрицами модификации алгоритмов. В этом разделе приводятся способы представления наиболее часто употребляемых в IMSL видов матриц.

4.3.1. Представление всех элементов матрицы

Матрица, если она не представляется как ленточная или разреженная, называется *полной*. Полные квадратные общего вида, симметрические и эрмитовы матрицы, имеющие размер $n \times n$, и прямоугольные $m \times n$-матрицы хранятся в процедурах IMSL в массивах, содержащих все элементы матриц:

```
integer(4), parameter :: Lda = 100, n = 50
тип матрицы, dimension(Lda, n) :: a      ! a – массив, представляющий матрицу A
```

Такое представление матриц в программах будем называть *общим представлением*. Матрица может иметь тип REAL(4), REAL(8), COMPLEX(4) или COMPLEX(8).

Параметр *Lda* называется *ведущим размером* матрицы, причем в случае квадратной матрицы $Lda \geq n$, а в случае прямоугольной – $Lda \geq m$. Как правило, есть все основания принять $Lda = n$ в случае квадратной матрицы и $Lda = m$ в случае прямоугольной.

Таким же образом объявляются и треугольные матрицы. Причем в случае *LU*-разложения для хранения его нижней и верхней треугольных матриц используется один массив формы (*Lda*, *n*). Нижняя треугольная матрица хранится в нижней части массива, и ее диагональные элементы используются со значением 1. Верхняя треугольная матрица занимает главную диагональ массива и его верхнюю, расположенную над главной диагональю часть.

Базовым процедурам линейной алгебры, принимающим массив, хранящий треугольные матрицы, можно указать, какая часть массива должна быть использована. Эта информация передается параметром *uplo*. Если *uplo* равен 'U', то будет использована верхняя часть массива, представляющая верхнюю треугольную матрицу. В случае 'L' – нижняя. Также, употребив параметр *diag*, можно запретить процедурам обращаться к диагональным элементам треугольных матриц. Для этого следует передать в *diag* значение 'U', указывающее, что на главной диагонали массива расположены единицы. Значение *diag*, равное 'N', говорит о необходимости использовать и элементы главной диагонали. Тип *diag* и *uplo* – CHARACTER(1).

Замечание. Матрицы, размеры которых определяются в процессе вычислений, следует объявлять с атрибутом ALLOCATABLE, с тем чтобы затем выделить под них необходимую память, например:

```
real(4), allocatable, dimension(:, :) :: a
ввод значения n
allocate(a(n, n))
```

4.3.2. Ленточные матрицы

Ленточные матрицы могут быть представлены более компактно. Так, 5×5-матрица

$$A=\begin{pmatrix} 3 & 2 & 7 & 0 & 0 \\ 2 & 5 & 9 & 4 & 0 \\ 0 & 9 & 8 & 3 & 5 \\ 0 & 0 & 4 & 2 & 6 \\ 0 & 0 & 0 & 2 & 9 \end{pmatrix}$$

с шириной нижней и верхней лент *nlca* = 1 и *nuca* = 2 представляется в программе массивом *a*:

```
integer(4), parameter :: n = 5, nlca = 1, nuca = 2
real(4), dimension(nlca + nuca + 1, n) :: a
```

в котором в каждой строке расположены элементы кодиагоналей или главной диагонали матрицы *A*:

$$\begin{pmatrix} \times & \times & 7 & 4 & 5 \\ \times & 2 & 9 & 3 & 6 \\ 3 & 5 & 8 & 2 & 9 \\ 2 & 9 & 4 & 2 & \times \end{pmatrix}.$$

Символом × отмечены элементы, к которым IMSL не обращается. В них могут быть установлены любые, например нулевые, значения.

4.3.3. Ленточные симметрические матрицы

В ленточной симметрической, например, 5×5-матрице

$$A=\begin{pmatrix} 3 & 2 & 0 & 0 & 0 \\ 2 & 5 & 9 & 0 & 0 \\ 0 & 9 & 8 & 4 & 0 \\ 0 & 0 & 4 & 2 & 6 \\ 0 & 0 & 0 & 6 & 9 \end{pmatrix}$$

nlca = *nuca* = *ncoda* = 1. Для представления подобного рода матриц достаточно хранить в массиве *a* элементы верхних (нижних) кодиагоналей и главной диагонали:

```
integer(4), parameter :: n = 5, ncoda = 1    ! ncoda – число верхних (нижних) кодиагоналей
real(4), dimension(ncoda + 1, n) :: a
```

При таком объявлении массив *a* в случае приведенной выше матрицы *A* инициализируется так:

$$\begin{pmatrix} \times & 2 & 9 & 4 & 6 \\ 3 & 5 & 8 & 2 & 9 \end{pmatrix}$$

или, если используется нижняя кодиагональ, так:

$$\begin{pmatrix} 3 & 5 & 8 & 2 & 9 \\ 2 & 9 & 4 & 6 & \times \end{pmatrix}.$$

Символом × отмечены элементы, к которым IMSL не обращается. В них могут быть установлены любые, например нулевые, значения.

Базовым процедурам линейной алгебры, принимающим массив, хранящий ленточные симметрические матрицы, следует указать, какие кодиагонали, нижние или верхние, представлены в массиве. Эта информация передается параметром *uplo*, имеющим тип CHARACTER(1). Если *uplo* равен 'U', то в массиве должны представляться верхние кодиагонали, если – 'L', то – нижние.

4.3.4. Ленточные эрмитовы матрицы

Ленточная эрмитова матрица A имеет тип COMPLEX(4) или COMPLEX(8), она является аналогом ленточной симметрической матрицы и имеет вид:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ \bar{a}_{21} & a_{22} & a_{23} & a_{24} & 0 \\ \bar{a}_{31} & \bar{a}_{32} & a_{33} & a_{34} & a_{35} \\ 0 & \bar{a}_{42} & \bar{a}_{43} & a_{44} & a_{45} \\ 0 & 0 & \bar{a}_{53} & \bar{a}_{54} & a_{55} \end{pmatrix}.$$

В ленточной эрмитовой матрице $a_{ij} = \bar{a}_{ji}$, где $\bar{a}_{ji}$ - число, комплексно-сопряженное с a_{ij}.

Для ее представления используется массив a:

```
integer(4), parameter :: n = 5, ncoda = 2
complex(4), dimension(ncoda + 1, n) :: a   ! или complex(8)
```

хранящий информацию о главной диагонали и верхних или нижних кодиагоналях:

$$\begin{pmatrix} \times & \times & a_{13} & a_{24} & a_{35} \\ \times & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{pmatrix} \text{ или } \begin{pmatrix} a_{11} & \underline{a}_{22} & \underline{a}_{33} & \underline{a}_{44} & \underline{a}_{55} \\ \times & a_{21} & \underline{a}_{32} & \underline{a}_{43} & \underline{a}_{54} \\ \times & \times & a_{31} & a_{42} & a_{53} \end{pmatrix}.$$

Базовым процедурам линейной алгебры, принимающим массив, хранящий ленточные эрмитовы матрицы, надо указать, какие кодиагонали, нижние или верхние, представлены в массиве. Эта информация передается параметром *uplo*, смысл которого раскрыт в предшествующем разделе.

4.3.5. Ленточные треугольные матрицы

Ленточная треугольная 5×5-матрица

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{pmatrix}$$

с числом кодиагоналей $ncoda = 2$ представляется массивом a:

```
integer(4), parameter :: n = 5, ncoda = 2
real(4), dimension(ncoda + 1, n) :: a          ! или real(8)
```

в который элементы матрицы заносятся в следующем порядке:

$$\begin{pmatrix} \times & \times & a_{13} & a_{14} & a_{35} \\ \times & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{pmatrix}.$$

Транспонированная матрица A^T представляется массивом

$$\begin{pmatrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{12} & a_{23} & a_{34} & a_{45} & \times \\ a_{13} & a_{24} & a_{35} & \times & \times \end{pmatrix}.$$

К отмеченным символом × элементам IMSL не обращается.

Информация о варианте представления матрицы передается в процедуры базовой линейной алгебры ранее описанным параметром *uplo*. В случае нетранспонированной ленточной треугольной матрицы *uplo* = 'U', при транспонированной *uplo* = 'L'. Кроме *uplo*, базовым процедурам линейной алгебры, работающим с такими матрицами, передается параметр *diag*, смысл и значения которого описаны в разд. 4.3.1.

4.3.6. Кодиагональное представление ленточных симметрических матриц

В процедурах, использующих для решения системы линейных уравнений $Ax = b$ метод Холецкого, для $n \times n$-матрицы A, если она является ленточной симметрической и имеет *ncoda* кодиагоналей, используется массив a формы (Lda, $ncoda + 2$). Причем $Lda \geq n + ncoda$. Последний столбец массива используется для хранения вектора b. Например, если

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} \\ 0 & 0 & a_{53} & a_{54} & a_{55} \end{pmatrix} \text{ и } b=\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix},$$

то массив объявляется, например, так:

```
integer(4), parameter :: n = 5, ncoda = 2, Lda = n + ncoda
real(4), dimension(Lda, ncoda + 2) :: a      ! или real(8)
```

и должен быть инициализирован следующим образом:

$$\begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ a_{11} & \times & \times & b_1 \\ a_{22} & a_{12} & \times & b_2 \\ a_{33} & a_{23} & a_{13} & b_3 \\ a_{44} & a_{34} & a_{24} & b_4 \\ a_{55} & a_{45} & a_{35} & b_5 \end{pmatrix}.$$

Элементы, отмеченные символом ×, могут быть инициализированы, например, нулем. Эти элементы нужны некоторым процедурам IMSL. Решение системы, вектор x, записывается в элементы, хранящие вектор b. Причем x_i замещает b_i. Такая структура данных обеспечивает хорошую производительность при вычислениях по методу Холецкого.

4.3.7. Кодиагональное представление ленточных эрмитовых матриц

Аналогично и для тех же целей, приведенных в предшествующем разделе, некоторые процедуры IMSL используют кодиагональное представление ленточной эрмитовой матрицы. Поскольку она содержит комплексные элементы, ее можно представить в виде суммы вещественных матриц U и V: $A=U+\sqrt{-1}\,V$. Вектор x – решение системы линейных уравнений $Ax=b$ – представляется как сумма вещественных векторов e и f: $x=e+\sqrt{-1}\,f$, а вектор b – как сумма вещественных векторов c и d: $b=c+\sqrt{-1}\,d$. Рассмотрим, например, ленточную эрмитову матрицу

$$A=\begin{pmatrix} u_{11} & u_{12} & u_{13} & 0 & 0 \\ u_{21} & u_{22} & u_{23} & u_{24} & 0 \\ u_{31} & u_{32} & u_{33} & u_{34} & u_{35} \\ 0 & u_{42} & u_{43} & u_{44} & u_{45} \\ 0 & 0 & u_{53} & u_{54} & u_{55} \end{pmatrix}+\sqrt{-1}\begin{pmatrix} 0 & v_{12} & v_{13} & 0 & 0 \\ -v_{21} & 0 & v_{23} & v_{24} & 0 \\ -v_{31} & -v_{32} & 0 & v_{34} & v_{35} \\ 0 & -v_{42} & -v_{43} & 0 & v_{45} \\ 0 & 0 & -v_{53} & -v_{54} & 0 \end{pmatrix}$$

и вектор

$$b=\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix}+\sqrt{-1}\begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{pmatrix}.$$

В программе матрица и вектор заносятся в массив a:

```
integer(4), parameter :: n = 5, ncoda = 2, Lda = n + ncoda
real(4), dimension(Lda, 2 * ncoda + 3) :: a          ! или real(8)
```

который инициализируется так:

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ u_{11} & \times & \times & \times & \times & c_1 & d_1 \\ u_{22} & u_{12} & v_{12} & \times & \times & c_2 & d_2 \\ u_{33} & u_{23} & v_{23} & u_{13} & v_{13} & c_3 & d_3 \\ u_{44} & u_{34} & v_{34} & u_{24} & v_{24} & c_4 & d_4 \\ u_{55} & u_{45} & v_{45} & u_{35} & v_{35} & c_5 & d_5 \end{pmatrix}.$$

Элементы, отмеченные символом ×, можно не инициализировать. Векторы e и f – решение системы $Ax = b$ – замещают векторы c и d: e_i замещает c_i, а $f_i - d_i$.

4.3.8. Разреженные матрицы

Рассмотрим разреженную 6×6-матрицу

$$A=\begin{pmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{pmatrix}.$$

Для представления такой матрицы используем два целочисленных вектора *irow* и *jcol* и вектор a (вещественный или комплексный). В *irow* запишем номера строк ненулевых элементов матрицы A, в *jcol* – номера их столбцов, а в вектор a – соответствующие ненулевые элементы. Такая форма представления матрицы называется *координатной формой*.

Разместим A в файле a.txt.

Состав файла a.txt:

10	0	0	0	0	0
0	10	-3	-1	0	0
0	0	15	0	0	0
-2	0	0	10	-1	0
-1	0	0	-5	1	-3
-1	-2	0	0	0	6

Сформируем, обрабатывая файл, необходимые массивы:

```
integer(4) :: i, j, n = 6, nz                  ! nz – число ненулевых элементов матрицы
integer(4), dimension(:), allocatable :: irow, jcol
real(4), dimension(:), allocatable :: a, temp
allocate(temp(n))
nz = 0
open(10, file = 'a.txt')                       ! Подсоединяем файл к устройству 10
do i = 1, n                                    ! Поиск nz
  read(10, *) temp                             ! Читаем строчку файла
  nz = nz + count(temp /= 0.0)
end do
allocate(a(nz), irow(nz), jcol(nz))            ! Выделяем память под массивы
nz = 0
rewind(10)                                     ! Переход на начало файла
do i = 1, n                                    ! Формирование массивов irow, icol и a
  read(10, *) temp                             ! Читаем строчку файла
  do j = 1, n
    if(temp(j) /= 0.0) then
      nz = nz + 1
      irow(nz) = i; jcol(nz) = j; a(nz) = temp(j)
    end if
  end do
end do
close(10)
deallocate(temp)                               ! Освобождаем память
print '(1x, "irow", 20i4)', irow               ! Контрольный вывод результата
print '(1x, "jcol", 20i4)', jcol
print '(1x, "a   ", 20f4.0)', a
```

Результат:

irow	1	2	2	2	3	4	4	4	5	5	5	5	6	6
jcol	1	2	3	4	3	1	4	5	1	4	5	6	1	2
a	10.	10.	-3.	-1.	15.	-2.	10.	-1.	-1.	-5.	1.	-3.	-1.	-2.

4.4. БАЗОВЫЕ ПРОЦЕДУРЫ ЛИНЕЙНОЙ АЛГЕБРЫ УРОВНЯ 1

4.4.1. Замечания для программиста

Типы параметров, результата и промежуточных данных процедур БПЛА можно определить по первой или двум первым буквам имени процедуры. При этом тип результата всегда характеризуется первой буквой имени. Взаимосвязь между типами данных и первыми буквами процедур и параметров приведена в табл. 4.1.

Таблица 4.1. Типы данных БПЛА

Типы параметров, результата и промежуточных данных	*Первые буквы процедур и/или параметров*
INTEGER(4)	I или N
REAL(4)	S
COMPLEX(4)	C
REAL(8)	D
COMPLEX(8)	Z
REAL(4) и REAL(8)	SD
COMPLEX(4) и COMPLEX(8)	CZ
REAL(8) и REAL(16)	DQ
COMPLEX(8) и COMPLEX(16)	ZQ

Замечание. Обозначения REAL(16) и COMPLEX(16) употреблены для указания точности, которая используется в процедуре БПЛА для промежуточных данных. Такая точность достигается за счет применения аккумулятора: для REAL(16) аккумулятором служит массив типа REAL(8) из двух элементов, для COMPLEX(16) – массив типа REAL(8) из четырех элементов.

Например, в подпрограмме

CALL ZDSCAL(*n*, *da*, *zx*, *incx*)

параметры *n* и *incx* – типа INTEGER(4), *da* – REAL(4) и *zx* – COMPLEX(8). А в функции

dw = DSDOT(*n*, *sx*, *incx*, *sy*, *incy*)

параметры *n*, *incx* и *incy* – типа INTEGER(4), *sx* и *sy* – REAL(4). Тип функции – REAL(4).

Процедуры одного назначения, работающие с разными типами данных, образуют группу процедур. Так, группу образуют процедуры

CALL ISET(*n*, *ia*, *ix*, *incx*)

CALL SSET(*n*, *sa*, *sx*, *incx*); CALL DSET(*n*, *da*, *dx*, *incx*)

CALL CSET(n, ca, cx, $incx$); CALL ZSET(n, za, zx, $incx$)

выполняющие присваивание вектору скаляра. Для экономии места при описании подпрограмм используются обобщенные имена групповой подпрограммы и ее параметров. Например:

CALL {G}SET(n, ga, gx, $incx$)

В этом имени {G} – это I, S, D, C или Z, а g – это i, s, d, c или z. В дальнейшем расшифровки g приводиться не будет, поскольку всегда g повторяет значения {G} или первого символа {G}, например если {G} – это CS, то g – это c. Синтаксис процедур, не входящих в группу, указывается как есть. Для функций обобщенные имена не вводятся.

Процедуры, работающие с вектором, например с вектором x, имеют параметры n, sx и $incx$. Первый параметр задает число итераций, sx – первый передаваемый элемент вектора, и $incx$ – расстояние между его элементами. Вектор x и параметры связаны соотношениями:

$$x_i = \begin{cases} sx((i-1)incx+1), & incx \geq 0, \\ sx((i-1)incx+1), & incx < 0. \end{cases}$$

Например, вызов

```
integer(4) :: ix(20), ia = 2
call iset(10, ia, ix, 1)
```

означает, что в подпрограмму ISET передается вектор ix, начиная с первого элемента, и для вычислений будет использовано 10 его подряд расположенных элементов.

Вызов

```
integer(4) :: ix(20), ia = 2
call iset(6, ia, ix(10), 2)
```

означает, что в подпрограмму ISET передается вектор ix, начиная с 10-го элемента, и для вычислений будет использовано 5 его элементов с именами $ix(10)$, $ix(12)$, ..., $ix(20)$.

К сожалению, БПЛА не контролируют нарушений нижней и верхней границ векторов, поэтому необходимо следить за значением параметра n, поскольку в общем случае результаты при выходе за границы вектора непредсказуемы.

Отрицательное значение $incx$ означает цикл с отрицательным шагом вида

```
do i = n, 1, incx
```

Для процедур с одним вектором $incx$ должен быть больше нуля.

В процедурах БПЛА уровня 1 могут присутствовать следующие объявления формальных параметров:

```
implicit integer(4) (i-n)
```

```
implicit real(4) s
implicit real(8) d
implicit complex(4) c
implicit complex(8) z
integer(4) ix(mx)
real(4) sx(mx), sy(my), sz(mz), sparam(5)
real(8) dx(mx), dy(my), dz(mz), dparam(5)
real(8) dacc(2), dzacc(4)
complex(4) cx(mx), cy(my)
complex(8) zx(mx), zy(my)
```

в которых

```
mx = MAX(1, 1 + (n - 1) * ABS(incx))
my = MAX(1, 1 + (n - 1) * ABS(incy))
mz = MAX(1, 1 + (n - 1) * ABS(incz))
```

Порядок применения БПЛА иллюстрируется примерами.

4.4.2. Список процедур уровня 1

Перечень процедур с указанием типа результата приведен в табл. 4.2. Процедуры, номер которых отмечен *, используют аккумулятор двойной точности. Символ ← является аналогом символа присваивания =.

Таблица 4.2. БПЛА уровня 1

№	*Действия*	*Процедуры*				
		Integer(4)	Real(4)	Real(8)	Complex(4)	Complex(8)
1	$x_i \leftarrow \alpha$	ISET	SSET	DSET	CSET	ZSET
2	$y_i \leftarrow x_i$	ICOPY	SCOPY	DCOPY	CCOPY	ZCOPY
3	$x_i \leftarrow \alpha x_i$		SSCAL	DSCAL	CSCAL CSSCAL	ZSCAL ZDSCAL
4	$y_i \leftarrow \alpha x_i$		SVCAL	DVCAL	CVCAL CSVCAL	ZVCAL ZDVCAL
5	$x_i \leftarrow x_i + \alpha$	IADD	SADD	DADD	CADD	ZADD
6	$x_i \leftarrow \alpha - x_i$	ISUB	SSUB	DSUB	CSUB	ZSUB
7	$y_i \leftarrow \alpha x_i + y_i$		SAXPY	DAXPY	CAXPY	ZAXPY
8	$y_i \leftrightarrow x_i$	ISWAP	SSWAP	DSWAP	CSWAP	ZSWAP
9	(x, y)		SDOT	DDOT	CDOTU	ZDOTU
10	$(\bar{x}, y)$				CDOTC	ZDOTC
11*	(x, y)		DSDOT		CZDOTU	ZQDOTU
12*	$(\bar{x}, y)$				CZDOTC	ZQDOTC
13*	$\alpha + (x, y)$		SDSDOT	DQDDOT	CZUDOT	ZQUDOT

14*	$\alpha + (\bar{x}, y)$				CZCDOT	ZQCDOT
15*	$\beta + (x, y)$		SDDOTI	DQDOTI	CZDOTI	ZQDOTI
17	$z_i \leftarrow x_i y_i$		SHPROD	DHPROD		
18*	$ACC + \beta + (x, y)$		SDDOTA	DQDOTA	CZDOTA	ZQDOTA
18	$\sum x_i y_i z_i$		SXYZ	DXYZ		
19	$\sum x_i$	ISUM	SSUM	DSUM		
20	$\sum \lvert x_i \rvert$		SASUM	DASUM	SCASUM	DZASUM
21	$\lVert x \rVert_2$		SNRM2	DNRM2	SCNRM2	DZNRM2
22	Πx_i		SPRDCT	DPRDCT		
23	$i : x_i = \min_j x_j$	IIMIN	ISMIN	IDMIN		
24	$i : x_i = \max_j x_j$	IIMAX	ISMAX	IDMAX		
25	$i : \lvert x_i \rvert = \min_j \lvert x_j \rvert$		ISAMIN	IDAMIN	ICAMIN	IZAMIN
26	$i : \lvert x_i \rvert = \max_j \lvert x_j \rvert$		ISAMAX	IDAMAX	ICAMAX	IZAMAX
29	Построение матрицы вращения Гивенса		SROTG	DROTG		
27	Вращение Гивенса		SROT	DROT	CSROT	ZDROT
28	Построение матрицы быстрого вращения Гивенса		SROTMG	DROTMG		
29	Быстрое вращение Гивенса		SROTM	DROTM	CSROTM	ZDROTM

4.4.3. Присваивание вектору скаляра

Присваивание вида $x = \alpha$, где x – вектор, а α – скаляр, реализует подпрограмма

CALL {G}SET(n, ga, gx, $incx$) ! {G} – это I, S, D, C или Z

Также подобное присваивание выполняется непосредственно, например:

```
real(4), x(10), a = 2.5
x = a                                   ! Присваивание скаляра
```

Пример 1:

```
program go
 use msimsl
 integer(4), parameter :: n = 20
 integer(4) :: ix(n), ia = 2
 complex(4) :: cx(n), ca = (2.5_4, 2.5_4)
 call iset(n, ia, ix, 1)
```

```
 call cset(n, ca, cx, 1)
 ix(n/2:) = 3                          ! Меняем значения ix(n/2), ..., ix(n)
 print *, ix(3), ix(15), cx(5)         !   2    3    (2.500000,2.500000)
end program go
```

Пример 2. Присвоить элементам вектора *ix* с четными индексами значение -3, а с нечетными – +7. Присваивание выполнить начиная с 10-го элемента вектора. Число элементов, получающих значение +7, должно быть равно трем.

```
program go
 use msimsl
 integer(4), parameter :: n = 20
 integer(4) :: ix(n) = 0, ia1 = -3, ia2 = 7
 call iset(6, ia1, ix(10), 2)
 call iset(3, ia2, ix(11), 2)
 print '(1x, 30i3)', ix(5:)            !0 0 0 0 0 -3 7 -3 7 -3 7 -3 0 -3 0 -3
 ! Выполним то же, использовав присваивание массивов
 ix = 0
 ix(10::2) = ia1
 ix(11:15:2) = ia2
 print '(1x, 30i3)', ix(5:)            !0 0 0 0 0 -3 7 -3 7 -3 7 -3 0 -3 0 -3
end program go
```

4.4.4. Копирование вектора

Присваивание вида $y = x$, где x и y – векторы, выполняет подпрограмма

CALL {G}COPY(*n*, *gx*, *incx*, *gy*, *incy*) ! {G} – это I, S, D, C или Z

Для любого типа данных такое же присваивание выполняется непосредственно:

```
complex(8) :: zx(10) = (1.1_8, 1.1_8), zy(10)
zy = zx                                ! Присваивание массива
```

Пример. Скопировать в вектор *iy* элементы *ix* в обратном порядке.

```
program go
 use msimsl
 integer(4), parameter :: n = 10
 integer(4) :: i, ix(n) = (/ (i, i = 1, n) /)
 call icopy(n, ix, 1, iy, -1)
 print '(1x, 30i3)', ix                !   1 2 3 4 5 6 7 8 9 10
 print '(1x, 30i3)', iy                !   10 9 8 7 6 5 4 3 2 1
 ! Выполним то же, использовав присваивание массивов
 ix = (/ (i, i = 1, n) /)
 iy = ix(n:1:-1)
 print '(1x, 30i3)', ix                !   10 9 8 7 6 5 4 3 2 1
end program go
```

Пример. Скопировать в вектор *b* диагональные элементы матрицы *A*.

```
program go2
```

```
use msimsl
integer(4), parameter :: n = 10
integer(4) :: i, a(n, n) = 0, b(n) = 0
do i = 1, n                          ! Цикл задания значений элементов
  a(i, i) = i                        ! на диагонали матрицы a
end do
! Расстояние между диагональными элементами равно n + 1
call icopy(n, a, n + 1, b, 1)
print '(1x, 30i3)', b                ! 1 2 3 4 5 6 7 8 9 10
end program go2
```

4.4.5. Масштабирование вектора

Присваивание вида $x = \alpha x$ (масштабирование вектора), где x – вектор, а α – скаляр, выполняют подпрограммы

CALL {G}SCAL(*n*, *ga*, *gx*, *incx*) ! {G} – это S, D, C или Z

CALL CSSCAL(*n*, *sa*, *cx*, *incx*); CALL ZDSCAL(*n*, *da*, *zx*, *incx*)

Предупреждение. В CSSCAL и ZDSCAL скаляры *sa* и *da* вещественные, а векторы комплексные.

Для любого типа данных такое же присваивание выполняется непосредственно:

```
complex(8) :: zx(10) = (1.1_8, 1.1_8), za = (2.1_8, 2.1_8)
zx = za * zx                         ! Масштабирование вектора
```

4.4.6. Умножение вектора на скаляр

Присваивание вида $y = \alpha x$, где x и y – векторы, а α – скаляр, выполняют подпрограммы

CALL {G}VCAL(*n*, *ga*, *gx*, *incx*, *gy*, *incy*) ! {G} – это S, D, C или Z

CALL CSVCAL(*n*, *sa*, *cx*, *incx*, *cy*, *incy*); CALL ZDVCAL(*n*, *da*, *zx*, *incx*, *zy*, *incy*)

Предупреждение. В CSVCAL и ZDVCAL скаляры *sa* и *da* вещественные, а векторы комплексные.

Для любого типа данных такое же присваивание выполняется непосредственно:

```
complex(8) :: zx(10) = (1.1_8, 1.1_8), zy(10)
real(8) :: da = 2.1_8
zy = da * zx                         ! Умножение вектора на скаляр
```

4.4.7. Сумма вектора и скаляра

Присваивание вида $x = x + \alpha$, где x – вектор, а α – скаляр, выполняет подпрограмма

CALL {G}ADD(*n*, *ga*, *gx*, *incx*) ! {G} – это I, S, D, C или Z

Для любого типа данных такое же присваивание выполняется непосредственно:

```
integer(4) :: i
real(4) :: sx(10) = (/ (i, i = 1, 10) /), sa = 3.5
sx = sx + sa                                  ! Прибавление к вектору скаляра
```

4.4.8. Вычитание из скаляра вектора

Присваивание вида $x = \alpha - x$, где x – вектор размера n, а α – скаляр, выполняет подпрограмма

CALL {G}SUB(n, ga, gx, $incx$) ! {G} – это I, S, D, C или Z

Для любого типа данных такое же присваивание выполняется непосредственно:

```
integer(4) :: i
real(4) :: sx(10) = (/ (i, i = 1, 10) /), sa = 3.5
sx = sa - sx                                  ! Вычитание из скаляра вектора
```

4.4.9. Присваивание вида $y = \alpha x + y$

Присваивание вида $y = \alpha x + y$, где x и y – векторы, а α – скаляр, выполняет подпрограмма

CALL {G}AXPY(n, ga, gx, $incx$, gy, $incy$) ! {G} – это I, S, D, C или Z

Для любого типа данных такое же присваивание выполняется непосредственно:

```
integer(4) :: i
real(4) :: sx(10) = (/ (i, i = 1, 10) /), sa = 3.5, sy(10) = (/ (2 * i, i = 1, 10) /)
sy = sa * sx + sy                             ! Присваивание массивов
```

4.4.10. Обмен данными векторов

Векторы x и y обмениваются данными в подпрограммах

CALL {G}SWAP(n, gx, $incx$, gy, $incy$) ! {G} – это I, S, D, C или Z

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 10
 integer(4) :: i, ix(n) = (/ (i, i = 1, 10) /), iy(n) = (/ (2 * i, i = 1, 10) /)
 integer(4), allocatable :: itemp(:)
 call iswap(n, ix, 1, iy, 1)                  ! Обмен данными
 print '(10i3)', ix                           ! 2  4  6  8 10 12 14 16 18 20
 print '(10i3)', iy                           ! 1  2  3  4  5  6  7  8  9 10
 ! Организуем обмен, используя присваивание массивов
 allocate(itemp(n))
 itemp = ix; ix = iy; iy = itemp
 print '(10i3)', ix                           ! 1  2  3  4  5  6  7  8  9 10
```

```
print '(10i3)', iy                    ! 2  4  6  8 10 12 14 16 18 20
end program go
```

4.4.11. Скалярное произведение векторов

Скалярное произведение векторов x и y возвращается функциями

sw = SDOT(*n*, *sx*, *incx*, *sy*, *incy*); *dw* = DDOT(*n*, *dx*, *incx*, *dy*, *incy*)

cw = CDOTU(*n*, *cx*, *incx*, *cy*, *incy*); *cw* = CDOTC(*n*, *cx*, *incx*, *cy*, *incy*)

zw = ZDOTU(*n*, *zx*, *incx*, *zy*, *incy*); *zw* = ZDOTC(*n*, *zx*, *incx*, *zy*, *incy*)

Функции SDOT, DDOT, CDOTU и ZDOTU вычисляют $\sum_{i=1}^{n} x_i y_i$. Функции CDOTC и ZDOTC возвращают $\sum_{i=1}^{n} \bar{x}_i y_i$, где $\bar{x}_i$ - число, комплексно-сопряженное с числом x_i.

Также скалярное произведение векторов любого типа данных возвращается встроенной функцией DOT_PRODUCT, причем если оба вектора комплексного типа, то DOT_PRODUCT вернет $\sum_{i=1}^{n} \bar{x}_i y_i$; чтобы вернуть $\sum_{i=1}^{n} x_i y_i$, следует выполнить

```
complex(4) :: x(10) = (1, 1), y(10) = (2, 2), cw
cw = dot_product(conjg(x), y)
```

Эти и все последующие функции БПЛА вернут 0, если $n \le 0$.

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 10
 complex(4) :: x(n) = (1, 1), y(n) = (2, 2), cw1, cw2
 cw1 = cdotu(n, x, 1, y, 1)
 cw2 = dot_product(conjg(x), y)
 print *, cw1, cw2          ! (0.000000E+00,40.000000)   (0.000000E+00,40.000000)
 cw1 = cdotc(n, x, 1, y, 1)
 cw2 = dot_product(x, y)
 print *, cw1, cw2          ! (40.000000,0.000000E+00)   (40.000000,0.000000E+00)
end program go
```

4.4.12. Скалярное произведение векторов с аккумулятором двойной точности

Скалярное произведение векторов x и y возвращается функциями

dw = DSDOT(*n*, *sx*, *incx*, *sy*, *incy*)

cw = CZDOTC(*n*, *cx*, *incx*, *cy*, *incy*); *cw* = CZDOTU(*n*, *cx*, *incx*, *cy*, *incy*)

zw = ZQDOTC(*n*, *zx*, *incx*, *zy*, *incy*); *zw* = ZQDOTU(*n*, *zx*, *incx*, *zy*, *incy*)

использующими при вычислениях аккумулятор двойной точности.

Функции DSDOT, CZDOTU и ZQDOTU вычисляют $\sum_{i=1}^{n} x_i y_i$. Точность, которую обеспечивают используемые в функциях аккумуляторы: в DSDOT – REAL(8), CZDOTU – COMPLEX(8) и ZQDOTU – COMPLEX(16).

Функции CZDOTC и ZQDOTC применяют соответственно аккумуляторы с точностью COMPLEX(8) и COMPLEX(16). Они вычисляют $\sum_{i=1}^{n} \bar{x}_i y_i$.

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 10
 complex(8) :: x(n) = (1, 1), y(n) = (2, 2), cw1, cw2
 cw1 = zqdotu(n, x, 1, y, 1)
 cw2 = zqdotc(n, x, 1, y, 1)
 print *, cw1                    ! (0.000000000000000E+000, 40.000000000000000)
 print *, cw2                    ! (40.000000000000000, 0.000000000000000E+000)
end program go
```

4.4.13. Сумма скаляра и скалярного произведения векторов с аккумулятором двойной точности

Сумма скаляра и скалярного произведения векторов *x* и *y* возвращается функциями

sw = SDSDOT(*n*, *sa*, *sx*, *incx*, *sy*, *incy*); *dw* = DQDDOT(*n*, *da*, *dx*, *incx*, *dy*, *incy*)
cw = CZCDOT(*n*, *ca*, *cx*, *incx*, *cy*, *incy*); *cw* = CZUDOT(*n*, *ca*, *cx*, *incx*, *cy*, *incy*)
zw = ZQCDOT(*n*, *za*, *zx*, *incx*, *zy*, *incy*); *zw* = ZQUDOT(*n*, *za*, *zx*, *incx*, *zy*, *incy*)

использующими при вычислениях аккумулятор двойной точности.

Функции SDSDOT, DQDDOT, CZUDOT и ZQUDOT вычисляют $\alpha + \sum_{i=1}^{n} x_i y_i$. Точность, которую обеспечивают используемые в функциях аккумуляторы: в SDSDOT – REAL(8), DQDDOT – REAL(16), CZUDOT – COMPLEX(8) и ZQUDOT – COMPLEX(16). Функции CZDOT и ZQDOT применяют соответственно аккумуляторы с точностью COMPLEX(8) и COMPLEX(16). Они вычисляют $\alpha + \sum_{i=1}^{n} \bar{x}_i y_i$.

4.4.14. Скалярное произведение векторов с аккумулятором двойной точности, передаваемым в качестве параметра

Сумма скаляра и скалярного произведения векторов *x* и *y* возвращается функциями

sw = SDDOT{I A}(*n*, *sb*, *dacc*, *sx*, *incx*, *sy*, *incy*) ! То есть SDDOTI или SDDOTA
cw = CZDOT{I A}(*n*, *cb*, *dacc*, *cx*, *incx*, *cy*, *incy*)
dw = DQDOT{I A}(*n*, *db*, *dacc*, *dx*, *incx*, *dy*, *incy*)
zw = ZQDOT{I A}(*n*, *zb*, *dzacc*, *zx*, *incx*, *zy*, *incy*)

использующими при вычислениях аккумулятор DACC двойной точности.

Параметр DACC является массивом типа REAL(8) размера 2; используется в качестве аккумулятора, обеспечивающего точность REAL(16). Параметр DZACC – массив типа REAL(8) размера 4; используется в качестве аккумулятора, обеспечивающего точность COMPLEX(16). Функции, имена которых завершаются буквой I, инициализируют DACC значением 0.

Все функции вычисляют DACC+ $\beta + \sum_{i=1}^{n} x_i y_i$ и размещают результат в DACC(1). Также результат возвращается как значение функции.

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 10
 real(4) :: b = 3.0, x(n) = 1.0, y(n) = 2.0, sw1, sw2
 real(8) :: dacc(2)
 sw1 = SDDOTI(n, b, dacc, x, 1, y, 1)
 dacc = 3.0_8
 sw2 = SDDOTA(n, b, dacc, x, 1, y, 1)
 print *, dot_product(x, y)              ! 20.000000
 print *, sw1                            ! 23.000000 .
 print *, sw2, dacc(1)                   ! 26.000000     26.000000000000000
end program go
```

4.4.15. Произведение Адамара

Произведение Адамара – это присваивание вида $z = x * y$. Оно может быть вычислено непосредственно или подпрограммой

CALL {G}HPROD(n, gx, $incx$, gy, $incy$, gz, $incz$) ! {G} – это S или D

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 5
 real(4) :: x(n) = 2.0, y(n) = 3.5, z(n)
 call shprod(n, x, 1, y, 1, z, 1)
 print '(1x, 10f5.2)', x * y                ! 7.00 7.00 7.00 7.00 7.00
 print '(1x, 10f5.2)', z                    ! 7.00 7.00 7.00 7.00 7.00
end program go
```

4.4.16. Сумма произведений трех векторов

Сумма произведений трех векторов $\sum_{i=1}^{n} x_i y_i z_i$ возвращается функциями

sw = SXYZ(n, sx, $incx$, sy, $incy$, sz, $incz$); dw = DXYZ(n, dx, $incx$, dy, $incy$, dz, $incz$)

Ее также вернет и вызов SUM($x * y * z$).

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 5
 real(4) :: x(n) = 2.0, y(n) = 3.5, z(n) = 4.0, sw
 sw = sxyz(n, x, 1, y, 1, z, 1)
 print *, sw                          ! 140.000000
 print *, sum(x * y * z)              ! 140.000000
end program go
```

4.4.17. Сумма элементов вектора

Сумма элементов вектора $\sum_{i=1}^{n} x_i$ возвращается встроенной функцией SUM и функциями IMSL

$iw = \mathrm{ISUM}(n, ix, incx)$; $sw = \mathrm{SSUM}(n, sx, incx)$; $dw = \mathrm{DSUM}(n, dx, incx)$

4.4.18. Сумма абсолютных значений элементов вектора

Возвращается и функциями

$sw = \mathrm{SASUM}(n, sx, incx)$; $dw = \mathrm{DASUM}(n, dx, incx)$

$sw = \mathrm{SCASUM}(n, cx, incx)$; $dw = \mathrm{DZASUM}(n, zx, incx)$

Функции SASUM и DASUM вычисляют $\sum_{i=1}^{n} |x_i|$. Функции SCASUM и DZASUM возвращают $\sum_{i=1}^{n} \left(|\mathrm{REAL}(x_i)| + |\mathrm{AIMAG}(x_i)|\right)$.

Предупреждение. Результаты функций SCASUM и DZASUM – вещественные величины.

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 5
 complex(4) :: cx(n) = (-2.0, 3.0)
 real(4) :: sw
 sw = scasum(n, cx, 1)
 print *, sw                          ! 25.000000
 ! Используем встроенные функции SUM, ABS, REAL и AIMAG
 sw = sum(abs(real(cx))) + sum(abs(aimag(cx)))
 print *, sw                          ! 25.000000
 sw = sum(sqrt(real(cx)*real(cx) + aimag(cx)*aimag(cx)))
 print *, sw                          ! 18.027760
 ! То же самое, но короче
 print *, sum(abs(cx))                ! 18.027760
end program go
```

4.4.19. Евклидова, или 2-норма вектора

Евклидова норма вектора $\sqrt{\sum_{i=1}^{n}|x_i|^2}$ возвращается функциями

sw = SNRM2(*n*, *sx*, *incx*); *dw* = DNRM2(*n*, *dx*, *incx*)
sw = SCNRM2(*n*, *cx*, *incx*); *dw* = DZNRM2(*n*, *zx*, *incx*)

В случае вещественных чисел евклидова норма равна $\sqrt{\sum_{i=1}^{n} x_i^2}$.

Предупреждение. Результаты функций SCNRM2 и DZNRM2, принимающих комплексный вектор, – вещественные величины.

4.4.20. Произведение элементов вектора

Произведение элементов вектора $\prod_{i=1}^{n} x_i$ возвращается функциями

sw = SPRDCT(*n*, *sx*, *incx*); *dw* = DPRDCT(*n*, *dx*, *incx*)

Также его вернет и встроенная функция PRODUCT.

4.4.21. Индекс минимального элемента вектора

Вернут функции

iw = IIMIN(*n*, *ix*, *incx*); *iw* = ISMIN(*n*, *sx*, *incx*); *iw* = IDMIN(*n*, *dx*, *incx*)

Также его вернет и встроенная функция MINLOC.

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 10
 real(4) :: sx(n) = 1.0
 sx(5) = 0.0; sx(7) = 0.0
 print *, ismin(n, sx, 1)          !   5
 print *, minloc(sx)               !   5
end program go
```

4.4.22. Индекс максимального элемента вектора

Вернут функции

iw = IIMAX(*n*, *ix*, *incx*); *iw* = ISMAX(*n*, *sx*, *incx*); *iw* = IDMAX(*n*, *dx*, *incx*)

Также его вернет и встроенная функция MAXLOC.

4.4.23. Индекс минимального по модулю элемента вектора

Вернут функции

iw = ISAMIN(*n*, *sx*, *incx*); *iw* = IDAMIN(*n*, *dx*, *incx*)
iw = ICAMIN(*n*, *cx*, *incx*); *iw* = IZAMIN(*n*, *zx*, *incx*)

Функции ISAMIN и IDAMIN возвращают MINLOC(ABS(x)). Функции ICAMIN и IZAMIN возвращают MINLOC(ABS(REAL(x)) + ABS(AIMAG(x))), что также равно MINLOC(ABS(x)).

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 10
 complex(4) :: x(n) = (-2.0, 3.0)
 x(5) = (1.0, 3.0)
 print *, icamin(n, x, 1)                    !  5
 ! То же, но с применением встроенных функций
 print *, minloc(abs(real(x))+abs(aimag(x)))  !  5
 print *, minloc(abs(x))                     !  5
end program go
```

4.4.24. Индекс максимального по модулю элемента вектора

Вернут функции

iw = ISAMAX(*n*, *sx*, *incx*); *iw* = IDAMAX(*n*, *dx*, *incx*)

iw = ICAMAX(*n*, *cx*, *incx*); *iw* = IZAMAX(*n*, *zx*, *incx*)

Функции ISAMAX и IDAMAX возвращают MAXLOC(ABS(x)). Функции ICAMAX и IZAMAX возвращают MAXLOC(ABS(REAL(x)) + ABS(AIMAG(x))), что также равно MAXLOC(ABS((x)).

4.4.25. Построение плоского вращения Гивенса

Элементы *sc* и *ss* матрицы плоского вращения (преобразования) Гивенса возвращает подпрограмма

CALL {G}ROTG(*ga*, *gb*, *gc*, *gs*) ! {G} – это S или D

По заданным величинам a и b подпрограмма вычисляет значения c и s по формулам

$$c=\begin{cases}\sigma(a/r), & r\neq 0,\\ 1, & r=0,\end{cases}\qquad s=\begin{cases}\sigma(b/r), & r\neq 0,\\ 1, & r=0,\end{cases}$$

где $r=\sqrt{sa^2+sb^2}$, а $\sigma=\begin{cases}sign(a), & |a|>|b|,\\ sign(b), & |a|\le|b|.\end{cases}$

Тогда значения c и s удовлетворяют равенству $\begin{pmatrix} c & s\\ -s & c\end{pmatrix}\begin{pmatrix} a\\ b\end{pmatrix}=\begin{pmatrix} r\\ 0\end{pmatrix}$.

Дополнительно подпрограмма возвращает значения r (замещает им параметр a) и z (на месте параметра b). Величина z вычисляется по формуле

$$z = \begin{cases} s, & |s| < c \text{ или } c = 0, \\ 1/c, & 0 < |c| \le s. \end{cases}$$

Значение переменной z используется при необходимости для восстановления значений c и s:

- если $z = 1$, то $c = 0$ и $s = 1$;
- если $|z| < 1$, то $sc = \sqrt{1 - z^2}$ и $s = z$;
- если $|z| > 1$, то $c = 1/z$ и $s = \sqrt{1 - c^2}$.

Замечания:

1. Преобразования Гивенса используются, в частности, при вычислении QR-разложения матрицы, которое, в свою очередь, может быть применено для решения систем линейных уравнений и поиска собственных значений.
2. В используемой при написании пособия версии библиотеки IMSL, поставленной с FPS 4.0, SROTG возвращает неверные значения, а DROTG работает корректно. В случае CVF верно работают и SROTG, и DROTG.

4.4.26. Применение плоского вращения Гивенса

По найденным ранее элементам матрицы плоского вращения

CALL {G}ROT(n, gx, $incx$, gy, $incy$, gc, gs) ! {G} – это S, D, CS или ZD

вычисляет $\begin{pmatrix} x_i \\ y_i \end{pmatrix} \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$.

Предупреждение. Для CSROT и ZDROT скаляры c и s – вещественные, а x и y – комплексные.

Пример. В цикле обнуляются последовательно все, кроме первого, элементы вектора x.

```
program giv
 use msimsl
 integer(4), parameter :: n = 4
 real(8) :: x(n) = (/ 4, -5, 6, -7 /), da, db, dc, ds
 do i = n, 2, -1
  da = x(i - 1); db = x(i)
  call drotg(da, db, dc, ds)                  ! Вычисляем dc и ds
  call drot(1, x(i-1:i-1), 1, x(i:i), 1, dc, ds)  ! Обнуляем элемент x(i)
 end do
 print '(1x, a, 10f7.3)', 'x: ', x            ! x: -11.225  .000  .000  .000
end program giv
```

4.4.27. Построение быстрого вращения Гивенса

Выполняется подпрограммой

CALL {G}ROTMG($gd1, gd2, gx1, gy1, gparam$) ! {G} – это S или D

sparam (*dparam*) – массив из пяти элементов.

Входные параметры $d1, d2, x1$ и $y1$ задают вектор $(w1, z1)^T$:

$$\begin{pmatrix} w_1 \\ z_1 \end{pmatrix} = \begin{pmatrix} \sqrt{d_1} & 0 \\ 0 & \sqrt{d_2} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}.$$

Подпрограмма вычисляет матрицу G быстрого вращения Гивенса и заменяет $d1$, $d2$ и $x1$ на $\tilde{d}_1$, $\tilde{d}\,2$ и $\tilde{x}_1$ – величины, представляющие быстрое преобразование Гивенса. Параметр $y1$, значение которого после преобразования равно нулю, не меняется.

Входное значение $d1$ не должно быть отрицательным; $d2$ может быть на входе меньше нуля.

Представление матрицы G хранится в массиве *sparam* или *dparam*. Информация о форме матрицы записывается в $param(1)$.

Если $param(1) = 1$, то $\left|d_1 x_1^2\right| \le \left|d_2 y_1^2\right|$ и $G = \begin{pmatrix} param(2) & 1 \\ 1 & param(5) \end{pmatrix}$; элементы $param(3)$ и $param(4)$ не меняются.

Если $param(1) = 0$, то $\left|d_1 x_1^2\right| > \left|d_2 y_1^2\right|$ и $G = \begin{pmatrix} 1 & param(4) \\ param(3) & 1 \end{pmatrix}$; элементы $param(2)$ и $param(5)$ не меняются.

Если $param(1) = -1$, то $G = \begin{pmatrix} param(2) & param(4) \\ param(3) & param(5) \end{pmatrix}$.

Если $param(1) = -2$, то $G = I$, где I – единичная матрица; элементы $param(2)$, $param(3)$, $param(4)$ и $param(5)$ не меняются.

Матрица G такова, что $G\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$.

4.4.28. Применение быстрого вращения Гивенса

Подпрограмма

CALL {G}ROTM($n, gx, incx, gy, incy, gparam$) ! {G} – это S, D, CS или ZD

по найденному ранее массиву *sparam* (*dparam*) выполняет быстрое плоское вращение Гивенса. Она вычисляет:

- $\begin{pmatrix} x_i \\ y_i \end{pmatrix} \leftarrow \begin{pmatrix} param(2) & 1 \\ 1 & param(5) \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $i = 1, \ldots, n$, если $param(1) = 1$;

- $\begin{pmatrix} x_i \\ y_i \end{pmatrix} \leftarrow \begin{pmatrix} 1 & param(4) \\ param(3) & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $i = 1, \ ..., n$, если $param(1) = 0$;
- $\begin{pmatrix} x_i \\ y_i \end{pmatrix} \leftarrow \begin{pmatrix} param(2) & param(4) \\ param(3) & param(5) \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$, $i = 1, \ ..., n$, если $param(1) = -1$.

Если $n \le 0$ или $param(1) = -2$, то вычислений не выполняется.

Предупреждение. Для CSROTM и ZDROTM массив PARAM вещественный, а x и y комплексные.

Пример. В цикле посредством быстрых вращений Гивенса обнуляются последовательно все, кроме первого, элементы вектора x.

```
program fgiv
 use msimsl
 integer(4), parameter :: n = 4
 real(8) :: x(n) = (/ 4, -5, 6, -7 /), dd1, dd2, dx1, dy1, dparam(5)
 do i = n, 2, -1
  dd1 = 1.0_8; dd2 = 2.0_8; dx1 = x(i - 1); dy1 = x(i)
  call drotmg(dd1, dd2, dx1, dy1, dparam)  ! Вычисляем dparam
  ! Обнуляем элемент x(i)
  call drotm(1, x(i - 1:i - 1), 1, x(i:i), 1, dparam)
 end do
 print '(1x, a, 10f7.3)', 'x: ', x                    ! x:  -11.613  .000  .000  .000
end program fgiv
```

4.5. БАЗОВЫЕ ПРОЦЕДУРЫ ЛИНЕЙНОЙ АЛГЕБРЫ УРОВНЕЙ 2 И 3

4.5.1. Замечания для программиста

Имя базовой процедуры состоит из префикса, корня и суффикса.

Префикс определяет тип используемых данных. Если в качестве префикса используется S, то тип данных REAL(4); если – D, то REAL(8) ; если – C, то COMPLEX(8); если – Z, то COMPLEX(16).

Корень имени указывает на вид применяемой матрицы:

- GE – матрица общего вида;
- GB – ленточная матрица общего вида;
- SY – симметрическая матрица;
- SB – ленточная симметрическая матрица;
- HE – эрмитова матрица;
- HB – ленточная эрмитова матрица;
- TR – треугольная матрица;
- TB – ленточная треугольная матрица.

Суффикс указывает на вид преобразований:

- MV – произведение матрицы и вектора;
- SV – вычисления с векторами;
- R – преобразования ранга 1;
- RU – преобразования ранга 1 с комплексно-несопряженными данными;
- RC – преобразования ранга 1 с комплексно-сопряженными данными;
- R2 – преобразования ранга 2;
- MM – произведение матриц;
- SM – произведение симметрических матриц;
- RK – преобразования ранга k;
- R2K – преобразования ранга $2k$.

Сохраняется введенный в разд. 4.4.1 синтаксис обобщенных имен процедур.

IMSL не поддерживает упакованные симметрические, эрмитовы и треугольные матрицы (корни процедур SP, HP и TP) и не выполняет в БПЛА уровня 2 вычислений с повышенной точностью.

Параметры процедур типа CHARACTER(1) передают приведенную в табл. 4.3 информацию.

Таблица 4.3. Смысл параметров процедур, имеющих тип CHARACTER(1)

Параметры	*Значения*	*Используемые матрицы*
trans, transa, transb	'N' или 'n'	Нетранспонированная
То же	'T' или 't'	Транспонированная
"	'C' или 'c'	Комплексно-сопряженная и транспонированная
uplo	'L' или 'l'	Нижняя треугольная
"	'U' или 'u'	Верхняя треугольная
diag	'N' или 'n'	Треугольная со своей диагональю
"	'U' или 'u'	Треугольная с единичной диагональю
side	'L' или 'l'	AB
"	'R' или 'r'	BA

Подпрограммы, выполняющие вычисления, включающие произведения by или bC, не требуют задания y или C, когда $b = 0$.

Подпрограммы БПЛА уровня 2 могут содержать объявления формальных параметров:

```
integer(4) incx, incy, ncoda, nlca, nuca, Lda, m, n
character(1) diag, trans, uplo
real(4) salpha, sbeta, sx(*), sy(*), sa(Lda, *)
real(8) dalpha, dbeta, dx(*), dy(*), da(Lda, *)
complex(4) calpha, cbeta, cx(*), cy(*), ca(Lda, *)
complex(8) zalpha, zbeta, zx(*), zy(*), za(Lda, *)
```

Смысл параметров *sx*, *sy* и *incx* и *incy* пояснен в разд. 4.4.1. Допустимы только ненулевые значения *incx* и *incy*. Величина ведущего измерения *Lda* должна быть больше или равна числу рядов матрицы, которая содержится в соответствующем массиве.

Подпрограммы БПЛА уровня 3 могут содержать объявления формальных параметров:

```
integer(4) k, Lda, Ldb, Ldc, m, n
character(1) diag, trans, transa, transb, side, uplo
real(4) salpha, sbeta, sa(Lda, *), sb(Ldb,*), sc(Ldc, *)
real(8) alpha, dbeta, da(Lda, *), db(Ldb, *), dc(Ldc, *)
complex(4) calpha, cbeta, ca(LDA, *), cb(Ldb, *), cc(Ldc, *)
complex(8) zalpha, zbeta, za(Lda, *), zb(Ldb, *), zc(Ldc, *)
```

Параметры *k*, *m*, *n* должны быть больше или равны нулю, в противном случае возникнет ошибка. Величина каждого из ведущих измерений *Lda*, *Ldb*, *Ldc* должна быть больше или равна числу рядов матрицы, которая содержится в соответствующем массиве.

Большинство преобразований 2-го уровня могут быть выполнены с применением встроенной функции MATMUL. Однако ее использование следует ограничить, поскольку выполняемые процедурами уровня 2 преобразования матриц, например умножение матрицы на вектор или низкоранговые модификации матриц, оптимизированы по скорости выполнения. Поэтому, как правило, процедуры уровня 2 работают существенно быстрее, чем дающий те же результаты код, содержащий MATMUL.

4.5.2. Список процедур уровней 2 и 3

Перечень процедур с указанием типа результата приведен в табл. 4.4.

Таблица 4.4. БПЛА уровней 2 и 3

№	*Действия*	*Процедуры*			
		Real(4)	Real(8)	Complex(4)	Complex(8)
1	Произведение матрицы общего вида на вектор	SGEMV	DGEMV	CGEMV	ZGEMV
2	Произведение ленточной матрицы на вектор	SGBMV	DGBMV	CGBMV	ZGBMV
3	Произведение эрмитовой матрицы на вектор			CHEMV	ZHEMV
4	Произведение ленточной эрмитовой матрицы на вектор			CHBMV	ZHBMV
5	Произведение вещественной симметрической матрицы на вектор	SSYMV	DSYMV		

6	Произведение ленточной симметрической матрицы на вектор	SSBMV	DSBMV		
7	Произведение треугольной матрицы на вектор	STRMV	DTRMV	CTRMV	ZTRMV
8	Произведение ленточной треугольной матрицы на вектор	STBMV	DTBMV	CTBMV	ZTBMV
9	Вычисление вектора с применением треугольной матрицы	STRSV	DTRSV	CTRSV	ZTRSV
10	Вычисление вектора с применением ленточной треугольной матрицы	STBSV	DTBSV	CTBSV	ZTBSV
11	Преобразования ранга 1 матрицы общего вида	SGER	DGER	CGERU CGERC	ZGERU ZGERC
12	Преобразования ранга 1, 2, k и $2k$ эрмитовой матрицы			CHER CHER2 CHERK CHER2K	ZHER ZHER2 ZHERK ZHER2K
13	Преобразования ранга 1, 2, k и $2k$ симметрической матрицы	SSYR SSYR2 SSYRK SSYR2K	DSYR DSYR2 DSYRK DSYR2K	CSYRK CSYR2K	ZSYRK ZSYR2K
14	Произведение матриц общего вида	SGEMM	DGEMM	CGEMM	ZGEMM
15	Произведение с симметрической матрицей	SSYMM	DSYMM	CSYMM	ZSYMM
16	Произведение с эрмитовой матрицей			CHEMM	ZHEMM
17	Произведение с треугольной матрицей	STRMM	DTRMM	CTRMM	ZTRMM
18	Вычисления с треугольными матрицами	STRSM	DTRSM	CTRSM	ZTRSM

4.5.3. Умножение матрицы общего вида на вектор

Умножение $m \times n$-матрицы A на вектор выполняет подпрограмма

CALL {G}GEMV(*trans*, *m*, *n*, *galpha*, *ga*, *Lda*, *gx*, *incx*, *gbeta*, *gy*, *incy*)

где {G} – это S, D, C или Z. Для всех типов данных подпрограмма вычисляет $y \leftarrow \alpha Ax + \beta y$ (*trans* = 'N') или $y \leftarrow \alpha A^T x + \beta y$ (*trans* = 'T') и для комплексных данных, если *trans* = 'C', – $y \leftarrow \alpha \overline{A}^T + \beta y$.

То же возвращает выражение с MATMUL:

```
sy = matmul(salpha * sa, sx) + sbeta * sy
```

или для комплексных данных, когда *trans* = 'C':

```
cy = matmul(calpha * transpose(conjg(ca)), cx) + cbeta * cy
```

Замечание. Если *trans* = 'N', то размер *y* равен *m*, иначе он равен *n*.

Пример:

```
program go
 use msimsl, nouse => cbeta
 integer(4), parameter :: m = 2, n = 3, Lda = m
 real(4) :: sa(m, n) = 2, sx(n) = (/ 3, 4, 5 /), sy(m) = 1
 real(4) :: salpha = 0.5, sbeta = 0.3
 complex(4) :: ca(m, n) = (2, 2), cx(m) = (/ (3, 3), (4, 4) /), cy(n) = (1, 1)
 complex(4) :: calpha = (0.5, 0.5), cbeta = (0.3, 0.3)
 ! Вычисляем y ← αAx + βy (trans = 'N')
 call sgemv('N', m, n, salpha, sa, Lda, sx, 1, sbeta, sy, 1)
 ! Вычисляем y ← αĀ^T + βy (trans = 'C')
 call cgemv('C', m, n, calpha, ca, Lda, cx, 1, cbeta, cy, 1)
 call wrrrn('sy', 1, m, sy, 1, 0)              ! Вывод результата
 call wrcrn('cy', 1, n, cy, 1, 0)
 sy = 1                                        ! То же, но со встроенными функциями
 sy = matmul(salpha * sa, sx) + sbeta * sy
 cy = (1, 1)
 cy = matmul(calpha * transpose(conjg(ca)), cx) + cbeta * cy
 call wrrrn('sy', 1, m, sy, 1, 0)              ! Вывод результата
 call wrcrn('cy', 1, n, cy, 1, 0)
end program go
```

Результат:

```
          sy
   12.30          12.30

                          cy
( 14.00, 14.60)    ( 14.00, 14.60)    ( 14.00, 14.60)
```

4.5.4. Умножение ленточной матрицы на вектор

Умножение ленточной $m \times n$-матрицы A с *nlca* нижними кодиагоналями и *nuca* верхними кодиагоналями на вектор выполняет подпрограмма

CALL {G}GBMV(*trans*, *m*, *n*, *nlca*, *nuca galpha*, *ga*, *Lda*, *gx*, *incx*, *gbeta*, *gy*, *incy*)

где {G} – это S, D, C или Z. Матрица хранится в ленточном виде (см. разд. 4.3.2). Выполняет те же действия, что и подпрограмма из разд. 4.5.3.

4.5.5. Умножение эрмитовой матрицы на вектор

Умножение эрмитовой $n \times n$-матрицы A на вектор выполняет подпрограмма

CALL {G}HEMV(*uplo*, *n*, *galpha*, *ga*, *Lda*, *gx*, *incx*, *gbeta*, *gy*, *incy*)

где {G} – это C или Z. Подпрограмма вычисляет $y \leftarrow \alpha Ax + \beta y$. В зависимости от значения параметра *uplo* используется либо верхняя (*uplo* = 'U'), либо нижняя (*uplo* = 'L') треугольная часть матрицы A.

4.5.6. Умножение эрмитовой ленточной матрицы на вектор

Умножение эрмитовой ленточной $n \times n$-матрицы A с *ncoda* кодиагоналями на вектор выполняет подпрограмма

CALL {G}HBMV(*uplo*, *n*, *ncoda*, *galpha*, *ga*, *Lda*, *gx*, *incx*, *gbeta*, *gy*, *incy*)

где {G} – это C или Z. Подпрограмма вычисляет $y \leftarrow \alpha Ax + \beta y$. Эрмитова матрица хранится в ленточном виде (разд. 4.3.4).

4.5.7. Умножение вещественной симметрической матрицы на вектор

Умножение вещественной симметрической $n \times n$-матрицы A на вектор выполняет подпрограмма

CALL {G}SYMV(*uplo*, *n*, *galpha*, *ga*, *Lda*, *gx*, *incx*, *gbeta*, *gy*, *incy*)

где {G} – это S или D. Подпрограмма вычисляет $y \leftarrow \alpha Ax + \beta y$.

4.5.8. Умножение симметрической ленточной матрицы на вектор

Умножение симметрической ленточной $n \times n$-матрицы A с *ncoda* кодиагоналями на вектор выполняет подпрограмма

CALL {G}SBMV(*uplo*, *n*, *ncoda*, *galpha*, *ga*, *Lda*, *gx*, *incx*, *gbeta*, *gy*, *incy*)

где {G} – это S или D. Подпрограмма вычисляет $y \leftarrow \alpha Ax + \beta y$. Симметрическая матрица хранится в ленточном виде (разд. 4.3.3).

4.5.9. Умножение треугольной матрицы на вектор

Умножение треугольной $n \times n$-матрицы A на вектор выполняет подпрограмма

CALL {G}TRMV(*uplo*, *trans*, *diag*, *n*, *ga*, *Lda*, *gx*, *incx*)

где {G} – это S, D, C или Z. Для всех типов данных подпрограмма вычисляет $x \leftarrow Ax$ (*trans* = 'N') или $x \leftarrow A^T x$ (*trans* = 'T'). Для комплексных данных, когда *trans* = 'C', вычисляется $x \leftarrow \overline{A}^T x$. В зависимости от значения параметра *uplo* используется либо верхняя (*uplo* = 'U'), либо нижняя (*uplo* = 'L') треугольная часть матрицы A. Параметр указывает, нужно ли использовать диагональ матрицы в том виде, в каком она есть (*diag* = 'N'), или следует все диагональные элементы принять равными единице (*diag* = 'U').

4.5.10. Умножение треугольной ленточной матрицы на вектор

Умножение треугольной ленточной $n \times n$-матрицы A с *ncoda* кодиагоналями на вектор выполняет подпрограмма

CALL {G}TBMV(*uplo*, *trans*, *diag*, *n*, *ncoda*, *ga*, *Lda*, *gx*, *incx*)

где {G} – это S, D, C или Z. Треугольная матрица хранится в ленточном виде (разд. 4.3.5). Выполняет те же действия, что и подпрограмма из предыдущего раздела.

4.5.11. Умножение обратной треугольной матрицы на вектор

При заданной треугольной $n \times n$-матрице A подпрограмма

CALL {G}TRSV(*uplo*, *trans*, *diag*, *n*, *ga*, *Lda*, *gx*, *incx*)

где {G} – это S, D, C или Z, для всех типов данных вычисляет $x \leftarrow A^{-1}x$ или $x \leftarrow A^{-T}x$. Для комплексных данных, когда *trans* = 'C', вычисляется $x \leftarrow \overline{A}^{-T}x$.

4.5.12. Умножение обратной треугольной ленточной матрицы на вектор

При заданной ленточной треугольной $n \times n$-матрице A с *ncoda* кодиагоналями подпрограмма

CALL {G}TBSV(*uplo*, *trans*, *diag*, *n*, *ncoda*, *ga*, *Lda*, *gx*, *incx*)

где {G} – это S, D, C или Z, производит те же вычисления, что и подпрограмма предыдущего раздела. Треугольная матрица хранится в ленточном виде (разд. 4.3.5).

4.5.13. Преобразования ранга 1 матрицы общего вида

При заданной вещественной или комплексной общего вида $m \times n$-матрице A подпрограмма

CALL {G}GER(*m*, *n*, *galpha*, *gx*, *incx*, *gy*, *incy*, *ga*, *Lda*)

где {G} – это S, D, C или Z, вычисляет $A \leftarrow A + \alpha xy^T$.

Подпрограмма

CALL {G}GERC(*m*, *n*, *galpha*, *gx*, *incx*, *gy*, *incy*, *ga*, *Lda*)

где {G} – это C или Z, вычисляет $A \leftarrow A + \alpha x\overline{y}^T$.

<u>*Замечание*</u>. Матрица B имеет ранг не выше единицы, если ее можно представить в виде

$$B = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_m \end{pmatrix} \begin{pmatrix} y_1 & y_2 \dots y_m \end{pmatrix} = xy^T \in \mathbf{R}^{m \times n}.$$

Чтобы вычислить такое произведение с помощью встроенной функции MATMUL, необходимо преобразовать, применив встроенную функцию RESHAPE, векторы x и y, имеющие соответственно размеры m и n, в двумерные массивы форм $(m, 1)$ и $(1, n)$.

Пример:

```
program go
 use msimsl
 integer(4), parameter :: m = 2, n = 3, Lda = m
 real(4) :: sa(m, n) = 2, sx(m) = (/ 3, 4 /), sy(n) = (/ 5, 6, 7 /)
 real(4) :: salpha = 1.0
 call sger(m, n, salpha, sx, 1, sy, 1, sa, Lda)
 call wrrrn('a', m, n, sa, Lda, 0)              ! Вывод результата
 ! Те же преобразования с применением встроенных функций MATMUL и RESHAPE
 sa = 2
 sa = sa + matmul(reshape(sx, shape = (/ m, 1 /)), reshape(sy, shape = (/ 1, n /)))
 call wrrrn('sa', m, n, sa, Lda, 0)             ! Вывод результата
end program go
```

Результат:

```
                    sa
              1          2          3
   1      17.00      20.00      23.00
   2      22.00      26.00      30.00
```

4.5.14. Преобразования ранга 1, 2, k и $2k$ симметрической матрицы

Преобразования ранга 1 выполняет подпрограмма

CALL {G}SYR(*uplo*, *n*, *galpha*, *gx*, *incx*, *ga*, *Lda*)

где {G} – это S или D. Она вычисляет $A \leftarrow A + \alpha xx^T$, где A – симметрическая $n \times n$-матрица. Используется в зависимости от значения параметра *uplo* либо верхняя, либо нижняя треугольная часть матрицы.

Преобразования ранга 2 выполняет подпрограмма

CALL {G}SYR2(*uplo*, *n*, *galpha*, *gx*, *incx*, *gy*, *incy*, *ga*, *Lda*)

где {G} – это S или D. Они возвращают симметрическую $n \times n$-матрицу $A \leftarrow A + \alpha xy^T + \alpha yx^T$. Используется в зависимости от значения параметра *uplo* либо верхняя, либо нижняя треугольная часть матрицы.

Преобразования ранга k $(1 \le k \le n)$ выполняет подпрограмма

CALL {G}SYRK(*uplo*, *trans*, *n*, *k*, *galpha*, *ga*, *Lda*, *gbeta*, *gc*, *Ldc*)

где {G} – это S, D, C или Z. Их результатом является симметрическая $n \times n$-матрица $C \leftarrow \alpha A_{1:n,1:k} A^T_{1:n,1:k} + \beta C$ или $C \leftarrow \alpha A^T_{1:k,1:n} A_{1:k,1:n} + \beta C$, здесь и далее $A_{1:n,1:k}$ и $A_{1:k,1:n}$ – подматрицы, состоящие соответственно из k столбцов и k

строк $n \times n$-матрицы A. В зависимости от значения параметра *uplo* используется либо верхняя (*uplo* = 'U'), либо нижняя (*uplo* = 'L') треугольная часть матрицы C. В подпрограммах CSYRK и ZSYRK для параметра *trans* допустимы значения 'N' и 'T'; значение 'C' употребляться не должно.

Замечание. Тот же результат, но с большими временными затратами, если *trans* = 'N', вернет выражение со встроенной функцией MATMUL:

```
c = salpha * matmul(a(:, 1:k), transpose(a(:, 1:k))) + sbeta * c
```

Если же *trans* = 'T', то k-ранговые модификации матрицы C выполнит оператор

```
c = salpha * matmul(transpose(a(1:k, :)), a(1:k, :)) + sbeta * c
```

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 3
 integer(4) :: j, k
 real(4) :: sa(n, n), sc(n, n), c(n, n)
 real(4) :: salpha = 2.0, sbeta = 0.5
 sc = reshape((/      3.0,    1.0,    4.0,                &
                      1.0,    2.0,   -5.0,                &
                      4.0,   -5.0,   -2.0 /), shape = (/ n, n /))
 do k = 1, n
  sa = sc; sa(1, n) = 0.5; sa(2, n) = 1.0        ! Задаем массив sa
  ! Используем выражение с MATMUL и вычислим массив c
  c = salpha * matmul(sa(:, 1:k), transpose(sa(:, 1:k))) + sbeta * sc
  ! Получим массив sc, применив SSYRK; в результате верхняя треугольная часть sc
  ! будет равна верхней треугольной части c
  call ssyrk('U', 'N', n, k, salpha, sa, n, sbeta, sc, n)
  call wrrrn('c', n, n, c, n, 1)                 ! Вывод результата
  call wrrrn('sc', n, n, sc, n, 1)
  do j = 1, n –1                                 ! Чтобы получить нижнюю треугольную часть sc,
    sc(j + 1:, j) = sc(j, j + 1:)                ! выполним цикл
  end do
 end do
end program go
```

Результат при $k = 2$:

```
                 sc
              1          2          3
   1      854.8      295.8     1098.5
   2                 104.0      373.8
   3                            1428.0
```

Преобразования ранга $2k$ ($1 \le k \le n$) выполняет подпрограмма

CALL {G}SYR2K(*uplo*, *trans*, *n*, *k*, *galpha*, *ga*, *Lda*, *gb*, *Ldb*, *gbeta*, *gc*, *Ldc*)
где {G} – это S, D, C или Z. Они вернут

$$C \leftarrow \alpha A_{1:n,\,1:k} B^T_{1:n,\,1:k} + \alpha B_{1:n,\,1:k} A^T_{1:n,\,1:k} + \beta C \text{ или } C \leftarrow \alpha A^T_{1:k,\,1:n} B_{1:k,\,1:n} + \alpha B^T_{1:k,\,1:n} A_{1:k,\,1:n} + \beta C,$$

где C – симметрическая $n \times n$-матрица. В подпрограммах CSYR2K и ZSYR2K параметр *trans* должен принимать значение 'N' или 'T'; значение 'C' недопустимо.

Замечание. Тот же результат, но с большими временными затратами, если *trans* = 'N', вернет выражение со встроенной функцией MATMUL:

```
c = salpha * matmul(a(:, 1:k), transpose(b(:, 1:k))) + &
    salpha * matmul(b(:, 1:k), transpose(a(:, 1:k))) + sbeta * c
```

Если же *trans* = 'T', то $2k$-ранговые модификации матрицы C выполнит оператор

```
c = salpha * matmul(transpose(a(1:k, :)), b(1:k, :)) + &
    salpha * matmul(transpose(b(1:k, :)), a(1:k, :)) + sbeta * c
```

4.5.15. Преобразования ранга 1, 2, *k* и 2*k* эрмитовой матрицы

Преобразования ранга 1 выполняет подпрограмма
CALL {G}HER(*uplo*, *n*, *galpha*, *gx*, *incx*, *ga*, *Lda*)

где {G} – это C или Z. Они возвращают $A \leftarrow A + \alpha x \bar{x}^T$, где A – эрмитова $n \times n$-матрица.

Предупреждение. Скалярные параметры подпрограмм имеют вещественный тип, а матрица и вектор – комплексный.

Преобразования ранга 2 выполняет подпрограмма
CALL {G}HER2(*uplo*, *n*, *galpha*, *gx*, *incx*, *gy*, *incy*, *ga*, *Lda*)

где {G} – это C или Z. Она вычисляет $A \leftarrow A + \alpha x \bar{y}^T + \bar{\alpha} y \bar{x}^T$, где A – эрмитова $n \times n$-матрица.

Пример:

```
program go
 use msimsl
 integer(4), parameter :: n = 3, Lda = n
 complex(4) :: calpha = (2.0, 1.0), cx(n) = (3.0, 4.0), cy(n) = (5.0, 6.0), ca(n, n)
 ca = reshape((/     (3.0, 0.0),    (1.0, 1.0),    (4.0, 0.0),                &
                     (1.0, 1.0),    (2.0, 0.0),   (-5.0, 1.0),                &
                     (4.0, 0.0),   (-5.0, -1.0),   (-2.0, 0.0) /), shape = (/ n, n /))
 call cher2('U', n, calpha, cx, 1, cy, 1, ca, Lda)
 call wrcrn('ca', n, n, ca, Lda, 1)              ! Вывод результата
end program go
```

Результат:

```
                    ca
                 1                2                3
1     ( 155.0,  0.0)   ( 153.0,  1.0)   ( 156.0,  0.0)
2                      ( 154.0,  0.0)   ( 147.0, -1.0)
3                                       ( 150.0,  0.0)
```

Преобразования ранга k ($1 \le k \le n$) выполняет подпрограмма

CALL {G}HERK(*uplo*, *trans*, *n*, *k*, *galpha*, *ga*, *Lda*, *gbeta*, *gc*, *Ldc*)

где {G} – это C или Z. Их результатом является эрмитова $n \times n$-матрица $C \leftarrow \alpha A_{1:n,1:k} \overline{A}^T_{1:n,1:k} + \beta C$ или $C \leftarrow \alpha \overline{A}^T_{1:k,1:n} A_{1:k,1:n} + \beta C$. В зависимости от значения параметра *uplo* используется либо верхняя (*uplo* = 'U'), либо нижняя (*uplo* = 'L') треугольная часть матрицы C. Для параметра *trans* допустимы значения 'N' и 'C'; значение 'T' употребляться не должно.

Преобразования ранга $2k$ ($1 \le k \le n$) выполняет подпрограмма

CALL {G}HER2K(*uplo*, *trans*, *n*, *k*, *galpha*, *ga*, *Lda*, *gb*, *Ldb*, *gbeta*, *gc*, *Ldc*)

где {G} – это C или Z. Они вернут $C \leftarrow \alpha A_{1:n,1:k} \overline{B}^T_{1:n,1:k} + \overline{\alpha} B_{1:n,1:k} \overline{A}^T_{1:n,1:k} + \beta C$ или $C \leftarrow \alpha \overline{A}^T_{1:k,1:n} B_{1:k,1:n} + \overline{\alpha} \overline{B}^T_{1:k,1:n} A_{1:k,1:n} + \beta C$, где C – эрмитова $n \times n$-матрица. Для параметра *trans* допустимы значения 'N' и 'C'.

<u>*Предупреждение*</u>. Параметр b является вещественным, матрицы – комплексными.

4.5.16. Произведение матриц общего вида

Подпрограмма

CALL {G}GEMM(*transa*, *transb*, *m*, *n*, *k*, *galpha*, *ga*, *Lda*, *gb*, *Ldb*, *gbeta*, *gc*, *Ldc*)

где {G} – это S, D, C или Z, для всех типов данных вычисляет $C \leftarrow \alpha AB + \beta C$, $C \leftarrow \alpha A^T B + \beta C$, $C \leftarrow \alpha AB^T + \beta C$ или $C \leftarrow \alpha A^T B^T + \beta C$ и для комплексных данных – $C \leftarrow \alpha A \overline{B}^T + \beta C$, $C \leftarrow \alpha \overline{A}^T B + \beta C$, $C \leftarrow \alpha A^T \overline{B}^T + \beta C$, $C \leftarrow \alpha \overline{A}^T B^T + \beta C$ или $C \leftarrow \alpha \overline{A}^T \overline{B}^T + \beta C$, где C – $m \times k$-матрица. Результат произведения матриц A и B должен быть согласован по форме с C. Так, если *transa* = *transb* = 'N' и A имеет форму (m, n), а B – (n, k), то произведение AB будет иметь форму (m, k).

Пример:

```
program go
 use msimsl
 integer(4), parameter :: m = 2, n = 3, k = 3, Lda = m, Ldb = n, Ldc = m
 real(4) :: sa(m, n) = 2, sb(n, n) = 3, sc(m, k) = 4
 real(4) :: salpha = 2.1, sbeta = 3.4
 ! Вычисляем C ← αAB + βC (transa = transb = 'N')
```

```
 call sgemm('N', 'N', m, n, k, salpha, sa, Lda, sb, Ldb, sbeta, sc, Ldc)
 call wrrrn('sc', m, n, sc, Ldc, 0)          ! Вывод результата
 sc = 4                                      ! Те же преобразования с применением
 sc = salpha*matmul(sa, sb) + sbeta*sc       ! встроенной функции MATMUL
 call wrrrn('sc', m, n, sc, Ldc, 0)          ! Вывод результата
end program go
```

Результат:

	sc		
	1	2	3
1	51.40	51.40	51.40
2	51.40	51.40	51.40

4.5.17. Произведение с симметрической матрицей

Подпрограмма

CALL {G}SYMM(*side*, *uplo*, *m*, *n*, *galpha*, *ga*, *Lda*, *gb*, *Ldb*, *gbeta*, *gc*, *Ldc*)

где {G} – это S, D, C или Z, вычисляет $C \leftarrow \alpha AB + \beta C$, если *side* = 'L', или $C \leftarrow \alpha BA + \beta C$, если *side* = 'R', где A – симметрическая $n \times n$-матрица. В зависимости от значения параметра *uplo* используется либо верхняя (*uplo* = 'U'), либо нижняя (*uplo* = 'L') треугольная часть матрицы A.

4.5.18. Произведение с эрмитовой матрицей

Подпрограмма

CALL {G}HEMM(*side*, *uplo*, *m*, *n*, *galpha*, *ga*, *Lda*, *gb*, *Ldb*, *gbeta*, *gc*, *Ldc*)

где {G} – это C или Z, вычисляет то же, что и подпрограмма из разд. 4.5.17, но теперь в вычисляемом выражении A – эрмитова $n \times n$-матрица.

4.5.19. Произведение с треугольной матрицей

Подпрограмма

CALL {G}TRMM(*side*, *uplo*, *transa*, *diag*, *m*, *n*, *galpha*, *ga*, *Lda*, *gb*, *Ldb*)

где {G} – это S, D, C или Z, для всех типов данных возвращает $B \leftarrow \alpha AB$, $B \leftarrow \alpha A^T B$, $B \leftarrow \alpha BA$ или $B \leftarrow \alpha BA^T$; для комплексных данных – $B \leftarrow \alpha \overline{A}^T B$ или $B \leftarrow \alpha B \overline{A}^T$, где B – $m \times n$-матрица, A – треугольная $n \times n$-матрица. Параметры *side*, *uplo*, *transa* и *diag* задают вид треугольной матрицы и выполняемые преобразования (разд. 4.5.1).

Пример:

```
program go
 use msimsl
 integer(4), parameter :: m = 2, n = 3, Lda = n, Ldb = m
 real(4) :: sa(n, n), sb(m, n) = 3.0
 real(4) :: salpha = 2.1
```

```
sa = reshape((/  1.0, 2.0, 3.0,                    &
                 0.0, 5.0, 6.0,                    &
                 0.0, 0.0, 9.0 /),   shape = (/ 3, 3 /), order = (/ 2, 1 /))
! Вычисляем B ← αBA (side = 'R') с единичной диагональю (diag = 'U')
call strmm('R', 'U', 'N', 'U', m, n, salpha, sa, Lda, sb, Ldb)
call wrrrn('sb', m, n, sb, Ldb, 0)              ! Вывод результата
! Произведение с применением встроенной функции MATMUL даст тот же
! результат, но с большими временными затратами
sb = 3.0; sa(1, 1) = 1.0; sa(2, 2) = 1.0; sa(3, 3) = 1.0
sb = salpha * matmul(sb, sa)
call wrrrn('sb', m, n, sb, Ldb, 0)              ! Вывод результата
end program go
```

Результат:

	sb		
	1	2	3
1	6.30	18.90	63.00
2	6.30	18.90	63.00

4.5.20. Вычисления с треугольной матрицей

Подпрограмма

CALL {G}TRSM(*side*, *uplo*, *transa*, *diag*, *m*, *n*, *galpha*, *ga*, *Lda*, *gb*, *Ldb*)

где {G} – это S, D, C или Z, для всех типов данных возвращает $B \leftarrow \alpha A^{-1}B$, $B \leftarrow \alpha BA^{-1}$, $B \leftarrow \alpha A^{-T}B$ или $B \leftarrow \alpha BA^{-T}$ и для комплексных данных – $B \leftarrow \alpha \overline{A}^{-T}B$ или $B \leftarrow \alpha B\overline{A}^{-T}$, где B – $m \times n$-матрица, A – $n \times n$-треугольная матрица. Параметры *side*, *uplo*, *transa* и *diag* задают вид треугольной матрицы и выполняемые преобразования (разд. 4.5.1).

4.6. ДОПОЛНИТЕЛЬНЫЕ ОПЕРАЦИИ НАД ВЕКТОРАМИ И МАТРИЦАМИ

4.6.1. Процедуры, которые могут быть заменены встроенными возможностями Фортрана

К таким процедурам относятся:

- копирование вещественных и комплексных общего вида и ленточных матриц (подпрограммы CRGRG, CCGCG, CRBRB и CCBCB); заменяются присваиванием согласованных массивов: $b = a$;
- преобразование вещественной матрицы в комплексную (подпрограммы CRGCG, CRRCR и CRBCB); заменяются присваиванием b = CMPLX(a), если b – массив типа COMPLEX(4), и b = CMPLX(a, 8) или DCMPLX(a), если b – имеет тип COMPLEX(8);

- транспонирование прямоугольной матрицы (TRNRR); заменяется встроенной функцией TRANSPOSE;
- произведение A^TA, A^TB, AB^T и AB общего вида матриц A и B (подпрограммы MXTXF, MXTYF, MXYTF, MRRRR и MCRCR); практически всегда заменяются встроенными функциями MATMUL и TRANSPOSE, например AB^T вернет вызов MATMUL(a, TRANSPOSE(b));
- произведение Адамара $c_{ij} = a_{ij}b_{ij}$ (HRRRR); заменяется присваиванием $c = a * b$;
- произведение A^Tx (подпрограммы MURRV и MUCRV); заменяются функцией MATMUL.

4.6.2. Преобразование матриц

Все рассматриваемые в разделе подпрограммы, выполняющие преобразование матриц, приведены в табл. 4.5. В табл. 4.6 описаны используемые подпрограммами параметры, а в табл. 4.7 – ошибки, которые могут возникать в подпрограммах CHFCG и CHBCB.

Все преобразования могут быть выполнены с двойной точностью. Для этого имя подпрограммы должно быть снабжено префиксом D, например если CRGRB преобразовывает массивы типа REAL(4), то DCRGRB – массивы типа REAL(8).

Таблица 4.5. Подпрограммы, преобразовывающие матрицы

Подпрограмма	*Выполняемые преобразования*
CALL CRGRB(n, a, Lda, nlc, nuc, b, Ldb) *Входные параметры*: n, a, Lda, nlc, nuc, Ldb. *Выходной параметр*: b. Тип a и b – REAL(4) (или REAL(8) в случае DCRGRB); тип остальных параметров – INTEGER(4)	Вещественный массив a формы (Ldb, n), содержащий общее представление $n\times n$-матрицы A, копируется в вещественный массив b формы ($nuc + 1 + nlc$, n), содержащий ленточное представление матрицы A (разд. 4.3.2)
CALL CRBRG(n, a, Lda, nlc, nuc, b, Ldb) Описание параметров такое же, как и у CRGRB	Вещественный массив a формы ($nuc + 1 + nlc$, n), содержащий ленточное представление $n\times n$-матрицы A, копируется в вещественный массив b формы (Ldb, n), содержащий общее представление матрицы A
CALL CCGCB(n, a, Lda, nlc, nuc, b, Ldb) *Входные параметры*: n, a, Lda, nlc, nuc, Ldb. *Выходной параметр*: b. Тип a и b – COMPLEX(4) (или COMPLEX(8) в случае DCCGCB); тип остальных параметров – INTEGER(4)	Комплексный массив a формы (Ldb, n), содержащий общее представление $n\times n$-матрицы A, копируется в комплексный массив b формы ($nlc + nuc + 1$, n), содержащий ленточное представление матрицы A

CALL CCBCG(*n*, *a*, *Lda*, *nlc*, *nuc*, *b*, *Ldb*) Описание параметров такое же, как и у CCGCB	Комплексный массив *a* формы (*nlc* + + *nuc* + 1, *n*), содержащий ленточное представление *n*×*n*-матрицы *A*, копируется в комплексный массив *b* формы (*Ldb*, *n*), содержащий общее представление матрицы *A*
CALL CSFRG(*n*, *a*, *Lda*) *Входные параметры*: *n*, *a*, *Lda*. *Выходной параметр*: *a*. Тип *a* – REAL(4) (или REAL(8) в случае DCSFRG); тип *n* и *Lda* – INTEGER(4)	Вещественный массив *a*, содержащий верхнюю треугольную часть симметрической *n*×*n*-матрицы *A*, преобразовывается в массив, содержащий общее представление симметрической матрицы *A*
CALL CHFCG(*n*, *a*, *Lda*) *Входные параметры*: *n*, *a*, *Lda*. *Выходной параметр*: *a*. Тип *a* – COMPLEX(4) (или COMPLEX(8) в случае DCHFCG); тип *n* и *Lda* – INTEGER(4)	Комплексный массив *a*, содержащий верхнюю треугольную часть эрмитовой *n*×*n*-матрицы *A*, преобразовывается в массив, содержащий общее представление эрмитовой матрицы *A*
CALL CSBRB(*n*,*a*,*Lda*,*nuca*,*b*,*Ldb*,*nlcb*,*nucb*) *Входные параметры*: *n*, *a*, *Lda*, *nuca*, *Ldb*, *nlcb*, *nucb*. *Выходной параметр*: *b*. Тип *a* и *b* – REAL(4) (или REAL(8) в случае DCSBRB); тип остальных параметров – INTEGER(4)	Вещественный массив *a*, содержащий ленточное представление верхней треугольной части симметрической *n*×*n*-матрицы *A*, копируется в массив *b*, содержащий ленточное представление всей матрицы *A*
CALL CHBCB(*n*,*a*,*Lda*,*nuca*,*b*,*Ldb*,*nlcb*,*nucb*) *Входные параметры*: *n*, *a*, *Lda*, *nuca*, *Ldb*, *nlcb*, *nucb*. *Выходной параметр*: *b*. Тип *a* и *b* – COMPLEX(4) (или COMPLEX(8) в случае DCHBCB); тип остальных параметров – INTEGER(4)	Комплексный массив *a*, содержащий ленточное представление верхней треугольной части эрмитовой *n*×*n*-матрицы *A*, копируется в массив *b*, содержащий ленточное представление всей матрицы *A*

Таблица 4.6. Параметры подпрограмм из табл. 4.5

Имя параметра	*Смысл*
n	Порядок матриц *A* и *B*
a (*b*)	Двумерный массив, представляющий матрицу *A* (*B*)
Lda (*Ldb*)	Ведущий размер массива *a* (*b*)
nlc (*nuc*) или *nlcb* (*nucb*)	Число нижних (верхних) кодиагоналей в матрице *B*
nuca	Число верхних кодиагоналей в матрице *A*

Таблица 4.7. Ошибки, возникающие в подпрограммах CHFCG и CHBCB

Тип	*Код*	*Причина ошибки*
3	1	Входная матрица не является эрмитовой, так как ее диагональный элемент имеет маленькую мнимую часть
4	2	Входная матрица не является эрмитовой, так как ее диагональный элемент имеет значительную мнимую часть

В случае ошибки с кодом 1 выдается предупреждение, мнимая часть диагонального элемента обнуляется и вычисления продолжаются. Ошибка с кодом 2 приводит к останову программы.

Пример для CRGRB. Вещественный массив *a*, содержащий общее представление матрицы *A*, имеющей одну верхнюю и три нижних кодиагонали, копируется в вещественный массив *b*, содержащий ленточное представление матрицы *A*.

```
program rBand
 use msimsl
 integer(4), parameter :: Lda = 4, Ldb = 5, n = 4, nlc = 3, nuc = 1
 real(4) :: a(Lda, n), b(Ldb, n)
 a = reshape((/      1.0,    2.0,    0.0,    0.0,                &
                    -2.0,    1.0,    3.0,    0.0,                &
                     0.0,   -3.0,    1.0,    4.0,                &
                    -7.0,    0.0,   -4.0,     1.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call crgrb(n, a, Lda, nlc, nuc, b, Ldb)        ! Копируем a в b
 call wrrrn('b', nuc+nlc+1, n, b, Ldb, 0)       ! Вывод результата
end program rBand
```

Результат:

```
                    b
          1        2          3        4
   1   0.000    2.000      3.000    4.000
   2   1.000    1.000      1.000    1.000
   3   2.000   -3.000     -4.000    0.000
   4   0.000    0.000      0.000    0.000
   5  -7.000    0.000      0.000    0.000
```

Пример для CRBRG. Вещественный массив *a*, содержащий ленточное представление матрицы *A* с одной верхней и одной нижней кодиагоналями, копируется в вещественный массив *b*, содержащий общее представление матрицы *A*.

```
program bReal
 use msimsl
 integer(4), parameter :: Lda = 3, Ldb = 3, n = 3, nlc = 1, nuc = 1
```

```
real(4) :: a(Lda, n), b(Ldb, n)
a = reshape((/      0.0,    1.0,    1.0,                &
                    4.0,    3.0,    2.0,                &
                    2.0,    2.0,    0.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
call crbrg(n, a, Lda, nlc, nuc, b, Ldb)        ! Копируем a в b
call wrrrn('b', n, n, b, Ldb, 0)               ! Вывод результата
end program bReal
```

Результат:

```
                    b
              1          2          3
   1      4.000      1.000      0.000
   2      2.000      3.000      1.000
   3      0.000      2000       2.000
```

Пример для CCGCB. Комплексный массив *a*, содержащий общее представление матрицы *A*, имеющей одну верхнюю и три нижних кодиагонали, копируется в комплексный массив *b*, содержащий ленточное представление матрицы *A*.

```
program cBand
 use msimsl
 integer(4), parameter :: Lda = 4, Ldb = 5, n = 4, nlc = 3, nuc = 1
 complex(4) :: a(Lda, n), b(Ldb, n)
 a = reshape((/      (1.0, 0.0),    (2.0, 1.0),    (0.0, 0.0),    (0.0, 0.0),      &
                    (-2.0, 1.0),    (1.0, 0.0),    (3.0, 2.0),    (0.0, 0.0),      &
                     (0.0, 0.0),   (-3.0, 2.0),    (1.0, 0.0),    (4.0, 3.0),      &
                    (-7.0, 1.0),    (0.0, 0.0),   (-4.0, 3.0),    (1.0, 0.0) /),   &
                 shape = (/ Lda, n /), order = (/ 2, 1 /))
 call ccgcb(n, a, Lda, nlc, nuc, b, Ldb)       ! Копируем a в b
 call wrcrn('b', nuc+nlc+1, n, b, Ldb, 0)      ! Вывод результата
end program cBand
```

Результат:

```
                                   b
              1                  2                  3                  4
   1     ( 0.00, 0.00)      ( 2.00, 1.00)      ( 3.00, 2.00)      ( 4.00, 3.00)
   2     ( 1.00, 0.00)      ( 1.00, 0.00)      ( 1.00, 0.00)      ( 1.00, 0.00)
   3     (-2.00, 1.00)      (-3.00, 2.00)      (-4.00, 3.00)      ( 0.00, 0.00)
   4     ( 0.00, 0.00)      ( 0.00, 0.00)      ( 0.00, 0.00)      ( 0.00, 0.00)
   5     (-7.00, 1.00)      ( 0.00, 0.00)      ( 0.00, 0.00)      ( 0.00, 0.00)
```

Пример для CCBCG. Комплексный массив *a*, содержащий ленточное представление матрицы *A* с одной верхней и тремя нижними кодиагоналя-

ми, копируется в комплексный массив b, содержащий общее представление матрицы A.

```
program cBand
 use msimsl
 integer(4), parameter :: Lda = 5, Ldb = 4, n = 4, nlc = 3, nuc = 1
 complex(4) :: a(Lda, n), b(Ldb, n)
 a = reshape((/      (0.0, 0.0),    (2.0, 1.0),    (3.0, 2.0),    (4.0, 3.0),      &
                     (1.0, 0.0),    (1.0, 0.0),    (1.0, 2.0),    (1.0, 0.0),      &
                    (-2.0, 1.0),   (-3.0, 2.0),   (-4.0, 3.0),    (0.0, 0.0),      &
                     (0.0, 0.0),    (0.0, 0.0),    (0.0, 0.0),    (0.0, 0.0),      &
                    (-7.0, 1.0),    (0.0, 0.0),    (0.0, 0.0),    (0.0, 0.0) /),   &
                 shape = (/ Lda, n /), order = (/ 2, 1 /))
 call ccbcg(n, a, Lda, nlc, nuc, b, Ldb)      ! Копируем a в b
 call wrcrn('b', n, n, b, Ldb, 0)             ! Вывод результата
end program cBand
```

Результат:

b

	1	2	3	4
1	(1.00, 0.00)	(2.00, 1.00)	(0.00, 0.00)	(0.00, 0.00)
2	(-2.00, 1.00)	(1.00, 0.00)	(3.00, 2.00)	(0.00, 0.00)
3	(0.00, 0.00)	(-3.00, 2.00)	(1.00, 0.00)	(4.00, 3.00)
4	(-7.00, 1.00)	(0.00, 0.00)	(-4.00, 3.00)	(1.00, 0.00)

Пример для CSFRG. Вещественный массив a, содержащий верхнюю треугольную часть симметрической 3×3-матрицы A, преобразовывается в массив, содержащий общее представление симметрической матрицы A.

```
program tSimm
 use msimsl
 integer(4), parameter :: Lda = 3, n = 3
 real(4) :: a(Lda, n)
 a = reshape((/      0.0,    3.0,    4.0,                  &
                     0.0,    1.0,    5.0,                  &
                     0.0,    0.0,     2.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call csfrg(n, a, Lda)                        ! Преобразовываем a
 call wrrrn('a', n, n, a, Lda, 0)             ! Вывод результата
end program tSimm
```

Результат:

a

	1	2	3
1	0.000	3.000	4.000
2	3.000	1.000	5.000
3	4.000	5.000	2.000

Пример для CHFCG. Вещественный массив *a*, содержащий верхнюю треугольную часть эрмитовой 3×3-матрицы *A*, преобразовывается в массив, содержащий общее представление эрмитовой матрицы *A*.

```
program tH
 use msimsl
 integer(4), parameter :: Lda = 3, n = 3
 complex(4) :: a(Lda, n)
 a = reshape((/      (1.0, 0.0),   (1.0, 1.0),   (1.0, 2.0),          &
                     (0.0, 0.0),   (2.0, 0.0),   (2.0, 2.0),          &
                     (0.0, 0.0),   (0.0, 0.0),   (3.0, 0.0) /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call chfcg(n, a, Lda)                    ! Преобразовываем a
 call wrcrn('a', n, n, a, Lda, 0)         ! Вывод результата
end program tH
```

Результат:

a

	1	2	3
1	(1.000, 0.000)	(1.000, 1.000)	(1.000, 2.000)
2	(1.000, -1.000)	(2.000, 0.000)	(2.000, 2.000)
3	(1.000, -2.000)	(2.000,-2.000)	(3.000, 0.000)

Пример для CSBRB. Вещественный массив *a* с двумя верхними кодиагоналями, содержащий ленточное представление верхней треугольной части симметрической 4×4-матрицы *A*, преобразовывается в массив *b* с двумя верхними и нижними кодиагоналями, содержащий ленточное представление всей матрицы *A*.

```
program sbReal
 use msimsl
 integer(4), parameter :: n = 4, nuca = 2, Lda = nuca + 1,  &
                          nlcb = nuca, nucb = nuca, Ldb = nlcb + nucb + 1
 real(4) :: a(Lda, n), b(Ldb, n)
 a = reshape((/      0.0,    0.0,    2.0,    1.0,          &
                     0.0,    2.0,    3.0,    1.0,          &
                     1.0,    2.0,    3.0,    4.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call csbrb(n, a, Lda, nuca, b, Ldb, nlcb, nucb)     ! Копируем a в b
 call wrrrn('b', nlcb+nucb+1, n, b, Ldb, 0)          ! Вывод результата
end program sbReal
```

Результат:

b

	1	2	3	4
1	0.000	0.000	2.000	1.000
2	0.000	2.000	3.000	1.000
3	1.000	2.000	3.000	4.000

4	2.000	3.000	1.000	0.000
5	2.000	1.000	0.000	0.000

Пример для CHBCB. Комплексный массив *a* с одной верхней кодиагональю, содержащий ленточное представление эрмитовой 3×3-матрицы *A*, копируется в массив *b*, содержащий ленточное представление всей матрицы.

```
program hBand
 use msimsl
 integer(4), parameter :: n = 3, nuca = 1, Lda = nuca + 1,   &
                          nlcb = nuca, nucb = nuca, Ldb = nlcb + nucb + 1
 complex(4) :: a(Lda, n), b(Ldb, n)
 a = reshape((/      (0.0, 0.0),   (-1.0, 1.0),   (-2.0, 2.0),                &
                     (1.0, 0.0),    (1.0, 0.0),     (1.0, 0.0) /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call chbcb(n, a, Lda, nuca, b, Ldb, nlcb, nucb)      ! Копируем a в b
 call wrcrn('b', nlcb+nucb+1, n, b, Ldb, 0)           ! Вывод результата
end program hBand
```

Результат:

b

	1	2	3
1	(0.000, 0.000)	(-1.000, 1.000)	(-2.000, 2.000)
2	(1.000, 0.000)	(1.000, 0.000)	(1.000, 0.000)
3	(-1.000, -1.000)	(-2.000, -2.000)	(0.000, 0.000)

4.6.3. Билинейная и квадратичная формы

Билинейная форма, равная сумме

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij} x_i y_j ,$$

возвращается функцией BLINF или DBLINF:

value = BLINF(*nra*, *nca*, *a*, *Lda*, *x*, *y*)

Входные данные: nra, *nca*, *a*, *Lda*, *x*, *y*.

Выходные данные: BLINF.

Тип *a*, *x* и *y* – REAL(4) или REAL(8) в случае применения DBLINF.

Тип *nra*, *nca* и *Lda* – INTEGER(4).

nra и *nca* – соответственно число рядов и столбцов в матрице *A*.

a – массив формы (*Lda*, *nca*), представляющий матрицу *A*.

Lda – ведущий размер массива *a*.

x – вектор размера *nra*.

y – вектор размера *nca*.

Формула для вычисления билинейной формы является раскрытием скалярного произведения $(A^T x, y)$, первым элементом которого является произведение матрицы A^T и вектора *x*. Заметим, что $(A^T x, y) = (xA, y) = (x, Ay)$.

Замечание. Квадратичная форма $\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j$ вычисляется, если в $(A^T x, y)$ вектор y заменить на вектор x.

Пример:

```
program bLin
 use text_transfer                          ! Для вывода русского текста
 use msimsl
 integer(4), parameter :: Lda = 5, nra = 5, nca = 2
 real(4) :: a(Lda, nca), x(nra), y(nca), value
 a = reshape((/        -2.0,    2.0,                  &
                        3.0,   -6.0,                  &
                       -4.0,    7.0,                  &
                        1.0,   -8.0,                  &
                        0.0,   10.0 /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
 x = (/ 1.0, -2.0, 3.0, -4.0, -5.0 /)
 y = (/ -6.0, 3.0 /)
 value = blinf(nra, nca, a, Lda, x, y)          ! Билинейная форма
 print *, trim(ru_doswin('Билинейная форма: ', .false.)), value
 ! Вычислим билинейную форму, применив встроенные функции SUM и MATMUL
 value = dot_product(matmul(x, a), y)           ! Вернет 195.000000
end program bLin
```

Результат:

```
Билинейная форма:    195.000000
```

4.6.4. Оценка матричного степенного ряда

Результатом оценки матричного степенного ряда является матрица $B=\sum_{k=1}^{m} c_k A^{k-1}$, где c_k – числовые коэффициенты; возвращается подпрограммами POLRG и DPOLRG:

CALL POLRG(*n*, *a*, *Lda*, *ncoef*, *coef*, *b*, *Ldb*)

Входные данные: *n*, *a*, *Lda*, *ncoef*, *coef*, *Ldb*.

Выходные данные: *b*.

Тип *a*, *coef* и *b* – REAL(4) или REAL(8) в случае применения DPOLRG.

Тип *n*, *Lda*, *ncoef* и *Ldb* – INTEGER(4).

n – порядок матрицы *A*.

a – массив формы (*Lda*, *n*), представляющий $n\times n$-матрицу *A*.

Lda и *Ldb* – соответственно ведущие размеры массивов *a* и *b*.

ncoef – число коэффициентов.

coef – вектор размера *ncoef*, содержащий коэффициенты многочлена в порядке возрастания степени.

b – массив формы (*Ldb*, *n*), представляющий матрицу – результат вычислений матричного степенного ряда.

Оценка матричного степенного ряда выполняется POLRG по схеме Горнера:

$$B=\left(\ldots\left(\left(c_m A+c_{m-1}I\right)A+c_{m-2}I\right)A+\ldots+c_1 I\right),$$

где I – единичная $n\times n$-матрица.

Пример. Оценить матричный степенной ряд 3I + A + 2A2 с 3×3-матрицей A.

```
program pol
 use msimsl
 integer(4), parameter :: n = 3, ncoef = 3, Lda = n, Ldb = n
 integer(4) :: i
 real(4) :: a(Lda, n), b(Ldb, n), coef(ncoef)
 a = reshape((/        1.0,    3.0,    2.0,                &
                      -5.0,    1.0,    7.0,                &
                       1.0,    5.0,   -4.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 coef = (/ 3.0, 1.0, 2.0 /)
 call polrg (n, a, Lda, ncoef, coef, b, Ldb)   ! Вычисляем 3I + A + 2A²
 call wrrrn('B = 3I + A + 2*A**2', n, n, b, Ldb, 0)
 ! Получим тот же результат, применив встроенную функцию MATMUL
 b = a + 2*matmul(a, a)
 do i = 1, 3
   b(i, i) = b(i, i) + 3.0
 end do
 call wrrrn('B = 3I + A + 2*A**2', n, n, b, Ldb, 0)
end program pol
```

Результат:

```
        B = 3I + A + 2*A**2
              1          2          3
  1     -20.000     35.000     32.000
  2     -11.000     46.000    -55.000
  3     -55.000    -19.000    105.000
```

4.6.5. Умножение ленточной матрицы на вектор

В разделе рассматриваются две процедуры, вычисляющие произведение матрицы A и вектора x. Первая оперирует вещественными данными, вторая – комплексными. В обоих случаях массив a содержит ленточное представление матрицы A.

Произведение вещественных матрицы и вектора возвращают подпрограммы MURBV и DMURBV, а комплексных – MUCBV и DMUCBV:

CALL MURBV(*n*, *a*, *Lda*, *nlca*, *nuca*, *nx*, *x*, *ipath*, *ny*, *y*)

CALL MUCBV(*n*, *a*, *Lda*, *nlca*, *nuca*, *nx*, *x*, *ipath*, *ny*, *y*)

Входные данные: n, a, Lda, nlca, nuca, nx, x, ipath, ny.

Выходные данные: y.

Тип *a*, *x* и *y* – REAL(4) или REAL(8), если применяется MURBV или DMURBV; или COMPLEX(4) или COMPLEX(8) в случае MUCBV или DMUCBV.

Тип *n*, *Lda*, *nlca*, *nuca*, *nx*, *ipath* и *ny* – INTEGER(4).

n – порядок матрицы *A*.

a – массив формы (*Lda*, *n*), содержащий ленточное представление *n*×*n*-матрицы *A*.

Lda – ведущий размер массива *a*; должен быть не менее *nlca* + *nuca* + 1.

nlca и *nuca* – соответственно число верхних и нижних кодиагоналей ленточной матрицы *A*.

nx и *ny* – соответственно длина векторов *x* и *y*, причем *nx* = *ny* = *n*.

x и *y* – векторы длины *nx* и *ny*.

ipath – флаг, равный 1 или 2; если *ipath* = 1, то вычисляется $y = Ax$, если *ipath* = 2, то – $y = A^T x$.

Пример для MUCBV:

```
program cProd
 use msimsl
 integer(4), parameter :: nlca = 2, nuca = 1, Lda = nlca + nuca + 1, n = 4, nx = n, ny = n
 integer(4) :: ipath
 complex(4) :: a(Lda, n), x(nx), y(ny)
 a = reshape((/      (0.0, 0.0),   (1.0, 2.0),   (3.0, 4.0),   (5.0, 6.0),      &
                    (-1.0, -1.0), (-1.0, -1.0), (-1.0, -1.0), (-1.0, -1.0),     &
                     (-1.0, 2.0),  (-1.0, 3.0),  (-2.0, 1.0),   (0.0, 0.0),     &
                      (2.0, 0.0),   (0.0, 2.0),   (0.0, 0.0),   (0.0, 0.0) /),  &
                    shape = (/ Lda, n /), order = (/ 2, 1 /))
 x = (/ (3.0, 4.0), (0.0, 0.0), (1.0, 2.0), (-2.0, -1.0) /)
 ipath = 1                                  ! Вычисляем y = Ax
 call mucbv(n, a, Lda, nlca, nuca, nx, x, ipath, ny, y)
 call wrcrn('y = Ax', 1, ny, y, 1, 0)       ! Вывод результата
end program cProd
```

Результат:

```
                          y = Ax
( 1.00, -7.00)    ( -16.00, 12.00)    ( 3.00, -12.00)    ( -3.00, 0.00)
```

4.6.6. Сложение ленточных матриц

В разделе рассматриваются две процедуры, возвращающие сумму ленточных матриц $C = A + B$. Процедуры складывают вещественные и комплексные матрицы. Массивы *a*, *b* и *c* содержат ленточные представления соответственно матриц *A*, *B* и *C*.

Сумму ленточных вещественных матриц возвращают подпрограммы ARBRB и DARBRB, а комплексных – ACBCB и DACBCB:

CALL ARBRB(*n*, *a*, *Lda*, *nlca*, *nuca*, *b*, *Ldb*, *nlcb*, *nucb*, *c*, *Ldc*, *nlcc*, *nucc*)

CALL ACBCB(*n*, *a*, *Lda*, *nlca*, *nuca*, *b*, *Ldb*, *nlcb*, *nucb*, *c*, *Ldc*, *nlcc*, *nucc*)

Входные данные: *n*, *a*, *Lda*, *nlca*, *nuca*, *b*, *Ldb*, *nlcb*, *nucb*, *Ldc*, *nlcc*, *nucc*.

Выходные данные: *c*.

Тип *a*, *b* и *c* – REAL(4) или REAL(8), если применяется ARBRB или DARBRB; или COMPLEX(4) или COMPLEX(8) в случае ACBCB или DACBCB..

Тип *n*, *Lda*, *nlca*, *nuca*, *Ldb*, *nlcb*, *nucb*, *Ldc*, *nlcc* и *nucc* – INTEGER(4).

n – порядок матриц *A*, *B* и *C*.

a, *b* и *c* – соответственно массивы форм (*Lda*, *n*), (*Ldb*, *n*) и (*Ldc*, *n*), содержащие ленточные представления *n*×*n*-матриц *A*, *B* и *C*.

Lda, *Ldb* и *Ldc* – соответственно ведущие размеры массивов *a*, *b* и *c*; должны быть не менее *nlc** + *nuc** + 1, где * – это *a*, *b* или *c*.

nlca nlcb, *nlcc* и *nuca*, *nucb*, *nucc* – соответственно число верхних и нижних кодиагоналей ленточных матриц *A*, *B* и *C*, причем *nlcc* ≥ MAX(*nlca*, *nlcb*) и *nucc* ≥ MAX(*nuca*, *nucb*).

Пример для ARBRB:

```
program rAdd
 use msimsl
 integer(4), parameter :: Lda=3, Ldb=3, Ldc=4, n=4, nlca=1, nlcb=2, nlcc=2, nuca=1, nucb=0,
nucc=1
 real(4) :: a(Lda, n), b(Ldb, n) , c(Ldc, n)
 a = reshape((/      0.0,    2.0,    3.0,   -1.0,              &
                     1.0,    1.0,    1.0,    1.0,              &
                     0.0,    3.0,    4.0,    0.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 b = reshape((/      3.0,    3.0,    3.0,    3.0,              &
                     1.0,   -2.0,    1.0,    0.0,              &
                    -1.0,    2.0,    0.0,    0.0 /), shape = (/ Ldb, n /), order = (/ 2, 1 /))
 ! C = A + B
 call arbrb (n, a, Lda, nlca, nuca, b, Ldb, nlcb, nucb, c, Ldc, nlcc, nucc)
 call wrrrn('C = A + B', nlcc + nucc + 1, n, c, Ldc, 0)
end program rAdd
```

Результат:

C = A + B

	1	2	3	4
1	0.000	2.000	3.000	-1.000
2	4.000	4.000	4.000	4.000
3	1.000	1.000	5.000	0.000
4	-1.000	2.000	0.000	0.000

4.6.7. Вычисление нормы матрицы

Подпрограммы, возвращающие норму матрицы, перечислены в табл. 4.8; параметры подпрограмм – в табл. 4.9.

Таблица. 4.8. Подпрограммы, вычисляющие норму матрицы A

Подпрограмма	*Вычисляемая норма*
CALL NRIRR(*nra*, *nca*, *a*, *Lda*, *anorm*) *Входные параметры*: *nra*, *nca*, *a*, *Lda*. *Выходной параметр*: *anorm*. Тип *a* и *anorm* – REAL(4) (или REAL(8) в случае DNRIRR); тип остальных параметров – INTEGER(4)	∞-норма – максимум сумм абсолютных значений элементов рядов $m\times n$-матрицы A: $\|A\|_\infty = \max\limits_{1\le i\le m}\sum\limits_{j=1}^{n}\|a_{ij}\|$
CALL NR1RR(*nra*, *nca*, *a*, *Lda*, *anorm*) Описание параметров такое же, как и у NRIRR	1-норма – максимум сумм абсолютных значений элементов столбцов $m\times n$-матрицы A: $\|A\|_1 = \max\limits_{1\le j\le n}\sum\limits_{i=1}^{m}\|a_{ij}\|$
CALL NR2RR(*nra*, *nca*, *a*, *Lda*, *anorm*) Описание параметров такое же, как и у NRIRR	Норма Фробениуса $m\times n$-матрицы A: $\|A\|_F = \sqrt{\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{n}a_{ij}^2}$
CALL NR1RB(*n*, *a*, *Lda*, *nlca*, *nuca*, *anorm*) *Входные параметры*: *n*, *a*, *Lda*, *nlca*, *nuca*. *Выходной параметр*: *anorm*. Тип *a* и *anorm* – REAL(4) (или REAL(8) в случае DNR1RB); тип остальных параметров – INTEGER(4)	1-норма вещественной ленточной $m\times n$-матрицы A
CALL NR1CB(*n*, *a*, *Lda*, *nlca*, *nuca*, *anorm*) Описание параметров такое же, как и у NRIRB, за тем исключением, что тип *a* – COMPLEX(4) (или COMPLEX(8) в случае DNR1CB);	1-норма комплексной ленточной $m\times n$-матрицы A: $\|A\|_1 = \max\limits_{1\le j\le n}\sum\limits_{i=1}^{m}\left(\left\|\mathrm{REAL}(a_{ij})\right\|+\left\|\mathrm{AIMAG}(a_{ij})\right\|\right)$

Таблица 4.9. Параметры подпрограмм из табл. 4.8

Параметры	*Смысл*
nra и *nca*	Соответственно число рядов и столбцов в *nra*×*nca*-матрице *A*, норма которой вычисляется
a	Массив формы (*Lda*, *nca*), содержащий либо общее представление *nra*×*nca*-матрицы A (подпрограммы NRIRR, NR1RR и NR2RR), либо ее ленточное представление (подпрограммы NR1RB и NR1CB)
Lda	Ведущий размер массива *a*

anorm	Норма *nra*×*nca*-матрицы *A*
nlca и *nuca*	Соответственно число нижних и верхних кодиагоналей в ленточной матрице *A*

Пример для NRIRR, NR1RR и NR2RR:

```
program norm
 use text_transfer
 use msimsl
 integer(4), parameter :: Lda = 3, nca = 4, nra = 3
 real(4) :: a(Lda, nca), anorm, anorm1, anormf
 a = reshape((/      1.0,    0.0,    2.0,    0.0,                &
                     3.0,    4.0,   -1.0,    0.0,                &
                     2.0,    1.0,    2.0,     1.0 /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
 call nrirr(nra, nca, a, Lda, anorm)          ! ∞-норма
 call nr1rr (nra, nca, a, Lda, anorm1)        ! 1-норма
 call nr2rr (nra, nca, a, Lda, anormf)
 print *, trim(ru_doswin('Бесконечность-норма: ', .false.)), anorm
 print *, trim(ru_doswin('1-норма: ', .false.)), anorm1
 print *, trim(ru_doswin('Норма Фробениуса: ', .false.)), anormf
 ! Вычисление ∞-нормы с применением встроенных функций MAXVAL, SUM и ABS
 anorm = maxval(sum(abs(a), dim = 2))     ! Вернет 8.000000
 ! Вычисление 1-нормы с применением встроенных функций MAXVAL, SUM и ABS
 anorm1 = maxval(sum(abs(a), dim = 1))   ! Вернет 6.000000
 ! Вычисление нормы Фробениуса с применением встроенных функций SQRT и SUM
 anormf = sqrt(sum(sum(a * a, dim = 1)))   ! Вернет 6.403124
end program norm
```

Результат:

```
Бесконечность-норма:    8.000000
1-норма:                6.000000
Норма Фробениуса:       6.403124
```

Пример для NRICB:

```
program cNorm
 use text_transfer
 use msimsl
 integer(4), parameter :: Lda = 4, n = 4, nlca = 2, nuca = 1
 real(4) :: anorm1
 complex(4) :: a(Lda, n)
 complex(4), allocatable :: b(:, :)
 a = reshape((/      (0.0, 0.0),     (2.0, 3.0),     (-1.0, 1.0),    (-2.0, -1.0),      &
                    (-2.0, 3.0),     (1.0, 0.0),    (-4.0, -1.0),     (0.0, -4.0),      &
                     (2.0, 2.0),     (4.0, 6.0),     (3.0, 2.0),      (0.0, 0.0),       &
                    (0.0, -1.0),     (2.0, 1.0),     (0.0, 0.0),      (0.0, 0.0) /),    &
                 shape = (/ Lda, n /), order = (/ 2, 1 /))
```

```
 call nr1cb(n, a, Lda, nlca, nuca, anorm1)  ! 1-норма
 print *, trim(ru_doswin('1-норма комплексной матрицы: ', .false.)), anorm1
 ! Вычисление 1-нормы с применением встроенных функций
 ! Прежде скопируем a в b – в массив, содержащий общее представление матрицы A
 allocate(b(n, n))
 call ccbcg(n, a, Lda, nlca, nuca, b, n)
 anorm1 = maxval(sum(abs(real(b)) + abs(aimag(b)), dim = 1))
 deallocate(b)
 print *, anorm1                          ! 19.000000
end program cNorm
```

Результат:

```
1-норма комплексной матрицы:          19.000000
```

4.6.8. Вычисление расстояния между точками

Функции, возвращающие расстояние между точками, перечислены в табл. 4.10; их параметры – в табл. 4.11.

Таблица. 4.10. Функции, вычисляющие расстояние между точками

Функция	*Вычисляемое расстояние*
dist = DISL2(*n*, *x*, *incx*, *y*, *incy*) *Входные параметры*: *n*, *x*, *incx*, *y*, *incy*. *Результат*: DISL2. Тип *x* и *y* – REAL(4) (или REAL(8) в случае DDISL2); тип *n*, *incx*, *incy* – INTEGER(4)	Евклидово (2-норма) расстояние между точками, задаваемыми векторами *x* и *y*: $d_2 = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
*d*1 = DISL1(*n*, *x*, *incx*, *y*, *incy*) Описание параметров такое же, как и у DISL2	Манхеттеново (1-норма) расстояние между точками, задаваемыми векторами *x* и *y*: $d_1 = \sum_{i=1}^{n}\lvert x_i - y_i\rvert$
di = DISLI(*n*, *x*, *incx*, *y*, *incy*) Описание параметров такое же, как и у DISL2	∞-норма расстояние между точками, задаваемыми векторами *x* и *y*: $di = \max_{1\le i\le n}\lvert x_i - y_i\rvert$

Таблица 4.11. Параметры функций из табл. 4.10

Параметры	*Смысл*
n	Размер векторов *x* и *y*
x и *y*	Массивы формы (*Lda*, *nca*), содержащие либо общее представление *nra*×*nca*-матрицы *A* (подпрограммы NRIRR, NR1RR и NR2RR), либо ее ленточное представление (подпрограммы NR1RB и NR1CB)
incx и *incy*	Соответственно шаги между элементами векторов *x* и *y*; смысл параметров пояснен в разд. 4.4.1

Пример для DISL2, DISL1 и DISLI:

```
program dist
 use text_transfer
 use msimsl
 integer(4), parameter :: incx = 1, incy = 1, n = 4
 real(4) :: d2, d1, di, x(n), y(n)
 x = (/ 1.0, -1.0, 0.0, 2.0 /); y = (/ 4.0, 2.0, 1.0, -3.0 /)
 d2 = disl2(n, x, incx, y, incy)
 d1 = disl1(n, x, incx, y, incy)
 di = disli(n, x, incx, y, incy)
 print *, trim(ru_doswin('Евклидово расстояние: ', .false.)), d2
 print *, trim(ru_doswin('Манхеттеново расстояние: ', .false.)), d1
 print *, trim(ru_doswin('Бесконечность-норма расстояние: ', .false.)), di
 ! Вычисление расстояния с применением встроенных функций
 d2 = sqrt(sum((x - y) * (x - y)))          ! Вернет 6.633250
 d1 = sum(abs(x - y))                       ! Вернет 12.000000
 di = maxval(abs(x - y))                    ! Вернет 5.000000
end program dist
```

Результат:

```
Евклидово расстояние:                 6.633250
Манхеттеново расстояние:              12.000000
Бесконечность-норма расстояние:       5.000000
```

4.6.9. Вычисления с повышенной точностью

В разделе приводятся подпрограммы, использующие для умножения квадратичную точность: 128 бит для вещественных данных и 256 бит – для комплексных. При вычислениях используется аккумулятор, обладающий указанной точностью. В него заносятся или к его содержимому добавляются результаты умножения. Предварительно в аккумулятор можно записать некоторое число.

При работе с вещественными данными аккумулятор – это массив *qacc* типа REAL(8) размера 2. Скалярные параметры *da* и *db* также имеют тип REAL(8). В случае комплексных данных аккумулятор – это массив *zacc* типа REAL(8) размера 4. Скалярные параметры *za* и *zb* имеют тип COMPLEX(8). Подпрограммы, использующие аккумулятор, описаны в табл. 4.12.

Таблица 4.12. Подпрограммы, выполняющие вычисления с двойной точностью

Подпрограммы	*Назначение*
CALL DQINI(*da*, *qacc*), CALL ZQINI(*za*, *zacc*)	Запись числа *da* (*za*) в аккумулятор *qacc* (*zacc*) *qacc* = *da* (*zacc*= *za*)
CALL DQSTO(*qacc*, *da*), CALL ZQSTO(*zacc*, *za*)	Присваивание *da* = *qacc* (*za* = *zacc*)

CALL DQADD(*da*, *qacc*), CALL ZQADD(*za*, *zacc*)	Добавление в аккумулятор числа *da* (*za*): *qacc* = *qacc* + *da* (*zacc* = *zacc* + *za*)
CALL DQMUL(*da*, *db*, *qacc*), CALL ZQMUL(*za*, *zb*, *zacc*)	Добавление в аккумулятор произведения *da* * *db* (*za* * *zb*): *qacc* = *qacc* + *da* * *db* (*zacc* = *zacc* + *za* * *zb*)

Пример. В примере величина 1.0_8/3.0_8 вычисляется с квадратичной точностью. Для получения результата используются 4 итерации метода Ньютона вида $x_{k+1} = x_k + \left(x_k - ax_k^2\right)$. Значение a = 3. Начальное значение аккумулятора равно 1.0001/a. Также вычисляется ошибка ax - 1. Точность вычислений почти в 2 раза превышает точность, получаемую при использовании REAL(8).

```
program acc
 use msimsl
 integer(4) :: i
 real(8) :: a, dacc(2), error, sacc(2), x(2), x1, x2
 a = 3.0_8
 call dqini(1.0001_8 / a, x)
 do i=1, 4                          ! Вычисляем x(k + 1) = x(k) - a * x(k) * x(k) + x(k)
  x1 = x(1);  x2 = x(2)
  call dqadd(x1, x);  call dqadd(x2, x)     ! Вычисляем x + x
  call dqini(0.0_8, dacc)           ! Вычисляем x * x
  call dqmul(x1, x1, dacc);  call dqmul(x1, x2, dacc)
  call dqmul(x1, x2, dacc);  call dqmul(x2, x2, dacc)
  call dqini(0.0_8, sacc)           ! Вычисляем -a * (x * x)
  call dqmul(-a, dacc(1), sacc);  call dqmul(-a, dacc(2), sacc)
  call dqadd(sacc(1), x)            ! Вычисляем -a * (x * x) +(x + x)
  call dqadd(sacc(2), x)
 end do
 call dqini(0.0_8, sacc)            ! Вычисляем ошибку, равную a * x - 1
 call dqmul(a, x(1), sacc);  call dqmul(a, x(2), sacc)
 call dqadd(-1.0_8, sacc);  call dqsto(sacc, error)
 ! Ошибка должна быть меньше, чем ε_м^2
 print "(' a * x - 1 = ', d15.7, ' = ', f10.5, ' * macheps**2')", error, error / dmach(4)**2
end program acc
```

Результат:

```
a * x - 1 =   0.6162976D - 32 =    0.12500 * macheps**2
```

5. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

В главе рассматриваются методы решения систем линейных уравнений. Из множества существующих представлены лишь методы, используемые процедурами IMSL. Причем приводимые методы относятся к прямым и работают с вещественными общего вида и симметрическими матрицами. Итерационные алгоритмы обсуждаются в гл. 6, посвященной описанию процедур IMSL для систем линейных уравнений с $n \times n$-матрицей.

5.1. ПОСТАНОВКА ЗАДАЧИ

Система из n линейных уравнений с n неизвестными имеет вид:

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1,\\
&a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n.
\end{aligned}
$$

Матричная форма записи системы:

$$
\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}
=
\begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix}
\quad \text{или} \quad Ax = b,\ A \in \mathbf{R}^{n \times n},\ x \in \mathbf{R}^n,\ b \in \mathbf{R}^n,
$$

где $A = (a_{ij})$ – числовая $n \times n$-матрица коэффициентов, x – вектор-столбец неизвестных, b – заданный вектор-столбец.

Система линейных уравнений, обладающая решениями, называется *совместной*. В противном случае система *несовместна*. Всегда совместна система линейных *однородных* уравнений, т. е. уравнений, свободные члены которых равны нулю, так как обладает по крайней мере нулевым решением $(0, 0, \ldots, 0)^T$.

Система называется *определенной*, если она имеет одно-единственное решение, и *неопределенной* – в противном случае. Система оказывается определенной, если входящие в нее уравнения линейно независимы. Определитель такой системы (матрицы A) отличен от нуля. Процедуры IMSL ориентированы на решение определенных систем.

5.2. МЕТОД ИСКЛЮЧЕНИЯ ГАУССА

5.2.1. Идея метода исключения

Рассмотрим систему линейных уравнений

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & a_{14}x_4 & = & b_1, \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & a_{24}x_4 & = & b_2, \\ a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & + & a_{34}x_4 & = & b_3, \\ a_{41}x_1 & + & a_{42}x_2 & + & a_{43}x_3 & + & a_{44}x_4 & = & b_4. \end{array}$$

Идея метода исключения Гаусса в том, чтобы исходную систему привести к треугольному виду:

$$\begin{array}{ccccccccc} a^*_{11}x_1 & + & a^*_{12}x_2 & + & a^*_{13}x_3 & + & a^*_{14}x_4 & = & b^*_1, \\ & & a^*_{22}x_2 & + & a^*_{23}x_3 & + & a^*_{24}x_4 & = & b^*_2, \\ & & & & a^*_{33}x_3 & + & a^*_{34}x_4 & = & b^*_3, \\ & & & & & & a^*_{44}x_4 & = & b^*_4. \end{array}$$

Процесс приведения называется *прямым ходом*. После приведения выполняется *обратный ход*, в результате которого вычисляются искомые значения неизвестных: сначала x_4, затем x_3, затем x_2 и в последнюю очередь x_1. Алгоритм выполнения обратного хода тривиален.

Реализацию прямого хода рассмотрим на примере системы уравнений

$$\begin{array}{ccccccccr} 2x_1 & + & 3x_2 & + & 11x_3 & + & 5x_4 & = & 2, \\ x_1 & + & x_2 & + & 5x_3 & + & 2x_4 & = & 1, \\ 2x_1 & + & x_2 & + & 3x_3 & + & 2x_4 & = & -3, \\ x_1 & + & x_2 & + & 3x_3 & + & 4x_4 & = & -3. \end{array}$$

Исключим на первом шаге из трех последних уравнений x_1, оставив без изменений уравнение 1. Из второго уравнения исключается x_1 так: умножим уравнение 1 на $a_{21}/a_{11} = 1/2 = 0{,}5$ и вычтем результат из уравнения 2. Аналогично поступим и с уравнениями 3 и 4. Умножая уравнение 1 соответственно на $a_{31}/a_{11} = 2/2 = 1$ и $a_{41}/a_{11} = 1/2 = 0{,}5$ и вычитая результат первого умножения из уравнения 3, а второго из уравнения 4, получим систему

$$\begin{array}{ccccccccr} 2x_1 & + & 3x_2 & + & 11x_3 & + & 5x_4 & = & 2, \\ & & -0{,}5x_2 & - & 0{,}5x_3 & - & 0{,}5x_4 & = & 0, \\ & & -2x_2 & - & 8x_3 & - & 3x_4 & = & -5, \\ & & -0{,}5x_2 & - & 2{,}5x_3 & + & 1{,}5x_4 & = & -4. \end{array}$$

Заметим, что каждый раз на первом шаге при поиске множителя для уравнения 1 коэффициенты a_{21}, a_{31} и a_{41} уравнений 2, 3 и 4 делились на один и тот же коэффициент – a_{11}. Коэффициент, на который выполняется деле-

ние, называется *ведущим элементом*. Уравнение, содержащее ведущий элемент, называется *ведущим уравнением*.

Легко видеть, что на втором шаге следует, взяв в качестве ведущего элемента коэффициент a_{22}, исключить x_2 из уравнений 3 и 4, оставив без изменений уравнения 1 и 2. После второго шага имеем систему

$$\begin{array}{rcrcrcrcr} 2x_1 & + & 3x_2 & + & 11x_3 & + & 5x_4 & = & 2, \\ & & -0{,}5x_2 & - & 0{,}5x_3 & - & 0{,}5x_4 & = & 0, \\ & & & & -6x_3 & - & x_4 & = & -5, \\ & & & & -2x_3 & + & 2x_4 & = & -4. \end{array}$$

И, завершая прямой ход, исключим, взяв в качестве ведущего элемента коэффициент a_{33}, x_3 из уравнения 4, оставив без изменений уравнения 1, 2 и 3:

$$\begin{array}{rcrcrcrcr} 2x_1 & + & 3x_2 & + & 11x_3 & + & 5x_4 & = & 2, \\ & & -0{,}5x_2 & - & 0{,}5x_3 & - & 0{,}5x_4 & = & 0, \\ & & & & -6x_3 & - & x_4 & = & -5, \\ & & & & & & (7/3)x_4 & = & -7/3. \end{array}$$

Выполнив обратный ход, найдем корни $x_4 = -1$, $x_3 = 1$, $x_2 = 0$, $x_1 = -2$.

В приводимой ниже программе, реализующей метод Гаусса, искомые корни заносятся в массив x. Коэффициенты системы уравнений хранятся в массиве a, а свободные члены – в векторе b. Функция RESHAPE преобразовывает одномерный массив, задаваемый посредством конструктора, в двумерный массив формы (n, n). Параметр *order* = (/ 2, 1 /) обеспечивает запись каждых n элементов конструктора в соответствующую строку массива-результата. При его отсутствии формирование массива – a будет выполнено по столбцам.

Массив $a2$ и вектор $b2$, в которых запоминаются исходные данные, введены для проверки результата. Суть проверки в вычислении разности b - Ax. (Произведение Ax находится встроенной функцией MATMUL.) Такая разность называется *невязкой*. Проверка весьма условна, поскольку в методе исключения Гаусса малые невязки не гарантируют высокой точности (разд. 5.6.1).

```
program Gauss
 integer(4), parameter :: n = 4
 integer(4) :: i, k
 real(4) :: a(n, n), a2(n, n), p(n)          ! a – массив коэффициентов системы Ax = b
 real(4) :: b2(n), b(n)                      ! Вектор b содержит правую часть системы
 real(4) :: x(n) = 0.0, s                    ! x – вектор, содержащий решения линейной системы
 a = reshape((/      2,    3,   11,    5,          &
                     1,    1,    5,    2,          &
                     2,    1,    3,    2,          &
                     1,    1,    3,     4 /), shape = (/ n, n /), order = (/ 2, 1 /))
```

```
b = (/ 2, 1, -3, -3 /)
a2 = a; b2 = b                          ! a2 и b2 используются для вычисления невязки
do k = 1, n - 1                         ! Прямой ход; k – номер ведущего уравнения
  p(k + 1: n) = a(k + 1:n, k) / a(k, k) ! Вычисляем множители k-го шага
  do i = k + 1, n                       ! i – номер уравнения
    a(i, k + 1:n) = a(i, k + 1:n) - a(k, k + 1:n) * p(i)
    b(i) = b(i) - b(k) * p(i)
  end do
end do
x(n) = b(n) / a(n, n)                   ! Выполняем обратный ход
do k = n - 1, 1, -1                     ! k – номер уравнения и вычисляемого
  s = sum(a(k, k + 1:n) * x(k + 1:n))   ! компонента вектора x
  x(k) = (b(k) - s) / a(k, k)
end do
print '(4f9.3)', x                      ! Вывод результата
print '(4f9.6)', b2 - matmul(a2, x)     ! Вывод невязки
end program Gauss
```

Реализованный метод имеет ряд недостатков. Так, на k-м шаге прямого хода при исключении x_k в качестве ведущего элемента выбирался a_{kk} – коэффициент перед x_k уравнения с номером k. При этом, однако, не анализируется величина a_{kk}, которая, очевидно, должна быть отлична от нуля. Более того, абсолютные величины ведущих элементов не должны быть близки к нулю, ибо в противном случае исключение Гаусса может давать сколь угодно плохие решения [6].

Для устранения этого недостатка в библиотеке IMSL при решении линейных систем методом исключения Гаусса реализована схема *частичного выбора*; в ней на k-м шаге исключения уравнение с номером k меняется местами с уравнением, в котором перед x_k находится максимальный по модулю коэффициент. В частности, в рассмотренном выше примере после первого исключения мы имели систему

$$\begin{array}{rcrcrcrcr} 2x_1 & + & 3x_2 & + & 11x_3 & + & 5x_4 & = & 2, \\ & & -0{,}5x_2 & - & 0{,}5x_3 & - & 0{,}5x_4 & = & 0, \\ & & -2x_2 & - & 8x_3 & - & 3x_4 & = & -5, \\ & & -0{,}5x_2 & - & 2{,}5x_3 & + & 1{,}5x_4 & = & -4 \end{array}$$

и затем в качестве ведущего был выбран элемент a_{22} = -0,5. При работе по схеме частичного выбора перед очередным исключением будет выполнена перестановка уравнений 2 и 3, что обеспечит выбор максимального по модулю ведущего элемента – коэффициента -2.

Можно в поисках ведущего элемента обменивать местами вместо строк столбцы, а можно и строки и столбцы. В принципе существует последовательность ведущих элементов, при которых точность решения максимальна.

В действительности при решении систем линейных уравнений число подлежащих учету факторов существенно больше. К ним относится, например, оценка точности результата, выполняемая по числу обусловленности матрицы A, или потребность решения m систем с одинаковой матрицей A и разными правыми частями, т. е. систем вида

$$Ax = b^{(1)}, Ax = b^{(2)}, ..., Ax = b^{(m)}.$$

Конечно, результат можно получить, m раз применяя метод исключения. Однако такой подход непродуктивен, и взамен используется модификация метода Гаусса, предполагающая предварительное представление исходной матрицы коэффициентов A в виде LU-разложения, которое затем употребляется для поиска решений каждой из заданных m линейных систем.

5.2.2. *LU*-разложение матрицы

Любую квадратную матрицу, имеющую отличные от нуля главные диагональные миноры, можно представить в виде произведения нижней и верхней треугольных матриц. Причем это разложение будет единственным, если заранее зафиксировать диагональные элементы одной из треугольных матриц, приняв их, например, равными единице. Так, матрица

$$A = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 5 & 4 \\ 2 & 4 & 14 \end{pmatrix}$$

представляется в виде произведения следующих треугольных L- и U-матриц:

$$A = LU = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 1.5 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 2 \\ 0 & 4 & 6 \\ 0 & 0 & 1 \end{pmatrix}.$$

Такое представление матрицы называется ее *LU-разложением.*

Для хранения LU-разложения $n \times n$-матрицы, в котором зафиксированы диагональные элементы L- или U-матрицы, достаточно одной матрицы. Например, вышеприведенное LU-разложение матрицы A представляется матрицей

$$F = \begin{pmatrix} 1 & -1 & 2 \\ -1 & 4 & 6 \\ 2 & 1.5 & 1 \end{pmatrix}.$$

Треугольные матрицы интересны, в частности, тем, что для них сравнительно просто находятся обратные матрицы. Причем если $A = LU$, то $A^{-1} = L^{-1}U^{-1}$. Из двух последних обстоятельств следует *алгоритм* решения системы линейных уравнений $Ax = b$:

1. Найти LU-разложение матрицы A.

2. Вычислить L^{-1} и U^{-1} и найти $A^{-1} = L^{-1}U^{-1}$.
3. Определить корни уравнения: $x = A^{-1}b$.

Понятно, что алгоритм результативен, если квадратную матрицу можно разложить на треугольные L- и U-матрицы.

Впрочем, такой алгоритм на практике не находит применения, поскольку при известном LU-разложении поиск корней системы линейных уравнений можно выполнить с меньшими временными затратами, использовав приведенные ниже сведения.

5.2.2.1. Вычисление LU-разложения матрицы

LU-разложение матрицы получается в результате применения к ней рассмотренного выше метода исключения Гаусса. Заметим, что k-й шаг прямого хода Гаусса равнозначен умножению матрицы $A^{(k-1)}$, полученной после шага k - 1, на матрицу

$$M^{(k)} = I_n - \tau^{(k)} e_k^T ,$$

где

$$\tau^{(k)} = \left(\underbrace{0, \ldots, 0}_{k}, \tau_{k+1}, \ldots, \tau_n \right), \ \tau_i = \frac{a_{ik}}{a_{kk}}, \ i = k+1, \ldots, n .$$

Величина τ_i, на которую умножается ведущее уравнение с целью выполнения исключения в уравнении i, называется *множителем k-го шага*. Всего на шаге k вычисляется n - k множителей. Матрица $M^{(k)}$ называется *матрицей преобразования Гаусса*, а вектор τ - *вектором Гаусса*.

В табличном виде матрица

$$M^{(k)} = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & & 1 & 0 & & 0 \\ 0 & & -\tau_{k+1} & 1 & & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -\tau_n & 0 & \cdots & 1 \end{pmatrix}.$$

Легко убедиться, что матрица, обратная $M^{(k)}$, *есть* $M^{(k)^{-1}} = I_n + \tau^{(k)} e_k^T$ и что матрицы $M^{(k)}$ и $M^{(k-1)}$ являются нижними унитреугольными.

Итак, $A^{(k)} = M^{(k)} A^{(k-1)}$. Матрица

$$U = M^{(n-1)} \ldots M^{(1)} A ,$$

получаемая в результате выполнения прямого хода, является верхней треугольной. В то же время матрица

$$L = M^{(n-1)^{-1}} \dots M^{(1)^{-1}},$$

будучи произведением нижних унитреугольных матриц, сама является таковой. Поэтому матрица

$$A = M^{(n-1)^{-1}} \dots M^{(1)^{-1}} U = LU$$

является произведением нижней унитреугольной L и верхней треугольной U матриц, получаемых в результате прямого хода Гаусса.

Для их хранения в процедурах IMSL используется матрица FAC, содержащая, если не выполнялись в процессе LU-разложения перестановки строк, в своей верхней треугольной части матрицу U, а в нижней – матрицу

$$L^{-1} = M^{(n-1)} \dots M^{(1)}.$$

Причем k-й столбец матрицы L^{-1} содержит вектор Гаусса $\tau^{(k)}$ - множители k-го шага, взятые с отрицательным знаком.

Пример. Выполняется LU-разложение 4×4-матрицы. Результат возвращается матрицей FAC.

```
program LU
 use msimsl, nouse => fac                    ! Для вызова WRRRN
 integer(4), parameter :: n = 4
 real(4) :: a(n, n), fac(n, n)               ! a – массив, представляющий матрицу A
 a = reshape((/        2,    3,    11,    5,          &
                       1,    1,    5,     2,          &
                       2,    1,    3,     2,          &
                       1,    1,    3,      4 /), shape = (/ n, n /), order = (/ 2, 1 /))
 call Lufac(n, a, fac)                       ! Вычисляем LU-разложение матрицы A
 call wrrrn('fac', n, n, fac, n, 0)          ! Вывод результата
end program LU
subroutine Lufac(n, a, fac)                  ! Возвращает LU-разложение матрицы A
 integer(4) :: n, i, k
 real(4) :: a(n, n), fac(n, n), p(n)
 fac = a
 do k = 1, n - 1                             ! Прямой ход; k – номер ведущего уравнения
  p(k + 1: n) = fac(k + 1:n, k) / fac(k, k)  ! Вычисляем множители k-го шага и формируем
  do i = k + 1, n                            ! матрицу U; i – номер уравнения
   fac(i, k + 1:n) = fac(i, k + 1:n) - fac(k, k + 1:n) * p(i)
  end do
  fac(k + 1: n, k) = -p(k + 1: n)            ! Формируем матрицу L-1
 end do
end subroutine Lufac
```

Результат:

	fac			
	1	2	3	4
1	2.00	3.00	11.00	5.00
2	-0.50	-0.50	-0.50	-0.50
3	-1.00	-4.00	-6.00	-1.00
4	-0.50	-1.00	-0.33	2.33

При этом

$$L=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0{,}5 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0{,}5 & 1 & 1/3 & 1 \end{pmatrix}, U=\begin{pmatrix} 2 & 3 & 11 & 5 \\ 0 & -0{,}5 & -0{,}5 & -0{,}5 \\ 0 & 0 & -6 & -1 \\ 0 & 0 & 0 & 7/3 \end{pmatrix} \text{ и } A = LU.$$

Замечания:

1. В ссылке на модуль msimsl.mod присутствует переименование *nouse => fac*. Оно необходимо для того, чтобы избежать повторного использования имени *fac*, поскольку это же имя содержит и указанный модуль.
2. *LU*-разложение требует $2n^3/3$ флопов.

5.2.2.2. Применение LU-разложения для решения линейных систем

По известному *LU*-разложению матрицы *A* системы линейных уравнений $Ax = b$ ее решение находится в 2 этапа:

1. Найти, решив систему $Ly = b$, вектор y – преобразованный вектор b.
2. Решить систему $Ux = y$ и найти вектор x, являющийся также и решением системы $Ax = b$.

Эти этапы вытекают из равенств

$$x = U^{-1}y = U^{-1}L^{-1}b = A^{-1}b\,.$$

Решение системы $Ly = b$ находится в результате *прямых подстановок*, при которых на k-м шаге определяется

$$y_k = (b_k - \sum_{j=1}^{k-1} l_{kj}x_j)/l_{kk}\,,$$

где l_{kj} – элемент с номером j в k-й строке матрицы L, а $l_{kk} = 1$.

Система $Ux = y$, поскольку матрица U является верхней треугольной, решается в результате выполнения обратного хода метода Гаусса, называемого также *обратными подстановками*.

Замечание. Прямые и обратные подстановки требуют n^2 флопов.

Пример. Первоначально выполняется *LU*-разложение 4×4-матрицы *A*. Затем решается линейная система $Ax = b$, использующая найденное разложение матрицы *A*.

```
program LU2
 use msimsl, nouse => fac                    ! Для вызова WRRRN
 integer(4), parameter :: n = 4
 real(4) :: a(n, n), fac(n, n)               ! a – массив, представляющий матрицу A
 real(4) :: x(n), b(n) = (/ 2.0, 1.0, -3.0, -3.0 /)
 a = reshape((/      2,    3,    11,    5,                &
                     1,    1,     5,    2,                &
                     2,    1,     3,    2,                &
                     1,    1,     3,     4 /), shape = (/ n, n /), order = (/ 2, 1 /))
 call Lufac(n, a, fac)                       ! Возвращает LU-разложение матрицы A
 call useLU(n, fac, b, x)                    ! Решаем линейную систему Ax = b
 call wrrrn('x', 1, n, x, 1, 0)              ! Вывод результата
end program LU2
subroutine Lufac(n, a, fac)                  ! Текст подпрограммы приведен
. . .                                        ! в предшествующем разделе
end subroutine Lufac
subroutine useLU(n, fac, b, x)               ! Решает линейную систему Ax = b
 integer(4) :: n, k
 real(4) :: fac(n, n), x(n), b(n), s
 ! Нижняя треугольная матрица L имеет единичную диагональ
 do k = 2, n                                 ! Прямые подстановки
   b(k) = b(k) + dot_product(fac(k, 1:k - 1), b(1:k - 1))
 end do
 x(n) = b(n) / fac(n, n)                     ! Выполняем обратный ход
 do k = n - 1, 1, -1                         ! (обратные подстановки)
   s = sum(fac(k, k + 1:n) * x(k + 1:n))
   x(k) = (b(k) - s) / fac(k, k)
 end do
end subroutine useLU
```

Результат:

```
                    x
   -2.000      0.000      1.000      -1.000
```

5.2.3. Повышение качества разложения

5.2.3.1. Схема частичного выбора

В схеме частичного выбора ведущего элемента на k-м шаге исключения уравнение с номером k меняется местами с уравнением, в котором a_{kk} имеет максимальное по модулю значение. Этот прием в большинстве случаев позволяет избежать переполнения и повысить точность результата.

Процесс обмена двух уравнений на шаге k равносилен умножению матрицы $A^{(k-1)}$ на матрицу перестановок P, отличающейся от единичной перестановкой обмениваемых строк. Например, умножение 4×4-матрицы A на матрицу

$$P=\begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix}$$

приведет к перестановке в A строк 2 и 3.

После перестановки формируется вектор Гаусса и полученная матрица умножается на матрицу преобразования Гаусса, поэтому

$$A^{(k)} = M^{(k)} P^{(k)} A^{(k-1)}$$

и

$$U = M^{(n-1)} P^{(n-1)} \ldots M^{(2)} P^{(2)} M^{(1)} P^{(1)} A ,$$

где $P^{(k)}$ – матрица перестановок на шаге k, а U – верхняя треугольная матрица.

Приведенное разложение можно записать в виде

$$\tilde{A} = PA = LU ,$$

где $P = P^{(n-1)} \ldots P^{(1)} A$, а L – нижняя унитреугольная матрица с элементами $|l_{ij}| \le 1$; k-й ее столбец ниже диагонали является перестановочной версией k-го вектора Гаусса, т. е.

$$L(k+1:n,k) = g(k+1:n) ,$$

где $g = P^{(n-1)} \ldots P^{(k+1)} \tau^{(k)}$.

Матрица перестановок P ортогональна, поэтому $P^{-1} = P^T$. Произведение матриц перестановок также является матрицей перестановок.

Цена частичного выбора невелика и составляет $O(n^2)$ операций, связанных с поиском ведущего элемента и перестановками в векторах Гаусса.

Вернемся к примеру, рассмотрение которого начато в разд. 5.2.1. После первого шага получена система уравнений

$$\begin{aligned}
2x_1 + 3x_2 + 11x_3 + 5x_4 &= 2,\\
-0{,}5x_2 - 0{,}5x_3 - 0{,}5x_4 &= 0,\\
-2x_2 - 8x_3 - 3x_4 &= -5,\\
-0{,}5x_2 - 2{,}5x_3 + 1{,}5x_4 &= -4.
\end{aligned}$$

В соответствии с идеей частичного выбора, чтобы снизить значения множителей очередного шага, на шаге k в качестве ведущего среди элементов $A^{(k-1)}(k{:}n,\ k)$ выбирается максимальный по модулю. Поскольку на шаге $k = 2$ такой элемент принадлежит уравнению 3 и он равен -2, то для продолжения исключений Гаусса уравнения 3 и 2 меняются местами. Легко проверить, что больше перестановок не потребуется. Тогда в рассматриваемом примере

$$P^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, P^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, P^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

и, следовательно,

$$\tilde{A} = P^{(3)} P^{(2)} P^{(1)} A = P^{(3)} P^{(2)} P^{(1)} \begin{pmatrix} 2 & 3 & 11 & 5 \\ 1 & 1 & 5 & 2 \\ 2 & 1 & 3 & 2 \\ 1 & 1 & 3 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 11 & 5 \\ 2 & 1 & 3 & 2 \\ 1 & 1 & 5 & 2 \\ 1 & 1 & 3 & 4 \end{pmatrix}.$$

Заметим, что для хранения информации о перестановках уравнений достаточно одномерного массив, например *ipvt*(1:*n*), *i*-й элемент которого содержит номер уравнения, которое меняется местами с уравнением *i*. Причем равенство *ipvt*(*i*) = *i* означает, что на шаге *k* = *i* - 1 перестановка уравнений не выполнялась. В нашем примере массив *ipvt* получит значения: *ipvt*(1) = 1, *ipvt*(2) = 3, *ipvt*(3) = 3, *ipvt*(4) = 4. Массив *ipvt* называется *вектором перестановок*.

Пример. Выполняется *LU*-разложение 4×4-матрицы. При разложении применяется схема частичного выбора. Результат возвращается матрицей *FAC*, информация о перестановках – вектором *ipvt*.

```
program LU3
 use msimsl, nouse => fac                    ! Для вызова WRRRN
 integer(4), parameter :: n = 4
 integer(4) :: ipvt(n)                       ! Вектор перестановок
 real(4) :: a(n, n), fac(n, n)               ! a – массив, представляющий матрицу A
 a = reshape((/      2,    3,    11,    5,                &
                     1,    1,     5,    2,                &
                     2,    1,     3,    2,                &
                     1,    1,     3,    4 /), shape = (/ n, n /), order = (/ 2, 1 /))
 call Lufac2(n, a, fac, ipvt)                ! Вычисляем LU-разложение матрицы A и ipvt
 print '(1x, a, 10i4)', 'ipvt =', ipvt       ! Вывод результата
 call wrrrn('fac', n, n, fac, n, 0)
end program LU3
subroutine Lufac2(n, a, fac, ipvt)           ! Возвращает LU-разложение матрицы A
 integer(4) :: n, ipvt(n), i, j, k           ! и вектор перестановок ipvt
 real(4) :: a(n, n), fac(n, n), p(n), hold
 fac = a; ipvt = (/ (i, i = 1, n) /)
 do k = 1, n - 1                             ! Прямой ход; k – номер ведущего уравнения
 ! Ищем строку с ведущим элементом и храним информацию о перестановках
 ipvt(k:k) = k + maxloc(abs(fac(k:n, k))) - 1
 if(ipvt(k) /= k) then                       ! Если строка не совпадает с номером шага,
  p(k:n) = fac(k, k:n)                       ! то выполняем перестановку строк k и ipvt(k)
```

```
  fac(k, k:n) = fac(ipvt(k), k:n); fac(ipvt(k), k:n) = p(k:n)
  ! Выполняем, если нужны, перестановки в векторах Гаусса
  ! и реализуем идею g = P^(n-1)... P^(k+1)τ^(k)
  do j = 1, k - 1
   hold = fac(ipvt(k), j); fac(ipvt(k), j) = fac(k, j); fac(k, j) = hold
  end do
 end if
 p(k + 1: n) = fac(k + 1:n, k) / fac(k, k)        ! Вычисляем множители k-го шага и формируем
 do i = k + 1, n                                  ! матрицу U; i – номер уравнения
  fac(i, k + 1:n) = fac(i, k + 1:n) - fac(k, k + 1:n) * p(i)
 end do
 fac(k + 1: n, k) = -p(k + 1: n)                  ! Формируем матрицу L
end do
end subroutine Lufac2
```

Результат:

```
ipvt =  1   3   3   4
```

fac

	1	2	3	4
1	2.00	3.00	11.00	5.00
2	-1.00	-2.00	-8.00	– 3.00
3	-0.50	-0.25	1.50	0.25
4	-0.50	-0.25	0.33	2.33

При этом

$$L=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0{,}5 & 0{,}25 & 1 & 0 \\ 0{,}5 & 0{,}25 & -1/3 & 1 \end{pmatrix}, U=\begin{pmatrix} 2 & 3 & 11 & 5 \\ 0 & -2 & -8 & -3 \\ 0 & 0 & 1{,}5 & 0{,}25 \\ 0 & 0 & 0 & 7/3 \end{pmatrix} \text{ и } \tilde{A}=PA=LU.$$

Линейные системы $Ax = b$ и $PAx = LUx = Pb$ эквивалентны, поэтому после нахождения матриц $P^{(k)}$, L и U решение системы $Ax = b$ определяется так:

1. Привести правую часть системы к виду $\tilde{b}=P^{(n-1)}P^{(n-2)}\ldots P^{(1)}b$.
2. Решить методом прямой подстановки систему $Ly=\tilde{b}$.
3. Решить верхнюю треугольную систему $Ux = y$ и найти вектор x, являющийся также и решением системы $Ax = b$.

Пример. Первоначально выполняется LU-разложение 4×4-матрицы A с применением схемы частичного выбора. Затем найденное разложение используется для решения линейной системы $Ax = b$.

```
program LU4
 use msimsl, nouse => fac                  ! Для вызова WRRRN
 integer(4), parameter :: n = 4
 integer(4) :: ipvt(n)                     ! Вектор перестановок
```

```
real(4) :: a(n, n), fac(n, n)                  ! a – массив, представляющий матрицу A
real(4) :: x(n), b(n) = (/ 2.0, 1.0, -3.0, -3.0 /)
a = reshape((/        2,     3,    11,     5,                &
                      1,     1,     5,     2,                &
                      2,     1,     3,     2,                &
                      1,     1,     3,     4 /), shape = (/ n, n /), order = (/ 2, 1 /))
call Lufac2(n, a, fac, ipvt)                   ! Возвращает LU-разложение матрицы A и ipvt
call useLU2(n, fac, ipvt, b, x)                ! Решаем линейную систему Ax = b
call wrrrn('x', 1, n, x, 1, 0)                 ! Вывод результата
end program LU4
subroutine Lufac2(n, a, fac)                   ! Текст подпрограммы приведен выше
...
end subroutine Lufac2
subroutine useLU2(n, fac, ipvt, b, x)          ! Решает Ax = b, учитывая выполненные при
 integer(4) :: n, ipvt(n), k                   ! поиске fac перестановки строк в A
 real(4) :: fac(n, n), x(n), b(n), s, hold
 do i = 1, n                                   ! Перестановки в векторе b; получаем
  if(ipvt(i) /= i) then                        ! b̃ = P^(n-1) P^(n-2) ... P^(1) b
   hold = b(ipvt(i)); b(ipvt(i)) = b(i); b(i) = hold
  end if
 end do
 do k = 2, n                                   ! Прямые подстановки
  b(k) = b(k) + dot_product(fac(k, 1:k - 1), b(1:k - 1))
 end do
 x(n) = b(n) / fac(n, n)                       ! Выполняем обратный ход
 do k = n - 1, 1, -1
   s = sum(fac(k, k + 1:n) * x(k + 1:n))
   x(k) = (b(k) - s) / fac(k, k)
 end do
end subroutine useLU2
```

Результат:

```
                    x
-2.000      0.000      1.000      -1.000
```

5.2.3.2. Полный выбор ведущего элемента

При частичном выборе на шаге k в качестве ведущего выбирался максимальный элемент текущего подстолбца $A^{(k-1)}(k{:}n,\ k)$. При полном выборе в позицию $(k,\ k)$ размещается максимальный элемент подматрицы $A^{(k-1)}(k,\ k)$. Стоимость полного выбора существенна: он требует $n^3/3$ операций на сравнение матричных элементов.

Стратегия полного выбора позволяет определить в процессе исключения Гаусса ранг матрицы, объявляя его равным k, если ведущий элемент на шаге $k + 1$ достаточно мал. Алгоритм исключения Гаусса с полным выбором ве-

дущего элемента является устойчивым. В то же время на практике, в том числе и в IMSL, чаще применяется более простая схема частичного выбора, обеспечивающая, как правило, не менее точные, чем при полном выборе, решения.

Замечание. При работе со *строго диагонально доминирующими* матрицами, в которых

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|, \; i=1, \ldots, n, \; A \in \mathbf{R}^{n\times n},$$

нет необходимости прибегать к полному или частичному выбору ведущего элемента.

5.2.3.3. Масштабирование

С целью повышения точности результата матрицы, нормы столбцов и строк которых чрезмерно различаются, подвергаются *масштабированию*, или *балансировке*. Например, если решается система [17]

$$\begin{pmatrix} 10 & 100\,000 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 100\,000 \\ 2 \end{pmatrix},$$

то ей эквивалентна строчно-масштабированная система

$$\begin{pmatrix} 0.0001 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Масштабирование выполняется диагональными матрицами D_1 и D_2. При этом решение исходной системы $Ax=b$ находится решением системы $\left(D_1^{-1}AD_2\right)y = D_1^{-1}b$, где $y = D_2^{-1}x$. Матрица D_1^{-1} выполняет масштабирование матрицы A по ее строкам, а D_2 – по столбцам. Матрицы D_1 и D_2 неизвестны. Цель масштабирования – найти матрицу $D_1^{-1}AD_2$, число обусловленности которой существенно меньше числа обусловленности матрицы A. Масштабирование можно выполнить практически без ошибок округления. Вычислительная сложность масштабирования $O(n^2)$.

В IMSL при необходимости применяется строчное масштабирование, при котором D_2 – единичная матрица, а матрица D_1 выбирается так, что каждая строка в матрице $D_1^{-1}A$ имеет примерно одинаковую ∞-норму. Заметим, что в ряде случаев такой подход может привести не к увеличению, а к снижению точности результата.

5.2.4. Устойчивость метода исключений Гаусса

Метод исключения без выбора ведущих элементов является неустойчивым – он может давать сколь угодно плохие результаты даже для хорошо обусловленных систем.

Если для характеристики устойчивости метода использовать *коэффициент роста*, равный максимальному относительному росту модуля элементов матрицы, то целью выбора ведущего элемента можно объявить ограничение этого коэффициента. В этом смысле при частичном выборе ведущего элемента предпринимаются шаги по ограничению коэффициента роста, и, как правило, они оказываются результативными: в большинстве случаев значение коэффициента роста меньше 10. Это говорит в пользу хорошей обусловленности такого подхода. В то же время верхняя оценка коэффициента роста, равная 2^{n-1} (n – размерность матрицы), указывает на возможность существования матриц, для которых схема частичного выбора может оказаться плохо обусловленной.

При полном выборе ведущего элемента коэффициент роста существенно ниже, чем в схеме частичного выбора. Верхняя оценка коэффициента роста равна $1.8n^{0.25\ln n}$. Более того, до сих пор не найдено матрицы, коэффициент роста которой при полном выборе больше n [1]. То есть алгоритм с полным выбором хорошо обусловлен и единственной причиной ошибок при решении линейной системы $Ax = b$ может стать плохая обусловленность матрицы A.

5.2.5. Некоторые приложения метода исключений Гаусса

5.2.5.1. Вычисление обратной матрицы

Пусть надо найти обратную матрицу $A^{-1} = (x_{ij})$ невырожденной $n\times n$-матрицы $A = (a_{ij})$. Воспользуемся соотношением $AA^{-1} = I$, где I – единичная матрица. Представим A^{-1} и I в блочно-столбцовом виде

$$A(x^{(1)},\ x^{(2)},\ \ldots,\ x^{(n)}) = (e^{(1)},\ e^{(2)},\ \ldots,\ e^{(n)}),\ A \in \mathbf{R}^{n\times n},$$

где $e^{(k)}$ – k-й канонический вектор. Отсюда для нахождения столбца k матрицы A^{-1} требуется решить систему

$$Ax^{(k)} = e^{(k)}, \tag{5.1}$$

где $x^{(k)} = \left(x_1^{(k)},\ x_2^{(k)},\ \ldots,\ x_n^{(k)}\right)^T$ – элементы k-го столбца обратной матрицы A^{-1}. Например, в случае 3×3-матрицы произведение AA^{-1} даст 3 системы линейных уравнений:

$$Ax^{(1)} = (1, 0, 0)^T,\ Ax^{(2)} = (0, 1, 0)^T,\ Ax^{(3)} = (0, 0, 1)^T.$$

Для решения систем вида (5.1) применим *алгоритм*:

1. Вычислить $PA = LU$.

2. С параметром $j = 1, n$ выполнить
   Решить $Ly = Pb^{(k)}$.
   Решить $Ux^{(k)} = y$.
   конец цикла 2.

Пример. Вычисляется A^{-1} – матрица, обратная 3×3-матрице A. Первоначально ищется LU-разложение A, которое затем используется для решения систем вида (5.1).

```
program fainv
 use msimsl, nouse => fac
 integer(4), parameter :: n = 3
 integer(4) :: j, ipvt(n)                          ! Вектор перестановок
 real(4) :: a(n, n), fac(n, n), e(n), ainv(n, n)   ! ainv – хранит матрицу A-1
 a = reshape((/      1.0,   3.0,   3.0,                &
                     1.0,   3.0,   4.0,                &
                     1.0,   4.0,   3.0 /), shape = (/ n, n /), order = (/ 2, 1 /))
 call Lufac2(n, a, fac, ipvt)                      ! Возвращает LU-разложение матрицы A и ipvt
 do j = 1, n                                       ! При каждом вызове useLU2 находится
  e= 0.0; e(j) = 1.0                               ! j-й столбец матрицы A-1
  call useLU2(n, fac, ipvt, e, ainv(1, j))         ! Решаем линейную систему Ax = e,
 end do                                            ! где x = ainv(1, j)
 call wrrrn('ainv', n, n, ainv, n, 0)              ! Вывод A-1
end program fainv                                  ! Текст подпрограмм Lufac2 и useLU2 см. выше
```

Результат:

ainv

	1	2	3
1	7.000	-3.000	-3.000
2	-1.000	0.000	1.000
3	-1.000	1.000	0.000

5.2.5.2. Вычисление определителя

Имея LU-разложение n×n-матрицы A, просто вычислить ее определитель. В самом деле, поскольку $PA = LU$, то определитель

$$\det A = \det(P^{-1}L)*\det U.$$

Определитель верхней треугольной матрицы U равен произведению своих диагональных элементов.

Для иллюстрации утверждения относительно равенства определителя треугольной матрицы произведению его диагональных элементов выполним разложение 4×4-определителя $\det U$ и получаемых миноров по первому столбцу:

$$\det U = \begin{vmatrix} 2 & -1 & 2 & 3 \\ 0 & 5 & 4 & 2 \\ 0 & 0 & 14 & 1 \\ 0 & 0 & 0 & 4 \end{vmatrix} = 2 \cdot \begin{vmatrix} 5 & 4 & 2 \\ 0 & 14 & 1 \\ 0 & 0 & 4 \end{vmatrix} + 0 + 0 + 0 = 2 \cdot 5 \cdot \begin{vmatrix} 14 & 1 \\ 0 & 4 \end{vmatrix} + 0 + 0 = 2 \cdot 5 \cdot 14 \cdot 4 .$$

Отсюда следует, что определитель треугольной матрицы с единичными диагональными элементами равен единице. Определитель матрицы $P^{-1}L$ равен $(-1)^k$, поскольку она получается из L в результате перестановок ее строк (здесь k – число выполненных во время LU-разложения перестановок). А известно, что если поменять местами две строки (столбца) определителя, то его значение по абсолютной величине не изменится, но знак поменяется на противоположный.

5.2.6. Число обусловленности матрицы

5.2.6.1. Понятие числа обусловленности

Число обусловленности $cond(A)$ матрицы A является мерой относительного расстояния от A до множества вырожденных матриц. По определению *числом обусловленности* называется отношение [10]

$$cond(A) = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} \Big/ \min_{x \neq 0} \frac{\|Ax\|}{\|x\|}, \tag{5.2}$$

в котором максимум и минимум берутся по всем ненулевым векторам x. В равенстве (5.2) числитель и знаменатель – соответственно максимально и минимально возможные относительные изменения нормы $\|Ax\|$ вектора Ax по сравнению с нормой $\|x\|$ вектора x. Если продолжить приведенную в разд. 4.2.15 геометрическую интерпретации, то число обусловленности есть отношение максимального и минимального коэффициентов растяжения векторов x под действием матрицы A.

Запись (5.2) эквивалентна формуле

$$cond(A) = \|A\| \, \|A^{-1}\|,$$

где $\|A\|$ – норма матрицы A.

Для любой p-нормы $cond_p(A) \geq 1$. Матрицы с большим числом обусловленности называются *плохо обусловленными*, и наоборот, матрицы с малыми значениями $cond(A)$ называются *хорошо обусловленными*.

Некоторые оценки числа обусловленности:

- в 2-норме для ортогональной матрицы Q справедливо $cond_2(Q) = 1$;
- если P – матрица перестановок, то $cond(P) = 1$;
- если $D = \mathrm{diag}(d_1, d_2, ..., d_n)$, то $cond(D) = \max(d_i)/\min(d_i)$, $i = 1, 2, ..., n$;
- число обусловленности в 2-норме матрицы общего вида

$$cond_2(A) = \| A \|_2 \| A^{-1} \|_2 = \frac{\sigma_1(A)}{\sigma_n(A)}, \; A \in \mathbf{R}^{m \times n}, \; n \le m,$$

где $\sigma_1(A)$ и $\sigma_n(A)$ – сингулярные числа матрицы A, причем $\sigma_1 \ge \sigma_2 \ge \ldots \sigma_n > 0$.

В задаче решения системы линейных уравнений число обусловленности $cond_\infty$ связано с относительной ошибкой результата:

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \approx \varepsilon_м cond_\infty(A) \quad , \tag{5.3}$$

где x и $\hat{x}$ – соответственно точное и найденное решения, а $\varepsilon_м$ – машинная точность. Приведенная оценка говорит, что в плохо обусловленных системах относительная ошибка решения может быть весьма большой.

В процедурах библиотеки IMSL при вычислении $cond(A)$ используется 1- норма $\|A\|_1 = \max\limits_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|$, $A \in \mathbf{R}^{n \times n}$, поэтому обозначение числа обусловленности содержит индекс 1: $cond_1(A)$. Поскольку вычисление обратной матрицы A^{-1}, и следовательно $\|A^{-1}\|$, сопряжено с большими временными затратами, в процедурах библиотеки IMSL точное значение числа обусловленности не вычисляется, а дается его оценка, позволяющая достаточно достоверно охарактеризовать обусловленность задачи.

Число обусловленности матрицы $cond(A)$ характеризует обусловленность линейной системы $Ax = b$. Для плохо обусловленной системы линейных уравнений $Ax = b$, когда незначительные изменения A влекут серьезные изменения решения x, число обусловленности велико. Если найденное значение $cond_1(A) > 1/\varepsilon_м$, то процедуры IMSL, решающие системы линейных уравнений и оценивающие число обусловленности, выдают предупреждение.

5.2.6.2. Оценка числа обусловленности матрицы

Вычисление $\|A\|_1$ – первого сомножителя $cond_1(A)$ не составляет затруднений. Поэтому оценке подлежит значение $\|A^{-1}\|_1$. Идея оценки $\|A^{-1}\|$ изложена в [31]. Из определения числа обусловленности матрицы

$$\|A^{-1}\| = 1/\min_x \frac{\|Ax\|}{\|x\|} = \max_x \frac{\|x\|}{\|Ax\|} = \max_y \frac{\|A^{-1}y\|}{\|y\|}.$$

Последнее отношение получается после замены переменных $y = Ax$.

Рассмотрим линейную систему $Az = y$ и найдем для некоторого y значение $z = A^{-1}y$. Примем далее в качестве оценки нормы

$$\|A^{-1}\| \approx \|z\|_1 / \|y\|_1 = \|A^{-1}y\|_1 / \|y\|_1 .$$

Понятно, что выбор y не может быть случайным, а должен для хорошей оценки $\|A^{-1}\|$ обеспечивать хорошее приближение к максимуму выражения $\|z\|_1/\|y\|_1$.

Если известно LU-разложение матрицы A ($PA = LU$), то, чтобы снизить временные затраты, поиск максимального по величине отношения $\|z\|_1/\|y\|_1$ первоначально осуществляется для верхней треугольной матрицы U. Такая замена основана на эвристическом соображении: если A является плохо обусловленной, то и матрица U также плохо обусловлена, а нижняя унитреугольная матрица L обусловлена достаточно хорошо, поэтому целесообразнее оценивать обусловленность U, а не L.

Итак, вместо $Az = y$ решается $Uz = y$. Для решения используется столбцовая версия обратных подстановок, в которой на каждой k-й итерации выбирается значение $y(k)$, такое, что в итоге величина $\|z\|_1/\|y\|_1$ оказывается близкой к максимальной.

```
p = 0.0
do k = n, 1, -1
 <Выбрать y(k)>
 z(k) = (y(k) - p(k)) / U(k, k)
 p(1:k - 1) = p(1:k - 1) + z(k) * U(1:k - 1, k)
end do
```

В приведенном варианте обратных подстановок, как и полагается (см. подпрограмму *useLU* в разд. 5.2.2.2), $p(k)$ = SUM($U(k, k+1{:}n)$ * $x(k+1{:}n)$), но накопление этой суммы осуществляется иначе.

В известных реализациях алгоритма оценки $\|U^{-1}\|_1$ на каждом шаге обратной подстановки $y(k)$ выбирается равным +1 или -1. В этом случае $\|y\|_1 = n$, где n – размер вектора y. Поэтому максимизация отношения $\|z\|_1/\|y\|_1$ сводится к максимизации $\|z\|_1$.

Алгоритм выбора $y(k)$ может быть таким:

- вычислить $zp(k) = (+1 - p(k))/U(k, k)$ (здесь $y(k) = +1$);
- вычислить $zm(k) = (-1 - p(k))/U(k, k)$ (здесь $y(k) = -1$);
- выбрать $z(k) = \max(zp(k), zm(k))$.

Однако в работе [31] показано, что $\|z\|_1$ ближе приближается к возможному максимальному значению, если на каждом шаге обеспечивать рост текущей суммы, задаваемой соотношением

$$s = |z_k| + \|p_{1:k-1} + z_k * U_{1:k-1,k}\|_1.$$

Пример. Выполняется оценка $estU$ нормы $\|U^{-1}\|_1$, а затем вычисляются $\|U\|_1$ и оценка $cond_1(U) \approx \|U\|_1\ estU$.

```
program cond
 use msimsl
 integer, parameter :: n = 4
 real(4) :: U(n, n)                          ! z – искомый вектор
 real(4) :: fest, estU                       ! estU – оценка 1-нормы матрицы U^-1
 real(4) :: condU                            ! Оценка числа обусловленности матрицы U
 U = reshape((/      2.0,   3.0,  11.0,   5.0,          &
                     0.0,  -2.0,  -8.0,  -3.0,          &
                     0.0,   0.0,   1.5,  0.25,          &
                     0.0,   0.0,   0.0,  2.33 /), shape = (/ n, n /), order = (/ 2, 1 /))
 estU = fest(n, U)                           ! Находим оценку 1-нормы матрицы U^-1
 ! Оценка числа обусловленности cond1(U) ≈ condU = ||U||1 estU
 condU = maxval(sum(abs(U), dim = 1)) * estU
 print *, 'condU = ', condU
end program cond
function fest(n, U)                          ! Вычисляет оценку 1-нормы матрицы U^-1
 integer(4) :: n, k
 real(4) :: U(n, n), z(n), p(n), pm(n)       ! z – искомый вектор в системе Uz = y
 real(4) :: zp, zm, sp, sm
 ! Решаем, выполняя обратные подстановки, Uz = y, выбирая y(k) = +1 или -1
 p = 0.0
 do k = n, 1, -1
  zp = (+1 - p(k)) / U(k, k)                 ! y(k) = +1
  zm = (-1 - p(k)) / U(k, k)                 ! y(k) = -1
  p(1:k - 1) = p(1:k - 1) + zp * U(1:k - 1, k)
  pm(1:k - 1) = p(1:k - 1) + zm * U(1:k - 1, k)
  ! Находим s = |zk| + ||p1:k-1 + zk * U1:k-1,k||1 для zk = zp и zk = zm
  sp = abs(zp) + sum(abs(p(1:k - 1)))
  sm = abs(zm) + sum(abs(pm(1:k - 1)))
  if(sp > sm) then
   z(k) = zp
  else
   z(k) = zm; p(1:k - 1) = pm(1:k - 1)
  end if
 end do
 fest = sum(abs(z)) / float(n)               ! Оценка 1-нормы матрицы U^-1
end function fest
```

Результат:

```
condU =      44.973890
```

Оценка числа обусловленности матрицы общего вида, для которой известно ее разложение $PA = LU$, выполняется по алгоритму, в основе которого лежит оценка числа обусловленности U^{-T} [6]:

- получить наибольшее по норме решение системы $U^T z = y$ (находится по схеме, примененной выше для системы $Uz = y$);
- решить треугольные системы $L^T r = z$, $Lw = Pr$ и $Uy = w$;
- принять в качестве оценки $cond_1(A) \approx \|A\|_1 \, \|y\|_1 / \|r\|_1$.

Алгоритм оценки числа обусловленности требует $O(n^2)$ флопов.

5.2.7. Оценка точности и уточнение решения

5.2.7.1. Точность решения

Исследование ошибок метода исключения Гаусса показало, что вычисленное решение $\hat{x}$ системы $Ax = b$ точно удовлетворяет системе

$$(A+E)\hat{x} = b, \ \|E\|_\infty \approx \varepsilon_м \|A\|_\infty . \tag{5.4}$$

(Результат принадлежит Дж. Х. Уилкинсону.)

Из соотношения (5.4) имеем $\| b - A\hat{x} \|_\infty \approx \varepsilon_м \| A \|_\infty \| \hat{x} \|_\infty$. Отсюда следует, что метод исключения Гаусса дает решение $\hat{x}$ с маленькой относительной невязкой. В то же время из (5.3) следует, что маленькие относительные невязки не говорят о высокой точности: в плохо обусловленных системах ошибка решения может быть весьма большой.

Соотношение (5.3) позволяет заключить: если машинная точность $\varepsilon_м \approx 10^{-d}$ и $cond_\infty \approx 10^{-q}$, то исключение Гаусса дает решение с $d - q$ правильными десятичными знаками. Точность решения можно повысить, выполнив его итерационное уточнение.

5.2.7.2. Итерационное уточнение решения

Для достижения большей точности применяется *итерационное уточнение* решения, позволяющее при сравнительно небольшом числе обусловленности матрицы A выполнить вычисления с относительной точностью, сравнимой с $\varepsilon_м$.

Пусть x и $\hat{x}$ – соответственно точное и приближенное решения системы линейных уравнений. Положим

$$x = \hat{x} + \delta ,$$

где $\delta = (\delta_1, \delta_2, ..., \delta_n)^T$ – поправка решения $\hat{x}$, и вычислим невязку $r = (r_1, r_2, ..., r_n)^T$ для приближенного решения $\hat{x}$: $r = b - A\hat{x}$. Тогда имеем $A(x + \delta) = b$, или $A\delta = r$.

То есть, чтобы найти уточнение δ, следует решить систему $A\delta = r$ с прежней матрицей A. Для ее решения используется ранее найденное LU-разложение матрицы A ($PA = LU$), чем и обусловлены сравнительно небольшие временные затраты на итерационное уточнение.

Алгоритм итерационного уточнения.

1. Начало.
2. Задать *kmax* – предельно допустимое число итераций и ε – точность решения.
3. $x = \hat{x}$.
4. С параметром $k = 1$, *kmax* выполнить:

Вычислить невязку $r = b - Ax$ (двойная точность).

Решить $Ly = Pr$ относительно y.

Решить $U\delta = y$ относительно δ.

Найти уточненное решение $x = x + \delta$.

Если $\delta_i < \varepsilon$, $i = 1, 2, ..., n$, то выйти из цикла 4.

конец цикла 4.

5. Конец.

Замечания:

1. Каждое уточнение стоит $O(n^2)$ флопов, что существенно меньше затрат на LU-разложения матрицы, равных $2n^3/3$ флопов.
2. Вычисление невязки $r = b - Ax$ следует выполнять с двойной точностью.
3. Нельзя контролировать точность решения по значению невязки, которая в общем случае по мере уточнений может и увеличиваться.
4. Если машинная точность $\varepsilon_M \approx 10^{-d}$ и $cond_\infty \approx 10^{-q}$, то через k итераций уточненное решение имеет приблизительно $\min(d, k(d - q))$ правильных десятичных знаков.
5. Погрешности вычислений при использовании итерационных методов не накапливаются.
6. При больших значениях числа обусловленности матрицы система оказывается очень чувствительной к незначительным изменениям входных данных, что может привести к снижению в процессе итераций точности решения. Поэтому, выполняя итерационное уточнение корней, IMSL, обнаружив плохо обусловленную матрицу, выдаст сообщение:

```
*** WARNING      ERROR 2 from LSARG. The matrix is too ill-conditioned for
***              iterative refinement to be effective.
```

5.2.8. Системы с разреженными матрицами

Системы линейных уравнений с разреженными матрицами возникают, в частности, при аппроксимации дифференциальных уравнений конечно-разностными или вариационно-разностными уравнениями.

При решении уравнений с разреженными матрицами в памяти размещаются только их ненулевые элементы. Применим для решений и метод исключения Гаусса: так же, как и для систем с матрицами общего вида, первоначально выполняется LU-разложение матрицы и система представляется

в виде $PAQ = LU$, где P и Q – соответственно матрицы перестановок в рядах и столбцах матрицы A в процессе ее LU-разложения.

При LU-разложении матрицы A часто промежуточные матрицы оказываются более заполненными по сравнению с исходной матрицей. Причем рост степени заполнения зависит от выбранной стратегии LU-разложения, состоящей в выборе ведущего уравнения и ведущего элемента. Поэтому задачей алгоритмов, реализующих метод исключения Гаусса, является сохранить высокую степень разреженности матрицы. Решая эту задачу, надо также ограничивать и *коэффициент роста* – показатель, вычисляемый как отношение наибольшего по абсолютной величине элемента матрицы A на всех этапах LU-разложение к наибольшему по абсолютной величине элементу в первоначальной матрице A. Большое значение коэффициента роста означает, что решение может содержать ощутимую ошибку.

Итак, методы, выполняющие LU-разложение разреженных матриц, должны удовлетворять по крайней мере двум требованиям:

- промежуточные и результирующая матрицы должны оставаться разреженными;
- коэффициент роста должен минимизироваться и не должен превышать некоторого допустимого значения.

Алгоритмы, оптимизирующие решение по этим критериям, являются эвристическими и обеспечивают, как правило, хорошее, но неоптимальное переупорядочивание уравнений.

5.3. ЛИНЕЙНЫЕ СИСТЕМЫ С СИММЕТРИЧЕСКОЙ МАТРИЦЕЙ

Для симметрических матриц разработаны методы, позволяющие получить их представление в виде произведения матриц более простого вида, например треугольных, в 2 раза быстрее, чем метод исключения Гаусса. Имеющиеся в библиотеке IMSL процедуры предназначены для разного вида симметрических матриц: положительно определенных, неотрицательно определенных и неопределенных. Причем часть из них, например CHFAC, может работать и с алгоритмически вырожденными симметрическими матрицами.

5.3.1. Положительно определенные системы

Матрица $A \in \mathbf{R}^{n\times n}$ называется *положительно определенной*, если квадратичная форма $f = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j = x^T Ax > 0$ для всех ненулевых векторов $x = (x_1, x_2, \ldots, x_n)^T$, $x_i \in \mathbf{R}$. В линейной алгебре доказывается, что вещест-

венная матрица тогда и только тогда будет положительно определенной, если все ее главные диагональные миноры положительны. Системы линейных уравнений с положительно определенными матрицами также называются положительно определенными. Из определения положительной определенности матрицы следует, что она не может быть вырожденной.

Для вещественной положительно определенной матрицы A справедливо:

- наибольший элемент матрицы лежит на ее диагонали, и он больше нуля;
- все ее главные подматрицы являются положительно определенными;
- существует ее разложение $A = LDM^T$, где L и M^T – соответственно нижняя и верхняя унитреугольные матрицы, а диагональная матрица $D = \mathrm{diag}(d_1, d_2, ..., d_n)$ имеет положительные элементы.

Свойством положительной определенности могут обладать прямоугольные матрицы общего вида. В то же время дополнительно о симметрической положительно определенной матрице $A \in \mathbf{R}^{n\times n}$ можно сообщить:

- существует ее разложение $A = LL^T$, называемое *разложением Холецкого*, где $L \in \mathbf{R}^{n\times n}$ – нижняя треугольная матрица с положительными диагональными элементами. Матрицы L и L^T называются *треугольниками Холецкого*;
- все ее собственные значения положительны.

Типичные ситуации, вызывающие положительную определенность матрицы $A \in \mathbf{R}^{n\times n}$:

- $A = XX^T$, где X имеет полный столбцовый ранг;
- матрицы A и A^T диагонально доминирующие, и все $a_{ii} > 0$;
- матрица A является матрицей квадратичной формы, положительность которой вытекает из ее физического смысла.

Замечания:

1. Аналогичным образом формулируется понятие *положительно определенной* эрмитовой матрицы. Ее эрмитова форма $f = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i\bar{x}_j > 0$ для всех ненулевых векторов $x = (x_1, x_2, ..., x_n)^T$, $x_i \in \mathbf{C}$. В форме f коэффициенты при неизвестных связаны равенством $a_{ij} = \bar{a}_{ji}$.
2. Можно также ввести понятие *отрицательно определенной* невырожденной матрицы, т. е. матрицы, квадратичная форма которой всегда отрицательна.

5.3.2. Получение разложения Холецкого

Если в системе линейных уравнений $Ax = b$ матрица $A \in \mathbf{R}^{n\times n}$ является симметрической и положительно определенной, то для ее решения следует

использовать разложение Холецкого, которое требует в 2 раза меньше вычислительных затрат, чем LU-разложение, получаемое по методу Гаусса.

Разложение Холецкого предусматривает представление матрицы A в виде произведения LL^T, в котором нижняя треугольная матрица

$$L = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}$$

имеет на главной диагонали положительные элементы.

Замечание. С таким же успехом по методу Холецкого матрицу A можно представить в виде произведения R^TR, где R – верхняя треугольная матрица.

После нахождения разложения $A = LL^T$ вычисление корней линейной системы $Ax = b$ сводится к последовательному решению двух систем с треугольными матрицами:

$$Ly = b \text{ и } L^Tx = y.$$

Найдем, чему равны элементы матрицы L. Вычислим элементы LL^T и, приравняв их к соответствующим элементам матрицы A, получим систему уравнений

$$\begin{aligned}
&l_{11}^2 = a_{11}, \\
&l_{i1}l_{11} = a_{i1}, \quad i = 2, 3, \dots, n, \\
&l_{21}^2 + l_{22}^2 = a_{22}, \\
&l_{i1}l_{21} + l_{i2}l_{22} = a_{i2}, \quad i = 3, 4, \dots, n, \\
&\dots\dots\dots\dots\dots\dots\dots\dots \\
&l_{j1}^2 + l_{j2}^2 + \dots + l_{jj}^2 = a_{jj}, \\
&l_{i1}l_{j1} + l_{i2}l_{j2} + \dots + l_{ij}l_{jj} = a_{i2}, \quad i = j+1, \dots, n, \\
&\dots\dots\dots\dots\dots\dots\dots\dots \\
&l_{n1}^2 + l_{n2}^2 + \dots + l_{nn}^2 = a_{nn}.
\end{aligned}$$

Решение системы очевидно:

$$\begin{aligned}
&l_{11} = \sqrt{a_{11}}, \\
&l_{i1} = a_{i1}/l_{11}, \quad i = 2, 3, \dots, n, \\
&l_{22} = \sqrt{a_{22} - l_{21}^2}, \\
&l_{i2} = (a_{i2} - l_{i1}l_{21})/l_{22}, \quad i = 3, 4, \dots, n, \\
&\dots\dots\dots\dots\dots\dots\dots\dots \\
&l_{jj} = \sqrt{a_{jj} - l_{j1}^2 - l_{j2}^2 - \dots - l_{j,j-1}^2}, \qquad (5.5)
\end{aligned}$$

$$l_{ij} = (a_{ij} - l_{i1}l_{j1} - l_{i2}l_{j2} - \ldots - l_{i,j-1}l_{j,j-1})/l_{jj}, \quad i = j+1, \ldots, n,$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$l_{nn} = \sqrt{a_{nn} - l_{n1}^2 - l_{n2}^2 - \ldots - l_{n,n-1}^2}.$$

Замечание. В случае симметрической положительно определенной матрицы A процесс Холецкого, как правило, может быть доведен до конца, т. е. при его реализации не возникают квадратные корни из отрицательных чисел. Разложение Холецкого может быть выполнено, если $q_n \varepsilon_м cond_2(A) \le 1$, где q_n – маленькая константа [71]. Таким образом, если процесс Холецкого удается завершить, то матрица, для которой ищется разложение, является положительно определенной.

Пример. Решается система линейных уравнений

$$\begin{aligned} x_1 - 3x_2 + 2x_3 &= 2, \\ -3x_1 + 10x_2 - 5x_3 &= 1, \\ 2x_1 - 5x_2 + 6x_3 &= -3 \end{aligned}$$

с симметрической положительно определенной матрицей. Первоначально ищется ее разложение по Холецкому. Затем при помощи подпрограммы *stback* вычисляются ее корни.

```
program ch
 use msimsl
 integer(4), parameter :: n = 3
 real(4) :: a(n, n), L(n, n), x(n), b(n)
 integer(4) :: j
 ! Матрица A должна быть симметрической и положительно определенной
  a = reshape((/      1.0,  -3.0,   2.0,                  &
                     -3.0,  10.0,  -5.0,                  &
                      2.0,  -5.0,   6.0 /), shape = (/ n, n /))
 b = (/ 2.0, 1.0, -3.0 /)
 call chol(n, a, L)                        ! Возвращает треугольник Холецкого L
 call wrrrn('L', n, n, L, n, -1)           ! Выводим нижнюю треугольную часть L
 do j = 1, n                               ! Формируем матрицу stback
  L(j, j + 1:) = L(j + 1:, j);
  L(j + 1:, j) = -L(j + 1:, j)
 end do
 call stback(n, L, b, x)                   ! Решаем линейную систему Ax = b
 call wrrrn('x', 1, n, x, 1, 0)            ! Вывод результата
end program ch
subroutine chol(n, a, L)                   ! Вычисляет треугольник Холецкого -
 integer(4) :: n, i, k                     ! матрицу L
 real(4) :: a(n, n), L(n, n)
 L = a                                     ! Подготовка к разложению
```

```
 do k = 1, n                                  ! k – шаг разложения
  ! Вычисляем элемент на диагонали матрицы L
  L(k, k) = sqrt(L(k, k) - sum(L(k, :k - 1)**2))
  ! Модифицируем столбец k матрицы L начиная со строки k + 1
  do i = k + 1, n                             ! i – номер строки массива L
     L(i, k) = (L(i, k) - sum(L(i, :k - 1) * L(k, :k - 1))) / L(k, k)
  end do
 end do
end subroutine chol

subroutine stback(n, fac, b, x)
 ! Решает Ax = b, выполняя сначала прямые, а затем обратные подстановки
 ! В отличие от ранее приведенной подпрограммы useLU здесь нижняя
 ! треугольная матрица не является в общем случае унитреугольной
 integer(4) :: n, k
 real(4) :: fac(n, n), x(n), b(n), s
 b(1) = b(1) / fac(1, 1)                     ! Прямые подстановки
 do k = 2, n
   b(k) = (b(k) + dot_product(fac(k, 1:k - 1), b(1:k - 1))) / fac(k, k)
 end do
 x(n) = b(n) / fac(n, n)                     ! Выполняем обратный ход
 do k = n - 1, 1, -1                         ! (обратные подстановки)
   s = sum(fac(k, k + 1:n) * x(k + 1:n))
   x(k) = (b(k) - s) / fac(k, k)
 end do
end subroutine stback
```

Результат:

```
                    L
                1          2          3
 1          1.000
 2         -3.000      1.000
 3          2.000      1.000      1.000

                x
 93.00      21.00     -14.00
```

5.3.3. Неотрицательно определенные системы

5.3.3.1. Свойства неотрицательно определенных матриц

Матрица A является *неотрицательно определенной*, если квадратичная форма $x^T Ax \geq 0$. Элементы неотрицательно определенной симметрической матрицы обладают свойствами [6]:

$$
\begin{aligned}
&|a_{ij}| \le (a_{ii} + a_{jj})/2, \\
&|a_{ij}| \le \sqrt{a_{ii}a_{jj}}, \quad (i \ne j), \\
&\max_{i,j}|a_{ij}| = \max_{i}|a_{ii}|, \\
&a_{ii} = 0 \Rightarrow A(i,:) = 0, A(:,i) = 0.
\end{aligned}
\tag{5.6}
$$

5.3.3.2. LL^T-разложение симметрических неотрицательно определенных матриц

В случае неотрицательно определенной системы один или несколько диагональных элементов разложения Холецкого равны нулю. Такой элемент возникает, когда подкоренное выражение равенства (5.5) меньше некоторого допуска *tol*. Тогда версия подпрограммы, выполняющей LL^T-разложение Холецкого неотрицательно определенной матрицы, может быть такой:

```
subroutine chol2(n, a, tol, L, irank)          ! Вычисляет треугольник Холецкого -
 integer(4) :: n, irank, k, i                  ! матрицу L и ранг irank матрицы A
 real(4) :: a(n, n), L(n, n), tol, temp
 L = a                                         ! Подготовка к вычислениям
 irank = 0                                     ! irank – ранг матрицы A
 do k = 1, n                                   ! k – номер столбца массива a
  ! Вычисляем элемент на диагонали матрицы L
   temp = L(k, k) - sum(L(k, :k - 1)**2)
  if(temp > tol) then
   irank = irank + 1
   L(k, k) = sqrt(temp)
   ! Модифицируем столбец k матрицы L начиная со строки k + 1
   do i = k + 1, n                             ! i – номер строки массива L
    L(i, k) = (L(i, k) - sum(L(i, :k - 1) * L(k, :k - 1))) / L(k, k)
   end do
   else
    L(k:, k) = 0.0                             ! В соответствии с (5.6)
   end if
 end do
end subroutine chol2
```

Пример. Вычисляется нижняя треугольная матрица L неотрицательно определенной матрицы A, такая, что $A = LL^T$.

```
program ch2
 use msimsl
 integer(4), parameter :: n = 5
 real(4) :: a(n, n), L(n, n), tol
 integer(4) :: irank                           ! irank – ранг матрицы A
 ! Матрица A должна быть симметрической и положительно
 ! или неотрицательно определенной
```

```
a = reshape((/    36.0,  12.0,  30.0,   6.0,  18.0,                &
                  12.0,  20.0,   2.0,  10.0,  22.0,                &
                  30.0,   2.0,  29.0,   1.0,   7.0,                &
                   6.0,  10.0,   1.0,  14.0,  20.0,                &
                  18.0,  22.0,   7.0,  20.0,  40.0 /), shape = (/ n, n /))
! Допуск, используемый для определения малости очередного диагонального элемента L
tol = 0.001
! Возвращает треугольник Холецкого L и irank – ранг матрицы A
call chol2(n, a, tol, L, irank)
print '(1x, a, i3)', 'rank = ', irank
call wrrrn('L', n, n, L, n, -1)                ! Выводим нижнюю треугольную часть L
! Контрольный вывод произведения LL^T. Оно совпадает с матрицей A
do i = 1, n; L(i, i + 1:) = 0; end do          ! Подготовка матрицы L
call wrrrn('LLT', n, n, matmul(L, transpose(L)), n, 0)
end program ch2
```

Результат:

```
rank =  4
                          L
          1          2          3          4          5
  1   6.000
  2   2.000      4.000
  3   5.000     -2.000      0.000
  4   1.000      2.000      0.000      3.000
  5   3.000      4.000      0.000      3.000      2.449
```

Для неотрицательно определенной симметрической матрицы процедуры библиотеки IMSL, например CHFAC или LCHRG, также вычисляют разложение Холецкого. Причем в LCHRG выполняется симметричный выбор диагонального элемента.

5.3.3.3. Симметричный выбор диагонального элемента

В случае строчных ($A \leftarrow PA$) или столбцовых ($A \leftarrow AP$) перестановок симметрия нарушается. Симметрию сохраняет перестановка $A \leftarrow PAP^T$; она не переставляет внедиагональные элементы на диагональ.

Алгоритм перестановок для шага k:

1. Найти km ($k \le km \le n$), такой, что $A(km, km) = \max(A(k{:}n, k{:}n))$.
2. Если $km \ne k$, то
   $ipvt(k) = km$! Вектор перестановок
   ! Симметричная перестановка
   ! Перестановка строк k и km; $hold$ – вектор размера n
   $hold = A(k, :)$; $A(k, :) = A(km, :)$; $A(km, :) = hold$
   ! Перестановка столбцов k и km

$hold = A(:, k); A(:, k) = A(:, km); A(:, km) = hold$

конец если 2.

3. Выполнить очередной шаг разложения.

Алгоритм вычисляет матрицу перестановок P, хранимую вектором *ipvt*.

Пример. Подпрограммой *chol*3 возвращаются нижняя треугольная матрица L неотрицательно определенной матрицы A и матрица перестановок P, хранящаяся в векторе *ipvt*, такие, что $PAP^T = LL^T$. Также *chol*3 возвращает ранг *irank* входной матрицы A.

```
program ch3
 use msimsl
 integer(4), parameter :: n = 5
 real(4) :: a(n, n), L(n, n), tol
 integer(4) :: ipvt(n), irank                    ! irank – ранг матрицы A
! Матрица A должна быть симметрической и положительно
! или неотрицательно определенной
a = reshape((/     36.0,  12.0,  30.0,   6.0,  18.0,                &
                   12.0,  20.0,   2.0,  10.0,  22.0,                &
                   30.0,   2.0,  29.0,   1.0,   7.0,                &
                    6.0,  10.0,   1.0,  14.0,  20.0,                &
                   18.0,  22.0,   7.0,  20.0,   40.0 /), shape = (/ n, n /))
! Допуск, используемый для определения малости очередного диагонального элемента
tol = 0.001
! Возвращает треугольник Холецкого L, вектор перестановок ipvt и irank – ранг матрицы A
call chol3(n, a, tol, L, ipvt, irank)
print '(1x, a, i3)', 'rank = ', irank; print '(1x, a, 10i3)', 'ipvt: ', ipvt
call wrrrn('L', n, n, L, n, -1)                  ! Выводим нижнюю треугольную часть L
! Контрольный вывод произведения LL^T. Оно совпадает с матрицей PAP^T
 call check(n, a, L, ipvt)
end program ch3
subroutine chol3(n, a, tol, L, ipvt, irank)      ! Вычисляет треугольник Холецкого -
 integer(4) :: n, ipvt(n), irank, k, km, i       ! матрицу L, вектор перестановок ipvt
 real(4) :: a(n, n), L(n, n), tol, hold(n)       ! и ранг матрицы A
 ipvt = (/ (i, i = 1, n) /)                      ! Инициализация вектора перестановок
 L = a                                           ! Подготовка к вычислениям
 irank = 0                                       ! irank – ранг матрицы A
 do k = 1, n                                     ! k – номер столбца массива a
! km – номер столбца с максимальным диагональным элементом
  km = k
  do i = k + 1, n; if(L(i, i) > L(km, km)) km = i; end do
  if(km /= k) then                               ! Симметричные перестановки
    ipvt(k) = km
    hold = L(k, :); L(k, :) = L(km, :); L(km, :) = hold
    hold = L(:, k); L(:, k) = L(:, km); L(:, km) = hold
```

```
  end if
  temp = L(k, k) - sum(L(k, :k - 1)**2)        ! Вычисляем элемент на диагонали матрицы L
  if(temp > tol) then
   irank = irank + 1
   L(k, k) = sqrt(temp)
   ! Модифицируем столбец k матрицы L начиная со строки k + 1
   do i = k + 1, n                             ! i – номер строки массива L
    L(i, k) = (L(i, k) - sum(L(i, :k - 1) * L(k, :k - 1))) / L(k, k)
   end do
  else
   L(k:, k) = 0.0                              ! В соответствии с (5.6)
  end if
 end do
end subroutine chol3
subroutine check(n, a, L, ipvt)
 real(4) :: a(n, n), L(n, n), a2(n, n)
 integer(4) :: ipvt(n), k, i, j                ! irank – ранг матрицы A
 ! Вспомогательные массивы для организации контрольного вывода произведения LL^T
 real(4) :: pk(n, n), p(n, n), hold(n)
 a2 = a                                        ! Сохраняем массив a
 ! Контрольный вывод произведения LL^T. Оно совпадает с матрицей PAP^T
 do i = 1, n; L(i, i + 1:) = 0; end do         ! Подготовка матрицы L
 do i = 1, n; p(i, i) = 1.0; end do            ! Инициализация матрицы перестановок
 do k = 1, n                                   ! В этом цикле формируется матрица перестановок
  pk = 0.0                                     ! P = P_r ... P_1
  do j = 1, n; pk(j, j) = 1.0; end do          ! Формирование P_k
  j = ipvt(k)
  hold = pk(k, :); pk(k, :) = pk(j, :); pk(j, :) = hold
  p = matmul(pk, p)
 end do
 a2 = matmul(p, a2)                            ! Вычисляем PA
 a2 = matmul(a2, transpose(p))                 ! Вычисляем PAP^T
 call wrrrn('PAPT', n, n, a2, n, 0)
 call wrrrn('LLT', n, n, matmul(L, transpose(L)), n, 0)
end subroutine check
```

Результат:

```
rank =   4
ipvt:  5  5  3  5  5
```

L

	1	2	3	4	5
1	6.325				
2	2.846	5.282			
3	1.107	5.083	1.391		

4	3.479	0.398	-2.782	0.000	
5	3.162	-0.568	0.278	0.000	1.897

Замечание. Проблема выбора диагональных элементов рассматривается, например, в работе [53].

5.3.4. Симметрические неопределенные системы

Матрица A является *неопределенной*, если ее квадратичная форма x^TAx принимает и положительные и отрицательные значения. Разложение Холецкого таких матриц невыполнимо – диагональные элементы, из которых при разложении извлекаются квадратные корни, могут быть меньше нуля.

Неопределенные матрицы подвергаются LDL^T- или UDU^T-разложению, в котором L (U) – нижняя (верхняя) унитреугольная матрица, D – диагональная матрица, состоящая из блоков размера 1×1 или 2×2. (Далее рассматривается разложение LDL^T.) Применяемый для разложения *метод с диагональным выбором элементов* вычисляет матрицу P, такую, что

$$PAP^T = LDL^T, A \in \mathbf{R}^{n\times n} \tag{5.7}$$

и элементы матрицы L удовлетворяют условию $|L_{ij}| \le 1$. Разложение требует $n^3/3$ флопов и может быть использовано, например, для решения линейной системы $Ax = b$. Последовательность вычислений:

$$PAP^T = LDL^T,\ Lz = Pb,\ Dw = z,\ L^Ty = w,\ x = Py.$$

Система $Dw = z$ составлена из множества симметрических неопределенных систем размера 1×1 и 2×2. Задачи размера 2×2 решаются методом исключения Гаусса с выбором ведущего элемента. Их сложность – $O(n)$. Задачи с треугольными матрицами $Lz = Pb$ и $L^Ty = w$ решаются в результате выполнения обратных подстановок (разд. 5.2.2.2). Общий объем вычислений, исключая разложение, – $O(n^2)$. То есть в случае решения симметрических неопределенных линейных систем наибольшие временные затраты падают на получение разложения (5.7). Способ его получения описан в работе [30].

Пусть после первого шага разложения

$$P_1AP_1^T = \begin{array}{cc} \begin{bmatrix} E & C^T \\ C & B \end{bmatrix} & \begin{array}{l} s \\ n-s \end{array} \\ \begin{array}{cc} s & n-s \end{array} & \end{array}, \tag{5.8}$$

где P_1 – матрица перестановок и s = 1 или 2. Если E – невырожденная матрица, то (5.8) можно представить в виде

$$P_1AP_1^T = \begin{bmatrix} I_s & 0 \\ CE^{-1} & I_{n-s} \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & B - CE^{-1}C^T \end{bmatrix} \begin{bmatrix} I_s & E^{-1}C^T \\ 0 & I_{n-s} \end{bmatrix},$$

Для достижения устойчивости матрица E выбирается такой, что минимизируется рост элементов матрицы

$$\tilde{A} = B - CE^{-1}C^T. \qquad (5.9)$$

Стратегия Банча – Парлетта выбора матрицы E такова. Пусть $0 < \alpha \le 1$ и

$$\mu_0 = \max_{i,j}|a_{ij}|, \quad \mu_1 = \max_i |a_{ii}|,$$

тогда

Если $\mu_1 \ge \alpha\mu_0$, то

s = 1

Выбрать матрицу P_1, такую, что $|e_{11}| = \mu_1$.

иначе

s = 2

Выбрать матрицу P_1, такую, что $|e_{12}| = \mu_0$.

конец если

Из (5.9) следует, что при $s = 1$

$$|\tilde{a}_{ij}| \le (1+\alpha^{-1})\mu_0,$$

а при $s = 2$

$$|\tilde{a}_{ij}| \le \frac{3-\alpha}{1-\alpha}\mu_0.$$

Сравнивая фактор роста $(1+\alpha^{-1})^2$ за 2 шага метода при $s = 1$ и фактор роста $(3-\alpha)/(1-\alpha)$ при $s = 2$, Банч и Парлетт установили, что значение $\alpha = (1+\sqrt{17})/8$ является оптимальным с позиции минимизации оценки роста элементов матрицы (5.9).

Далее приведенные действия повторяются для матрицы $\tilde{A}$ порядка $n - s$.

Пример. Процедура *undef* возвращает LDL^T-разложение матрицы A, которая может быть в том числе и неопределенной.

```
program un
 use msimsl
 integer(4), parameter :: n = 5
 real(4) :: a(n, n), L(n, n), d(n, n)
 integer(4) :: ipvt(n)                    ! ipvt – вектор перестановок
 ! Матрица A должна быть симметрической и не должна быть вырожденной
 a = reshape((/      1.0,  10.0,  20.0,  30.0,  40.0,          &
                    10.0,   1.0,  25.0,  35.0,  50.0,          &
                    20.0,  25.0,   1.0,  45.0,  60.0,          &
                    30.0,  35.0,  45.0,   1.0,  70.0,          &
                    40.0,  50.0,  60.0,  70.0,   1.0 /), shape = (/ n, n /))
 ! Возвращает LDL^T-разложение матрицы A и вектор перестановок ipvt
 call undef(n, a, L, d, ipvt)             ! L и d – массивы, представляющие матрицы L и D
```

```
print '(1x, a, 10i3)', 'ipvt: ', ipvt
call wrrrn('L', n, n, L, n, -1)          ! Выводим нижнюю треугольную часть L
call wrrrn('D', n, n, d, n, 0)           ! Выводим матрицу D
call check2(n, a, L, d, ipvt)            ! Контрольный вывод произведения LDL^T.
end program un                           ! Оно совпадает с матрицей PAP^T

subroutine undef(n, a, L, d, ipvt)       ! Вычисляет LDL^T-разложение матрицы A
integer(4) :: n, ipvt(n), k, k1, i, j    ! и вектор перестановок ipvt
integer(4) :: im0(1), im1, nc
real(4) :: a(n, n), L(n, n), d(n, n), hold(n)  ! L и d – массивы, представляющие матрицы L и D
real(4) :: alpha, m0, m1                 ! α, μ0 и μ1
real(4) :: e(2, 2)                       ! e, c и c2 – вспомогательные массивы, применяемые
real(4), allocatable :: c(:, :), c2(:, :)  ! при формировании матриц L и Ã
alpha = (1.0 + sqrt(17.0)) / 8.0
ipvt = (/ (i, i = 1, n) /)               ! Инициализация вектора перестановок
L = a; d = 0.0                           ! Инициализация матриц L и D
k = 0                                    ! k – число преобразованных строк (столбцов)
do while (k < n - 1)
! Индекс начального диагонального элемента преобразовываемой матрицы Ã
k1 = k + 1
im0 = k1 + maxloc(abs(L(k1 + 1:, k1)))   ! Исследуем столбец k1 матрицы Ã
m0 = abs(L(im0(1), k1))                  ! и находим μ0 и номер его строки в массиве L
im1 = k1; m1 = abs(L(im1, im1))          ! Находим μ1 и номер его строки в массиве L
do i = k1 + 1, n                         ! Исследуем диагональ матрицы Ã
 if(abs(L(i, i)) > m1) then; im1 = i; m1 = abs(L(im1, im1)); end if
end do
if(m1 > alpha * m0) then                 ! s = 1
 call pma(im1, 0)                        ! Симметричные перестановки
 nc = n - k - 1
 allocate(c(1, nc))
 c(1, :) = L(k1 + 1:, k1)                ! Вычисляем элементы столбца im1 матрицы L
 L(k1 + 1:, k1) = L(k1 + 1:, k1) / L(k1, k1)
 do i = k1, n                            ! Вычисляем элементы матрицы (5.9)
  do j = k1, n
   L(i, j) = L(i, j) - L(i, k1) * c(1, j - k1)
  end do
 end do
 deallocate(c)
 d(k1, k1) = L(k1, k1); L(k1, k1) = 1.0  ! Формируем матрицу D и корректируем матрицу L
 k = k + 1                               ! Число преобразованных строк (столбцов)
else                                     ! s = 2
 call pma(im0(1), 1)                     ! Симметричные перестановки
 e(:, 1) = L(k1:k1 + 1, k1)              ! Матрица E
 e(:, 2) = L(k1:k1 + 1, k1 + 1)
```

```
  ! Начало формирования столбцов k1 и k1 + 1 нижней унитреугольной матрицы L
  call Linrg(2, e, 2, e, 2)                  ! Вычисляем E^-1
  nc = n - k - 2
  allocate(c(2, nc), c2(nc, nc))
  c = L(k1:k1 + 1, k1 + 2:)                  ! Используемая далее часть матрицы L^T
  ! Вычисляем LE^-1
  L(k1 + 2:, k1:k1 + 1) = matmul(L(k1 + 2:, k1:k1 + 1), e)
  c2 = matmul(L(k1 + 2:, k1:k1 + 1), c)      ! Вычисляем LE^-1L^T
  do i = 1, nc                               ! Вычисляем матрицу (5.9)
   L(k1 + 2:, k + 2 + i) = L(k1 + 2:, k + 2 + i) - c2(:, i)
  end do
  deallocate(c, c2)
  ! Формируем матрицу D и корректируем матрицу L
  d(k1:k1 + 1, k1) = L(k1:k1 + 1, k1); d(k1:k1 + 1, k1 + 1) = L(k1:k1 + 1, k1 + 1)
  L(k1, k1) = 1.0; L(k1 + 1, k1) = 0.0; L(k1 + 1, k1 + 1) = 1.0
  k = k + 2                                  ! Число преобразованных строк (столбцов)
 end if
end do
if(k == n - 1) then                          ! В L осталась одна необработанная строка
 d(n, n) = L(n, n)                           ! Последний элемент матрицы D
 L(n, n) = 1.0
end if
do i = 1, n - 1; L(i, i + 1:) = 0.0; end do  ! Обнуляем верхнюю треугольную часть L
contains
 subroutine pma(im, ks)                      ! Симметричные перестановки в матрице A
  integer(4) :: im, ks                       ! ks = 0, если порядок блока матрицы D равен 1
  if(im /= k1) then                          ! ks = 1, если порядок блока матрицы D равен 2
   if(ks == 0) then
    ipvt(k1) = im                            ! Запоминаем положительное число, если ks = 0
   else
    ipvt(k1 + ks) = -im                      ! Запоминаем отрицательное число, если ks = 1
   end if
   hold = L(k1 + ks, :); L(k1 + ks, :) = L(im, :); L(im, :) = hold
   hold = L(:, k1 + ks); L(:, k1 + ks) = L(:, im); L(:, im) = hold
  end if
 end subroutine pma
end subroutine undef
subroutine check2(n, a, L, d, ipvt)
 real(4) :: a(n, n), L(n, n), d(n, n), a2(n, n)
 integer(4) :: ipvt(n), k, i, j              ! ipvt – вектор перестановок
 ! Вспомогательные массивы для организации контрольного вывода произведения LDL^T
 real(4) :: pk(n, n), p(n, n), hold(n)
 a2 = a                                      ! Сохраняем массив a
 ! Контрольный вывод произведения LDL^T. Оно совпадает с матрицей PAP^T
```

```
do i = 1, n; p(i, i) = 1.0; end do          ! Инициализация матрицы перестановок
do k = 1, n                                 ! В этом цикле формируется матрица перестановок
 pk = 0.0                                   ! P = Pn ... P1
 do j = 1, n; pk(j, j) = 1.0; end do        ! Формирование Pk
 j = abs(ipvt(k))
 hold = pk(k, :); pk(k, :) = pk(j, :); pk(j, :) = hold
 p = matmul(pk, p)
end do
a2 = matmul(p, a2)                          ! Вычисляем PA
a2 = matmul(a2, transpose(p))               ! Вычисляем PAP^T
call wrrrn('PAPT', n, n, a2, n, 0)
call wrrrn('LDLT', n, n, matmul(L, matmul(d, transpose(L))), n, 0)
end subroutine check2
```

Результат:

ipvt: 5 5 3 5 5

L

	1	2	3	4	5
1	1.000				
2	0.000	1.000			
3	1.732	0.707	1.000		
4	1.488	0.463	0.319	1.000	
5	1.245	0.219	0.176	0.160	1.000

D

	1	2	3	4	5
1	1.0	40.0	0.0	0.0	0.0
2	40.0	1.0	0.0	0.0	0.0
3	0.0	0.0	100.4	0.0	0.0
4	0.0	0.0	0.0	-46.3	0.0
5	0.0	0.0	0.0	0.0	18.1

Замечание. Процедуры библиотеки IMSL, например LSLSF, работающие с симметрической неопределенной матрицей, выполняют и/или используют ее UDU^T-разложение. В отличие от LDL^T-разложения вычисление верхней треугольной матрицы U начинается с последнего столбца матрицы A.

6. ПРОЦЕДУРЫ IMSL ДЛЯ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ И ВЫЧИСЛЕНИЯ ОПРЕДЕЛИТЕЛЕЙ

6.1. ВВЕДЕНИЕ

Настоящая глава содержит описания процедур, употребляемых для решения систем линейных уравнений $Ax = b$ из n уравнений с n неизвестными, выполнения LU-разложения и иных разложений матрицы A, оценки числа обусловленности матрицы и вычисления матрицы A^{-1}.

Матрица A может быть:

- вещественной несимметрической или симметрической;
- комплексной неэрмитовой или эрмитовой.

В случае вещественной несимметрической или комплексной неэрмитовой матрицы она может быть общего вида, ленточной, трехдиагональной, треугольной и разреженной.

Вещественная симметрическая или комплексная эрмитова матрица может быть положительно определенной, неотрицательно определенной или неопределенной. Симметрическая или эрмитова положительно определенная матрица может быть представлена в ленточном или разреженном виде.

Большинство процедур для линейных систем находят и/или используют соответствующее разложение матрицы. Если требования к точности решения повышены, то можно применить процедуру, итерационным образом уточняющую полученное в результате применения разложения решение. Также при работе с симметрическими разреженными положительно определенными матрицами IMSL использует для решения системы линейных уравнений итерационный метод, а точнее – разновидность метода сопряженных градиентов.

Первоначально и более детально рассматриваются процедуры, решающие системы линейных уравнений с вещественной матрицей. Процедуры для систем с комплексной матрицей имеют аналоги среди процедур для вещественной матрицы. Это дает возможность ограничить их рассмотрение списком, имеющим ссылки на аналоги, и рядом примеров, иллюстрирующих порядок применения процедур.

Большинство процедур с вещественной матрицей можно отнести к одной из двух групп. В первой процедуры работают с несимметрической матрицей и используют ее LU-разложение. Во второй – с симметрической положительно определенной и применяют R^TR-разложение Холецкого. Это обстоятельство позволяет выделить в каждой группе и подробно описать базовые процедуры, а для оставшихся привести правила вызова и снабдить

примерами. Процедуры, не попадающие в эти группы, например работающие с симметрической неопределенной или неотрицательно определенной матрицей, рассматриваются полностью.

Поскольку многие процедуры используют одноименные и одинаковые по назначению параметры, то для сокращения объема материала они сведены в таблицы, на которые при необходимости выполняются ссылки. Параметры, не включенные в эти таблицы, поясняются отдельно.

В главе также имеются рекомендации по применению процедур. Однако, забегая вперед, можно выделить две из них. Прежде следует правильно определить вид матрицы A, а затем выбрать среди имеющихся подходящий для этой матрицы метод. Это позволит уменьшить либо объем занимаемой данными памяти, либо время вычислений, либо и то и другое.

Как правило, для каждой задачи IMSL содержит процедуры, работающие с одинарной и двойной точностью. Имя процедуры второго типа образуется из имени процедуры первого типа посредством добавления префикса D; например, если LSLTO – имя процедуры, использующей тип REAL(4), то процедура, применяющая тип REAL(8), будет иметь имя DLSLTO. То же справедливо и для процедур, оперирующих комплексной матрицей.

Всегда приводится синтаксис процедур первого типа, в которых типом REAL(4) или COMPLEX(4) обладают все нецелые параметры. При использовании двойной точности всем нецелым параметрам процедуры придается тип REAL(8) или COMPLEX(8).

Большинство процедур этой главы вызывает процедуры 2-го уровня, к которым можно обратиться явно. Так, LFCRG вызывает L2CRG. На практике, однако, использование процедур 2-го уровня, с одной стороны, более трудоемко, а с другой – не предоставляет, как правило, дополнительных возможностей. Поэтому, следуя принципу экономии места, описание процедур 2-го уровня, как правило, выполняться не будет.

6.2. СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ С ВЕЩЕСТВЕННОЙ НЕСИММЕТРИЧЕСКОЙ МАТРИЦЕЙ

6.2.1. Список процедур с вещественной несимметрической матрицей

Раздел содержит описания процедур с вещественной несимметрической матрицей:

- общего вида;
- ленточной;
- трехдиагональной;
- треугольной;
- разреженной.

Процедуры перечислены в табл. 6.1. В табл. 6.2 дан синтаксис их вызова. В таблицах указывается лишь имя процедуры, работающей с одинарной точностью. В то же время каждой из приведенных процедур соответствует процедура, выполняющая вычисления с двойной точностью. Ее имя предваряется префиксом D. Все процедуры являются подпрограммами.

Таблица 6.1. Процедуры с вещественной несимметрической матрицей

Подпрограмма	*Назначение*	*Аналог*
	Вещественная несимметрическая общего вида матрица	
LFCRG	Вычисляет *LU*-разложение и выполняет оценку числа обусловленности матрицы	-
LFTRG	Вычисляет *LU*-разложение матрицы без оценки ее числа обусловленности	-
LFSRG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное подпрограммой LFCRG или LFTRG *LU*-разложение матрицы A	-
LFIRG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное *LU*-разложение матрицы A, и выполняет итерационное уточнение решения	-
LSLRG	Вычисляет *LU*-разложение матрицы A и решает систему линейных уравнений $Ax = b$	-
LSARG	Вычисляет *LU*-разложение матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	-
LINRG	Вычисляет *LU*-разложение матрицы A, а затем находит матрицу A^{-1}, обратную матрице A	-
	Вещественная несимметрическая ленточная матрица	
LFCRB	Вычисляет *LU*-разложение и выполняет оценку числа обусловленности матрицы	LFCRG
LFTRB	Вычисляет *LU*-разложение матрицы без оценки ее числа обусловленности	LFTRG
LFSRB	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное подпрограммой LFCRB *LU*-разложение матрицы A	LFSRG
LFIRB	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное *LU*-разложение матрицы A, и выполняет итерационное уточнение решения	LFIRG
LSLRB	Вычисляет *LU*-разложение матрицы A и решает систему линейных уравнений $Ax = b$	LSLRG

LSARB	Вычисляет *LU*-разложение матрицы *A*, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	LSARG
	Вещественная несимметрическая трехдиагональная матрица	
LSLCR	Вычисляет *LDU*-разложение трехдиагональной матрицы	-
LSLTR	Решает трехдиагональную систему линейных уравнений	-
	Вещественная треугольная матрица	
LFCRT	Выполняет оценку числа обусловленности матрицы	-
LINRT	Вычисляет обратную матрицу	-
LSLRT	Решает систему линейных уравнений $Ax = b$	-
GIRTS	Решает треугольную (возможно, вырожденную) систему линейных уравнений и вычисляет обобщенное обращение верхней треугольной матрицы	-
	Вещественная несимметрическая разреженная матрица	
LFTXG	Вычисляет *LU*-разложение матрицы	-
LFSXG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное подпрограммой LFTXG *LU*-разложение матрицы *A*	-
LSLXG	Решает систему линейных уравнений $Ax = b$ методом исключения Гаусса	-

<u>*Замечание*</u>. В случае двойной точности автоматически предоставляется примерно в 2 раза больше памяти.

Таблица 6.2. Синтаксис вызова подпрограмм из табл. 6.1

Синтаксис вызова	*Автоматически выделяемая память*
Вещественная несимметрическая матрица общего вида	
CALL LFCRG(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *rcond*)	*n*
CALL LFTRG(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*)	"
CALL LFSRG(*n*, *fac*, *Ldfac*, *ipvt*, *b*, *ipath*, *x*)	-
CALL LFIRG(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *b*, *ipath*, *x*, *res*)	-
CALL LSLRG(*n*, *a*, *Lda*, *b*, *ipath*, *x*)	$n^2 + 3n$
CALL LSARG(*n*, *a*, *Lda*, *b*, *ipath*, *x*)	"
CALL LINRG(*n*, *a*, *Lda*, *ainv*, *Ldainv*)	$2n + n(n - 1)/2$
Вещественная несимметрическая ленточная матрица	
CALL LFCRB(*n*, *a*, *Lda*, *nlca*, *nuca*, *fac*, *Ldfac*, *ipvt*, *rcond*)	*n*
CALL LFTRB(*n*, *a*, *Lda*, *nlca*, *nuca*, *fac*, *Ldfac*, *ipvt*)	"
CALL LFSRB(*n*, *fac*, *Ldfac*, *nlca*, *nuca*, *ipvt*, *b*, *ipath*, *x*)	-

CALL LFIRB(*n*, *a*, *Lda*, *nlca*, *nuca*, *fac*, *Ldfac*, *ipvt*, *b*, *ipath*, *x*, *res*)	-
CALL LSLRB(*n*, *a*, *Lda*, *nlca*, *nuca*, *b*, *ipath*, *x*)	$n(2nlca+nuca+1)+2n$
CALL LSARB(*n*, *a*, *Lda*, *nlca*, *nuca*, *b*, *ipath*, *x*)	"
Вещественная несимметрическая трехдиагональная матрица	
CALL LSLCR(*n*, *c*, *a*, *b*, *ijob*, *y*, *u*, *ir*, *is*)	-
CALL LSLTR(*n*, *c*, *d*, *e*, *b*)	-
Вещественная треугольная матрица	
CALL LFCRT(*n*, *a*, *Lda*, *ipath*, *rcond*)	n
CALL LINRT(*n*, *a*, *Lda*, *ipath*, *ainv*, *Ldainv*)	-
CALL LSLRT(*n*, *a*, *Lda*, *b*, *ipath*, *x*)	-
CALL GIRTS(*n*, *r*, *Ldr*, *nb*, *b*, *Ldb*, *ipath*, *irank*, *x*, *Ldx*, *rinv*, *Ldrinv*)	-
Вещественная несимметрическая разреженная матрица	
CALL LFTXG(*n*, *nz*, *a*, *irow*, *jcol*, *iparam*, *rparam*, *nfac*, *nl*, *fac*, *irfac*, *jcfac*, *ipvt*, *jpvt*)	$15n + 5maxnz$
CALL LFSXG(*n*, *nfac*, *nl*, *fac*, *irfac*, *jcfac*, *ipvt*, *jpvt*, *b*, *ipath*, *x*)	-
CALL LSLXG(*n*, *nz*, *a*, *irow*, *jcol*, *b*, *ipath*, *iparam*, *rparam*, *x*)	-

6.2.2. Возможные ошибки процедур

В табл. 6.3 указаны информационные ошибки, которые могут возникать при вызове процедур, работающих с вещественной несимметрической матрицей. Для экономии места в последнем столбце табл. 6.3 приводится подпрограмма, соответствующая типу REAL(4); имена сопутствующих ей подпрограмм (например, LFCRG сопутствуют подпрограммы DLFCRG, L2CRG и DL2CRG), в которых возникают те же ошибки, опускаются.

Таблица 6.3. Возможные информационные ошибки при работе с вещественной несимметрической матрицей

Тип	*Код*	*Причина ошибки*	*Где возникает*
3	1	Входная матрица алгоритмически вырожденная	LFCRG, LFCRB, LFCRT, LFTXG, LSLXG
3	1	Входная матрица плохо обусловлена; решение может быть неточным	LSLRG, LSARG, LINRG, LSLRB, LSARB
3	2	Коэффициент роста превысил допустимое значение	LFTXG, LSLXG

3	2	Входная матрица плохо обусловлена; итерационное уточнение не может быть эффективным	LFIRG, LFIRB
3	3	То же	LSLXG
4	2	Входная матрица является вырожденной	LFCRG, LFTRG, LSLRG, LSARG, LINRG, LFCRB, LFTRB, LSLRB, LSARB
4	2	Элемент на диагонали во время вычислений стал равным нулю	LSLTR
4	5	На диагонали есть равный нулю элемент	LFCRT, LINRT, LSLRT

6.2.3. Параметры процедур с вещественными несимметрическими и комплексными неэрмитовыми матрицами

Параметры процедур с вещественными несимметрическими и комплексными неэрмитовыми матрицами приведены для матриц:

- общего вида, ленточных и треугольных в табл. 6.4;
- разреженных в табл. 6.5;
- трехдиагональных в разд. 6.2.6.1.

Таблица 6.4. Параметры процедур с вещественными несимметрическими и комплексными неэрмитовыми общего вида, ленточными и треугольными матрицами

Имя	*Смысл*	*Типы*
a	Массив формы (*Lda*, *Lda*), содержащий элементы матрицы *A*. Как правило, *Lda* задается равным *n* – порядку матрицы *A*	REAL(4) или REAL(8); COMPLEX(4) или COMPLEX(8)
ainv	Массив формы (*Ldainv*, *Ldainv*), содержащий $n \times n$-матрицу A^{-1}, обратную матрице коэффициентов *A*. Если *A* – нижняя треугольная матрица, то и A^{-1} – нижняя треугольная матрица; если *A* – верхняя треугольная матрица, то и A^{-1} – верхняя треугольная матрица. Если массив *a* после вызова подпрограммы, например LINRT, вычисляющей обратную матрицу, не нужен, то *a* и *ainv* могут занимать одну область памяти (на месте *ainv* может стоять *a*)	Совпадает с типом массива *a*
b	Вектор размера *n*, содержащий правую часть системы линейных уравнений	То же
fac	Массив, содержащий элементы *LU*-разложения матрицы *A*; если массив *a* для дальнейших вычислений не нужен, то *fac* и *a* могут быть одним и тем же массивом	"

ipath	Флаг; *ipath* = 1, если решается система $Ax = b$, и *ipath* = 2, если решается система $A^Tx = b$. В случае треугольной матрицы *ipath* = 1, если задается нижняя треугольная матрица, и *ipath* = 2, если – верхняя	INTEGER(4)
ipvt	Вектор размера *n*, содержащий информацию о перестановках строк матрицы *A* во время выполнения ее *LU*-разложения (о целях перестановок см. разд. 5.2.3)	"
Lda	Ведущий размер массива *a* по первому и второму измерениям	"
Ldainv	Ведущий размер массива *ainv*	"
Ldfac	Протяженность массива *fac* по первому и второму измерениям	"
n	Порядок матрицы *A*	"
nlca	Число нижних кодиагоналей ленточной матрицы	"
nuca	Число верхних кодиагоналей ленточной матрицы	"
rcond	Обратная величина оценки числа обусловленности матрицы *A*	REAL(4) или REAL(8)
x	Вектор размера *n*, в который записывается решение системы линейных уравнений. Если вектор *b* для последующих вычислений не нужен, то *b* и *x* могут быть одним вектором	То же

Таблица 6.5. Параметры процедур с вещественными несимметрическими и комплексными неэрмитовыми разреженными матрицами

Имя	*Смысл*	*Типы*
a	Вектор размера *nz*, содержащий ненулевые элементы матрицы *A*	REAL(4) или REAL(8); COMPLEX(4) или COMPLEX(8)
b	Вектор размера *n*, содержащий правую часть системы линейных уравнений	Совпадает с типом вектора *a*
fac	Вектор размера *nfac*, содержащий ненулевые элементы матрицы *L* (исключая диагональные элементы) в первых *nl* позициях, и ненулевые элементы матрицы *U* в позициях от *nl* + 1 до *nfac*	Совпадает с типом вектора *a*
iparam	Вектор параметров из шести элементов; для задания принятых по умолчанию значений параметров *iparam* и *rparam* следует установить в *iparam*(1) нуль	INTEGER(4)
ipath	Флаг; *ipath* = 1, если решается система $Ax = b$, *ipath* = 2, если решается система $A^Tx = b$	"

ipvt	Вектор размера *n*, содержащей информацию о перестановках строк матрицы *A* во время ее *LU*-разложения	INTEGER(4)
irfac	Вектор размера *nfac*, содержащий номера строк ненулевых элементов матрицы *FAC*	"
irow	Вектор размера *nz*, содержащий номера строк ненулевых элементов матрицы *A*	"
jcfac	Вектор размера *nfac*, содержащий номера столбцов ненулевых элементов матрицы *FAC*	"
jcol	Вектор размера *nz*, содержащий номера столбцов ненулевых элементов матрицы *A*	"
jpvt	Вектор размера *n*, содержащей информацию о перестановках столбцов матрицы *A* во время *LU*-разложения	"
n	Порядок матрицы *A*	"
nfac	На входе – размер вектора *fac*; на выходе – число ненулевых элементов в треугольных матрицах *L* и *U*	"
nfac	Число ненулевых элементов в векторе *fac*, хранящем полученное в LFTXG (DLFTXG) *LU*-разложение	"
nl	Число ненулевых элементов в треугольной матрице *L*, исключая диагональные элементы	"
nz	Число ненулевых элементов матрицы *A*	"
rparam	Вектор параметров из пяти элементов	Совпадает с типом вектора *a*
x	Вектор размера *n*, в который записывается решение системы линейных уравнений. Если вектор *b* для последующих вычислений не нужен, то *b* и *x* можно разместить в одной области памяти (*b* и *x* могут быть одним вектором)	То же

6.2.4. Процедуры с вещественной несимметрической общего вида матрицей

6.2.4.1. Задание вещественной несимметрической общего вида матрицы

Несимметрическая вещественная общего вида $n \times n$-матрица A задается двумерным массивом a формы (Lda, Lda), содержащим элементы матрицы A ($Lda \geq n$). Чаще всего $Lda = n$.

Несимметрическая вещественная общего вида $n \times n$-матрица FAC, содержащая LU-разложение матрицы A, задается двумерным массивом *fac* формы ($Ldfac$, $Ldfac$), хранящим элементы матрицы FAC ($Ldfac \geq n$). Чаще всего $Ldfac = n$.

Тип массивов зависит от выбранной точности и может быть REAL(4) или REAL(8).

6.2.4.2. Вычисление LU-разложения и числа обусловленности матрицы

Выполняет подпрограмма LFCRG (DLFCRG):

CALL LFCRG(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *rcond*)

Входные данные: *n*, *a*, *Lda*, *Ldfac*.

Выходные данные: *fac*, *ipvt*, *rcond*.

Все параметры описаны в табл. 6.4.

Рабочий массив под эту память можно задать явно, применив L2CRG (DL2CRG):

CALL L2CRG(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *rcond*, *wk*)

wk – рабочий вектор; используется для выполнения масштабирования элементов матрицы *A*. Размер *wk* равен *n* в случае L2CRG и 2*n* в случае DL2CRG.

Замечание. Далее синтаксис процедур 2-го уровня не приводится.

Описание:

При *LU*-разложении матрицы *A* реализуется модифицированная схема частичного выбора (разд. 5.2.3.1), предусматривающая масштабирование элементов матрицы таким образом, чтобы они имели одинаковую ∞-норму $\|A\|_\infty = \max\limits_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$.

Число обусловленности матрицы $cond_1(A) = \|A\|_1 \|A^{-1}\|_1$ из-за больших временных затрат на поиск A^{-1} точно не вычисляется, а дается его оценка. Порядок ее вычисления описан в [31]. Если найденная оценка больше $1/\varepsilon_м$, то выдается предупреждение. Такое большое значение оценки числа обусловленности указывает на то, что незначительные изменения в *A* могут привести к существенным изменениям решения *x*. В некоторых случаях решение можно улучшить, выполнив его итерационное уточнение.

Подпрограмма LFCRG завершается с ошибкой, если матрица *U* – верхняя треугольная матрица *LU*-разложения – имеет на главной диагонали равный нулю элемент. Такой элемент появляется, если *A* вырожденная или близкая к ней матрица.

Прототипом LFCRG является процедура SGECO пакета LINPACK, приведенная в [34].

Замечание. Вычисляемое LFCRG *LU*-разложение матрицы *A* может быть использовано процедурами LFIRG, LFSRG и LFDRG. Так, для решения системы линейных уравнений прежде вызывается LFCRG и находится *LU*-разложение матрицы *A*, затем вызывается LFIRG или LFSRG, использующая найденное разложение для вычисления решения. Первая процедура (LFIRG) применяет итерационное уточнение корней, вторая выполняет обратный ход метода Гаусса. В случае системы с несколькими правыми частями LFIRG или LFSRG вызыва-

ются для каждой правой части. Кроме того, используя LFIRG и *LU*-разложение матрицы A, можно найти A^{-1}. Также *LU*-разложение передается процедуре LFDRG, находящей определитель матрицы.

Пример. Вычисляется A^{-1} – матрица, обратная $n \times n$-матрице A.

Порядок вычислений:

1) определить, используя LFCRG, *LU*-разложение матрицы A;
2) вызвать n раз LFIRG, находя в каждом вызове очередной столбец A^{-1}.

```
program useLU
 use msimsl, nouse => fac
 integer(4), parameter :: ipath = 1, n = 3, Lda = n, Ldfac = n
 integer(4) :: ipvt(n), j
 real(4) :: a(Lda, Lda), ainv(Lda, Lda), fac(Ldfac, Ldfac), rcond, res(n), rj(n)
 a = reshape((/     1.0,     3.0,     3.0,                &
                    1.0,     3.0,     4.0,                &
                    1.0,     4.0,     3.0 /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 ! Находим LU-разложение и оценку числа обусловленности матрицы A
 call Lfcrg(n, a, Lda, fac, Ldfac, ipvt, rcond)
 print *, 'cond1(A) = ', 1.0 / rcond            ! Вывод оценки числа обусловленности матрицы A
 rj = 0.0
 do j=1, n
  rj(j) = 1.0     ! rj(j) – столбец с номером j единичной матрицы, поэтому LFIRG
                  ! разместит столбец j обратной матрицы A-1 в j-й столбец массива ainv
  call Lfirg(n, a, Lda, fac, Ldfac, ipvt, rj, ipath, ainv(1, j), res)
  rj(j) = 0.0
 end do
 call wrrrn('ainv', n, n, ainv, Lda, 0)          ! Вывод результата
end program useLU
```

Результат:

```
cond1(A) =    66.470590
                    ainv
                 1          2          3
   1          7.000     -3.000     -3.000
   2         -1.000      0.000      1.000
   3         -1.000      1.000      0.000
```

6.2.4.3. Вычисление LU-разложения матрицы

Выполняется подпрограммой LFTRG (DLFTRG):

CALL LFTRG(n, a, Lda, fac, $Ldfac$, $ipvt$)

Отличается от приведенной в предыдущем разделе подпрограммы LFCRG только тем, что не вычисляет оценку числа обусловленности матрицы A. Применяется так же, как и LFCRG, но для работы с хорошо обусловленными системами линейных уравнений.

Прототипом LFTRG является процедура SGEFA пакета LINPACK, приведенная в [34].

Пример. Решается система линейных уравнений с m правыми частями.

Порядок решения:

1) определить, используя LFTRG, LU-разложение матрицы A;
2) вызвать m раз LFSRG, находя в каждом вызове решение системы для очередной правой части.

```
program useLU2
 use msimsl, nouse => fac
 integer(4), parameter :: ipath = 1, n = 3, Lda = n, Ldfac = n, m = 2
 integer(4) :: ipvt(n), j
 real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), b(n, m), x(n, m)
 a = reshape((/      1.0,     3.0,     3.0,                                  &
                     1.0,     3.0,     4.0,                                  &
                     1.0,     4.0,     3.0    /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 b = reshape((/      1.0,     4.0,    -1.0,                                  &
                    10.0,    14.0,     9.0    /), shape = (/ n, m /))
 call Lftrg(n, a, Lda, fac, Ldfac, ipvt)          ! Находим LU-разложение матрицы A
 do j=1, m                                        ! Решаем систему с m правыми частями
   call Lfsrg(n, fac, Ldfac, ipvt, b(1, j), ipath, x(1, j))
 end do
 call wrrrn('x', n, m, x, n, 0)                   ! Вывод результата
end program useLU2
```

Результат:

```
          x
          1          2
  1    -2.000      1.000
  2    -2.000     -1.000
  3     3.000      4.000
```

6.2.4.4. Решение системы линейных уравнений с применением ранее найденного LU-разложения

Найденное LU-разложение используется подпрограммой LFSRG (DLFSRG) для решения линейной системы $Ax = b$ ($A^T x = b$). Информацию об LU-разложении передают массивы *fac* и *ipvt*. Синтаксис LFSRG:

CALL LFSRG(n, *fac*, *Ldfac*, *ipvt*, b, *ipath*, x)

Входные данные: n, *fac*, *Ldfac*, *ipvt*, b, *ipath*.

Выходные данные: x.

Все параметры описаны в табл. 6.4.

Описание:

Прежде чем вызывать подпрограмму LFSRG, вычисляется *LU*-разложение матрицы *A*. Это выполняется либо LFCRG, либо LFTRG. Решение системы $Ax = b$ ищется в результате решения систем с треугольными матрицами $Ly = b$ и $Ux = y$ (см. разд. 5.2.2.2).

Прототипом LFSRG является процедура SGESL пакета LINPACK, приведенная в [34].

Пример. Решается система линейных уравнений с хорошо обусловленной матрицей.

```
program useLU3
 use msimsl, nouse => fac
 integer(4), parameter :: ipath = 1, n = 3, Lda = n, Ldfac = n
 integer(4) :: ipvt(n), j
 real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), b(n), x(n)
 a = reshape((/    1.0,     3.0,     3.0,                    &
                   1.0,     3.0,     4.0,                    &
                   1.0,     4.0,     3.0 /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 b = (/ 1.0, -1.0, 2.5 /)
 ! Находим LU-разложение матрицы A без оценки cond1(A)
 call Lftrg(n, a, Lda, fac, Ldfac, ipvt)
 call Lfsrg(n, fac, Ldfac, ipvt, b, ipath, x)    ! Решаем линейную систему Ax = b
 call wrrrn('x', 1, n, x, 1, 0)                  ! Вывод результата
end program useLU3
```

Результат:

```
                x
  2.500       1.500       -2.000
```

6.2.4.5. Итерационное уточнение решения линейной системы

Подпрограмма LFIRG (DLFIRG) позволяет, используя найденное ранее подпрограммой LFCRG (DLFCRG) или LFTRG (DLFTRG) *LU*-разложение матрицы *A*, выполнить итерационное уточнение решения линейной системы $Ax = b$. Синтаксис LFIRG:

CALL LFIRG(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *b*, *ipath*, *x*, *res*)

Входные данные: *n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *b*, *ipath*.

Выходные данные: *x*, *res*.

Все параметры, кроме *res*, описаны в табл. 6.4.

res – вектор типа REAL(4) размера *n*, содержащий невязку улучшенного решения.

Алгоритм итерационного уточнения решения линейной системы приведен в разд. 5.2.7.2. Для вычислений необходимо, применив LFCRG или LFTRG, найти *fac* и *ipvt*.

Пример:

```
program useLU4
 use msimsl, nouse => fac
 integer(4), parameter :: ipath = 1, n = 3, Lda = n, Ldfac = n
 integer(4) :: ipvt(n)
 real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), b(n), x(n), res(n)
 a = reshape((/      1.0,     3.0,     3.0,                          &
                     1.0,     3.0,     4.0,                          &
                     1.0,     4.0,      3.0    /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 b = (/ -0.5, -1.0, 1.5 /)
 call Lftrg(n, a, Lda, fac, Ldfac, ipvt)          ! Находим LU-разложение матрицы A
 ! Выполняем итерационное уточнение решения линейной системы Ax = b
 call Lfirg (n, a, Lda, fac, Ldfac, ipvt, b, ipath, x, res)
 call wrrrn('x', 1, n, x, 1, 0)                   ! Вывод решения и невязки
 print '(1x, "res = ", 10f6.3)', res              ! res =  .000  .000  .000
end program useLU4
```

Результат:

```
                x
 -5.000        2.000        -0.500
```

6.2.4.6. Решение системы линейных уравнений без уточнения корней

Находит подпрограмма LSLRG (DLSLRG):

CALL LSLRG(*n*, *a*, *Lda*, *b*, *ipath*, *x*)

Входные данные: *n*, *a*, *Lda*, *b*, *ipath*.

Выходные данные: *x*.

Все параметры описаны в табл. 6.4.

Рабочие параметры LSLRG можно изменить подпрограммой IUMAG (разд. 3.6). Для предотвращения конфликтов памяти следует использовать с IUMAG в массиве *iopts* целочисленное значение 16. Тогда ведущее измерение массива *fac* увеличивается на *ivals*(3), когда *n* кратно *ivals*(4). Значения по умолчанию: *ivals*(1:4) = (/ 1, 16, 0, 1 /). Первые две позиции используются LSLRG, две вторые – для процедуры 2-го уровня L2LRG.

Целочисленное значение 17, заданное в массиве *iopts*, позволяет включать и отключать режим расчета оценки числа обусловленности. Оценка числа обусловленности будет вычисляться, если *ivals*(2) = 2, и нет – в противном случае. Значения по умолчанию: *ivals*(1:2) = (/ 1, 2 /). Первая позиция используется LSLRG, вторая – L2LRG.

Описание:

Первоначально используется LFCRG для вычисления *LU*-разложения. Если задать *ivals*(1) = 2 и использовать с IUMAG в массиве *iopts* целочисленное имя 17, то LSLRG будет вычислять оценку числа обусловленности матрицы. Решение системы осуществляется подпрограммой LFSRG.

Подпрограмма LFCRG завершается с ошибкой, если матрица U – верхняя треугольная матрица LU-разложения – имеет на главной диагонали равный нулю элемент. Такой элемент появляется, если A вырожденная или близкая к ней матрица.

В случае плохо обусловленной матрицы лучше применять LSVRR или LSARG.

Пример 1:

```
program useLrg
 use msimsl
 integer, parameter :: ipath = 1, Lda = 3, n = 3
 real :: a(Lda, Lda), b(n), x(n)
 a = reshape((/      33.0,     16.0,     72.0,                    &
                    -24.0,    -10.0,    -57.0,                    &
                     18.0,    -11.0,      7.0    /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 b = (/ 129.0, -96.0, 8.5 /)
 call Lslrg(n, a, Lda, b, ipath, x)          ! Решаем линейную систему Ax = b
 call wrrrn('x', 1, n, x, 1, 0)              ! Вывод результата
end program useLrg
```

Результат:

```
                    x
    1.000          1.500          1.000
```

Пример 2. Ведущее измерение fac ($fac = a$) увеличивается на $ivals(3) = 1$ и становится равным $n + ivals(3) = 16 + 1 = 17$. Увеличение выполняется поскольку $n = ivals(4) = 16$. Для теста используется несимметрическая матрица Адамара и известное решение $x_j = j$, $j = 1, \ldots, n$. (Элементы $n \times n$-матрицы Адамара равны +1 и -1 и имеет место равенство $AA^T = nI$, где I – единичная матрица порядка n, которое эквивалентно утверждению: любые две строки A ортогональны.)

```
program useL2lrg
 use msimsl
 integer(4), parameter :: Lda = 17, n = 16, ichp=1, ipath=1, iput=2, kbank=16
 integer(4) :: i, ipvt(n), j, k, nn, iopt(1), ivals(4)
 real(4) :: a(Lda, n), b(n), wk(n), x(n)
 a(1, 1) = 1.0; nn = 1
 do k = 1, 4                                 ! Генерация матрицы Адамара
  do j = 1, nn
   do i = 1, nn
    a(nn + i, j) = -a(i, j);  a(i, nn + j) = a(i, j);  a(nn + i, nn + j) = a(i, j)
   end do
  end do
  nn = nn + nn
 end do
```

```
x = (/ (j, j=1, n) /)                          ! Задаем решение
b = matmul(a(1:n, :), x)                       ! Находим b = Ax
x = 0.0                                        ! Восстанавливаем x
iopt(1) = kbank                                ! Параметры, позволяющие избежать
ivals = (/ 1, 16, 1, 16 /)                     ! конфликтов памяти (разд. 3.6)
call iumag('MATH', ichp, iput, 1, iopt, ivals)
call L2lrg(n, a, Lda, b, ipath, x, a, ipvt, wk) ! Решаем линейную систему Ax = b
call wrrrn('x', 1, n, x, 1, 0)
end program useL2lrg
```

Результат:

```
                                    x
  1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0  10.0  11.0  12.0  13.0  14.0  15.0  16.0
```

6.2.4.7. Решение системы линейных уравнений с итерационным уточнением корней

Находит подпрограмма LSARG (DLSARG):

CALL LSARG(*n*, *a*, *Lda*, *b*, *ipath*, *x*)

Входные данные: *n*, *a*, *Lda*, *b*, *ipath*.

Выходные данные: *x*.

Все параметры описаны в табл. 6.4.

Описание:

Первоначально используется LFCRG для вычисления *LU*-разложения матрицы коэффициентов. Затем вызывается LFIRG и выполняется итерационное уточнение решения.

Подпрограмма LSARG завершается с ошибкой, если *U* – верхняя треугольная матрица *LU*-разложения – имеет на главной диагонали равный нулю элемент или если в процессе итераций решение расходится. Ошибки возникают, если матрица коэффициентов вырожденная или очень близка к вырожденной. Если оценка числа обусловленности больше чем $1/\varepsilon_м$, то подпрограмма выдаст предупреждение. В то же время в некоторых случаях подпрограмма может найти решение и при таком высоком значении числа обусловленности, однако решение может быть весьма неточным.

Пример:

```
program useLsarg
 use msimsl
 integer, parameter :: ipath = 1, Lda = 3, n = 3
 real :: a(Lda, Lda), b(n), x(n)
 a = reshape((/      33.0,     16.0,     72.0,                    &
                    -24.0,    -10.0,    -57.0,                    &
                     18.0,    -11.0,      7.0    /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 b = (/ 129.0, -96.0, 8.5 /)                    ! Решаем линейную систему Ax = b
 call Lsarg(n, a, Lda, b, ipath, x)             ! и выполняем итерационное уточнение корней
```

```
 call wrrrn('x', 1, n, x, 1, 0)                    ! Вывод результата
end program useLsarg
```

Результат:

```
                    x
   1.000          1.500          1.000
```

6.2.4.8. Вычисление обратной матрицы

Выполняет подпрограмма LINRG (DLINRG):

CALL LINRG(*n*, *a*, *Lda*, *ainv*, *Ldainv*)

Входные данные: *n*, *a*, *Lda*, *b*, *ipath*, *Ldainv*.

Выходные данные: *ainv*.

Параметры описаны в табл. 6.4.

Описание:

Первоначально используется LFCRG для вычисления *LU*-разложения матрицы коэффициентов и оценки ее числа обусловленности. Затем вызывается LFIRG и выполняется итерационное уточнение решения. A^{-1} вычисляется как произведение $A^{-1} = U^{-1}L^{-1}$. Матрица U^{-1} вычисляется LINRT.

Подпрограмма LINRG завершается с ошибкой, если U – верхняя треугольная матрица *LU*-разложения – имеет на главной диагонали равный нулю элемент или если в процессе итераций решение расходится. Ошибки возникают, если матрица коэффициентов вырожденная или очень близка к вырожденной. Если оценка числа обусловленности больше чем $1/\varepsilon_м$, то подпрограмма выдаст предупреждение.

Пример:

```
program useLinrg
 use msimsl
 integer, parameter :: n = 3, Lda = n, Ldainv = n
 real :: a(Lda, Lda), ainv(Ldainv, Ldainv)
 a = reshape((/      1.0,     3.0,     3.0,                    &
                     1.0,     3.0,     4.0,                    &
                     1.0,     4.0,     3.0   /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 call Linrg(n, a, Lda, ainv, Ldainv)          ! Вычисляем A-1
 call wrrrn('ainv', n, n, ainv, Ldainv, 0)    ! Вывод результата
end program useLinrg
```

Результат:

```
                    ainv
                 1            2            3
   1         7.000       -3.000       -3.000
   2        -1.000        0.000        1.000
   3        -1.000        1.000        0.000
```

6.2.5. Пример описания процедуры с вещественной несимметрической ленточной матрицей

Список процедур с вещественной несимметрической ленточной матрицей приведен в табл. 6.1. Поскольку процедуры имеют ранее рассмотренные аналоги, в разделе в качестве примера рассмотрены только подпрограммы LFCRB и DLFCRB. Знакомство с иными процедурами для вещественной несимметрической ленточной матрицы следует выполнить по описанию аналогов.

6.2.5.1. Задание ленточной матрицы

Ленточная вещественная $n \times n$-матрица A (разд. 4.3.2) задается двумерным массивом a формы (Lda, n), содержащим ($nlca + nuca + 1$) n элементов матрицы A ($Lda \geq nlca + nuca + 1$), где $nlca$ и $nuca$ соответственно число нижних и верхних кодиагоналей. Чаще всего $Lda = nlca + nuca + 1$.

Ленточная вещественная $n \times n$-матрица FAC, содержащая LU-разложение матрицы A, задается двумерным массивом fac формы ($Ldfac$, n), хранящим элементы матрицы FAC ($Ldfac \geq 2nlca + nuca + 1$). Чаще всего $Ldfac = 2nlca + nuca + 1$. Тип массивов зависит от выбранной точности и может быть REAL(4) или REAL(8).

6.2.5.2. Вычисление LU-разложения и числа обусловленности матрицы

Вычисляемое приводимой в разделе процедурой LFCRB (DLFCRB) LU-разложение матрицы A может быть использовано процедурами LFIRB, LFSRB и LFDRB. Их назначение аналогично назначению приведенных в разд. 6.2.4 процедур LFIRG, LFSRG и LFDRG, работающих с несимметрической матрицей общего вида.

LU-разложение и оценку числа обусловленности возвращает подпрограмма LFCRB (DLFCRB):

CALL LFCRB(*n*, *a*, *Lda*, *nlca*, *nuca*, *fac*, *Ldfac*, *ipvt*, *rcond*)

Входные данные: *n*, *a*, *Lda*, *nlca*, *nuca*, *Ldfac*.

Выходные данные: *fac*, *ipvt*, *rcond*.

Все параметры описаны в табл. 6.4 и разд. 6.2.5.1.

Описание:

Алгоритм LU-разложения дан в разд. 6.2.4.2. Прототипом LFCRB является процедура SGBCO пакета LINPACK, приведенная в [34]. Оценка числа обусловленности матрицы выполняется по схеме, описанной в [31].

В массиве *fac* первые $nlca + nuca + 1$ строк содержат верхнюю треугольную матрицу U в ленточном представлении (разд. 4.3.2). Нижние $nlca$ строк – множители, необходимые для воспроизведения L^{-1}.

Пример. Вычисляется матрица, обратная 4×4-ленточной матрице с *nlca* = = *nuca* = 1 (столбцы обратной матрицы вычисляются подпрограммой LFIRB).

```
program useLUb
 use msimsl, nouse => fac
 integer(4), parameter :: ipath = 1, nlca = 1, nuca = 1,    &
                          Lda = nlca + nuca + 1, n = 4, Ldfac = 2 * nlca + nuca + 1
 integer(4) :: ipvt(n), j
 real(4) :: a(Lda, n), ainv(n, n), fac(Ldfac, n), rcond, res(n), rj(n) = 0.0
 ! Ленточное представление матрицы A
 a = reshape((/      0.0,     -1.0,     -2.0,      2.0,                    &
                     2.0,      1.0,     -1.0,      1.0,                    &
                    -3.0,      0.0,      2.0,      0.0    /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 ! Находим LU-разложение и оценку числа обусловленности матрицы A
 call Lfcrb(n, a, Lda, nlca, nuca, fac, Ldfac, ipvt, rcond)
 print *, 'cond1(A) = ', 1.0 / rcond            ! Вывод оценки числа обусловленности матрицы A
 do j=1, n
  rj(j) = 1.0      ! rj(j) – столбец с номером j единичной матрицы, поэтому LFIRB
                   ! разместит столбец j обратной матрицы A-1 в j-й столбец массива ainv
  call Lfirb(n, a, Lda, nlca, nuca, fac, Ldfac, ipvt, rj, ipath, ainv(1, j), res)
  rj(j) = 0.0
 end do
 call wrrrn('ainv', n, n, ainv, n, 0)           ! Вывод результата
end program useLUb
```

Результат:

cond1(A) = 15.350880

ainv

	1	2	3	4
1	-1.000	-1.000	0.400	-0.800
2	-3.000	-2.000	0.800	-1.600
3	0.000	0.000	-0.200	0.400
4	0.000	0.000	0.400	0.200

6.2.6. Процедуры с вещественной несимметрической трехдиагональной матрицей

6.2.6.1. Вычисление LDU-разложения и решение линейной системы Ax = b с трехдиагональной матрицей

Выполняется подпрограммой LSLCR (DLSLCR):

CALL LSLCR(*n*, *c*, *a*, *b*, *ijob*, y, *u*, *ir*, *is*)

Входные данные: *n*, *ijob*.

Входные/выходные данные: *c*, *a*, *b*, *y*.

Выходные данные: *u*, *ir*, *is*.

n – порядок трехдиагональной матрицы *A*.

c – массив размера 2*n*, содержащий верхнюю кодиагональ матрицы *A*; элементы кодиагонали заносятся в элементы *c*(1), ..., *c*(*n* - 1) массива *c*.

a – массив размера 2*n*, содержащий главную диагональ матрицы *A*; элементы диагонали заносятся в элементы *a*(1), ..., *a*(*n*) массива *a*.

b – массив размера 2*n*, содержащий нижнюю кодиагональ матрицы *A*; элементы кодиагонали заносятся в элементы *b*(1), ..., *b*(*n* - 1) массива *b*.

ijob – параметр управляющий разложением; если

- *ijob* = 1, то выполняется разложение матрицы и решается система *Ax* = *y*;
- *ijob* = 2, то решается система *Ax* = *y* (разложение матрицы выполнено ранее);
- *ijob* = 3, то выполняется разложение матрицы *A* (система не решается);
- *ijob* = 4, 5 или 6, то выполняется только разложение матрицы *A*, но для повышения быстродействия входные данные не проверяются на предмет наличия в них ошибок.

y – массив размера 2*n*, содержащий правую часть решаемой системы линейных уравнений *Ax* = *y*; правая часть заносится в элементы *y*(1), ..., *y*(*n*) массива *y*.

Решение (вектор *x*) заносится в первые *n* элементов вектора *y*.

u – массив флажков, сигнализирующих о вырожденности матрицы *A*. Величина *u*(*i*) = 1 означает, что во время разложения на шаге *i* произошло деление на нуль, при отсутствии такого деления *u*(*i*) = 0. Размер массива 2*n*.

ir и *is* – массивы целых чисел, возвращающие число итераций, выполненных алгоритмом циклической редукции. Размер массивов не менее $\log_2(n) + 3$.

Описание:

Подпрограмма LSLCR выполняет, используя алгоритм циклической редукции, разложение трехдиагональной матрицы *A* = *LDU*, в котором *L* и *U* – соответственно нижняя и верхняя унитреугольные матрицы, а *D* – диагональная матрица. Матрица *A* может быть и несимметрической. Скорее всего, LSLCR окажется более производительной, чем LSLTR или LSLPB, на векторных параллельных компьютерах, но, возможно, менее производительной при малых значениях *n*, на скалярных компьютерах и высокопроизводительных компьютерах без оптимизирующих компиляторов. С деталями алгоритма можно ознакомиться в [55] и [50].

Пример. Решается система из *n* = 1000 линейных уравнений. Матрица коэффициентов является симметрической матрицей вторых разностных производных. Вектор *y* правой части системы является первым столбцом единичной матрицы. Решением системы будет первый столбец матрицы A^{-1}. Затем решается новая система, в которой вектор *y* является последним столбцом единичной матрицы. Решением системы будет последний столбец матрицы A^{-1}.

```
program trid                                  ! Поиск LDU-разложения треугольной матрицы
 use msimsl
 use text_transfer
 integer(4), parameter :: Lp = 12, n = 1000, n2 = 2 * n
 integer(4) :: ijob, ir(Lp), is(Lp)
 real(4) :: a(n2), b(n2), c(n2), u(n2), y1(n2), y2(n2)
 ! Матрица A (ее представляющие массивы)
 c(1:n - 1) = -1.0; a(1:n) = 2.0; b(1:n - 1) = -1.0
 y1(2:n) = 0.0; y2(1:n - 1)  = 0.0; y1(1) = 1.0; y2(n) = 1.0
 ijob = 1                                     ! LDU-разложение матрицы
 call Lslcr(n, c, a, b, ijob, y1, u, ir, is)  ! и решение линейной системы Ax = b
 ijob = 2                                     ! Решаем систему с другой правой частью,
 call Lslcr(n, c, a, b, ijob, y2, u, ir, is)  ! используя ранее найденное LDU-разложение
 print *, trim(ru_doswin('Число уравнений: ', .false.)), n
 print '(1x, a, 2f9.2)',                      &
       trim(ru_doswin('Элементы столбцов 1 и n обратной матрицы:', .false.)), y1(1), y2(n)
end program trid
```

Результат:

```
Число уравнений:      1000
Элементы столбцов 1 и n обратной матрицы:    1.00    1.00
```

Замечание. В интерфейсе LSLCR (файл msimsls.f90) не описаны векторы *ir* и *is* поэтому в файле msimsls.mod его заголовок и две последние строки выглядят так:

```
PROC LSLCR,9,10,0: 8,0,0,0,262144,0,1
. . .
VAR IR,3,0,,: 1,0,0,0,2,0,0
VAR IS,3,0,,: 1,0,0,0,2,0,0
ENDPROC
```

Следует поправить интерфейс, добавив в него объявления *ir* и *is*:

```
interface
 subroutine Lslcr (n, c, a, b, ijob, y, u, ir, is)
  integer n, ijob, ir(*), is(*)
  real c(*), a(*), b(*), y(*), u(*)
 end subroutine
end interface
```

и внести изменения в заголовок и в две последние строки интерфейса в файле msimsls.mod:

```
PROC LSLCR,9,10,0: 8,0,0,0,393216,0,1
. . .
VAR IR,3,0,,: 1,0,0,22,3,0,0 (1,3,0: 1,0,1,4)
VAR IS,3,0,,: 1,0,0,25,3,0,0 (1,3,0: 1,0,1,4)
ENDPROC
```

6.2.6.2. Решение трехдиагональной системы линейных уравнений

Выполняется подпрограммой LSLTR (DLSLTR):

CALL LSLTR(n, c, d, e, b)

Входные данные: n.

Входные/выходные данные: c, d, e, b.

n – порядок трехдиагональной матрицы A.

c – вектор размера n, содержащий нижнюю кодиагональ матрицы A; элементы кодиагонали заносятся в элементы $c(2)$, ..., $c(n)$.

d – вектор размера n, содержащий главную диагональ матрицы A.

e – вектор размера n, содержащий верхнюю кодиагональ матрицы A; элементы кодиагонали заносятся в элементы $e(1)$, ..., $e(n - 1)$ массива b.

b – вектор размера n, содержащий на входе правую часть решаемой системы линейных уравнений $Ax = b$ и ее решение на выходе.

Описание:

Используется метод исключения Гаусса с частичным выбором. Подробно алгоритм приведен в [34] при рассмотрении процедуры SGTSL (DGTSL).

Пример. Решается система их четырех уравнений.

```
program trid2
 use msimsl
 integer(4), parameter :: n = 4
 real(4) :: b(n), c(n), d(n), e(n)
 c = (/ 0.0, 0.0, -4.0, 9.0 /); d = (/ 6.0, 4.0, -4.0, -9.0 /); e = (/-3.0, 7.0, -8.0, 0.0 /)
 b = (/ 48.0, -81.0, -12.0, -144.0 /)
 call Lsltr(n, c, d, e, b)                    ! Решаем линейную систему Ax = b
 call wrrrn('x', 1, n, b, 1, 0)               ! Вывод результата
end program trid2
```

Результат:

```
                    x
   4.000      -8.000      -7.000       9.000
```

<u>*Замечание*</u>. Помимо приведенных выше процедур LSLCR и LSLTR для решения линейных систем с трехдиагональной матрицей можно употреблять процедуру LSLPB (разд. 6.3.5.2), приспособленную для ленточных систем.

6.2.7. Процедуры с вещественной треугольной матрицей

Рассматриваемые в разделе процедуры перечислены в табл. 6.1.

6.2.7.1. Задание треугольной матрицы

Треугольная вещественная $n \times n$-матрица A задается двумерным массивом a формы (Lda, Lda), содержащим элементы матрицы A ($Lda \geq n$). Чаще всего $Lda = n$.

Для задания нижней или верхней треугольной матрицы используется параметр *ipath*; *ipath* = 1, если задается нижняя треугольная матрица, и *ipath* = 2, если – верхняя.

6.2.7.2. Оценка числа обусловленности треугольной матрицы

Выполняется подпрограммой LFCRT (DLFCRT):

CALL LFCRT(*n*, *a*, *Lda*, *ipath*, *rcond*)

Входные данные: *n*, *a*, *Lda*, *ipath*.

Выходные данные: *rcond*.

Параметры описаны в табл. 6.4 и разд. 6.2.7.1.

Если найденная оценка числа обусловленности $cond_1(A) = 1/rcond > 1/\varepsilon_M$, то выдается предупреждение. Способ оценки числа обусловленности такой же, как в [31].

Пример:

```
program findCond
 use msimsl
 integer(4), parameter :: ipath = 1, n = 3, Lda = n
 real(4) :: a(Lda, Lda), rcond
 a = reshape((/        2.0,      0.0,      0.0,                    &
                       2.0,     -1.0,      0.0,                    &
                      -4.0,      2.0,       5.0    /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 ! Находим оценку числа обусловленности матрицы A
 call Lfcrt(n, a, Lda, ipath, rcond)          ! Нижняя треугольная матрица (ipath = 1)
 print *, 'cond1(A) = ', 1.0 / rcond          ! Вывод оценки числа обусловленности матрицы A
end program findCond
```

Результат:

```
cond1(A) = 10.967740
```

6.2.7.3. Вычисление обратной треугольной матрицы

Выполняется подпрограммой LINRT (DLINRT):

CALL LINRT(*n*, *a*, *Lda*, *ipath*, *ainv*, *Ldainv*)

Входные данные: *n*, *a*, *Lda*, *ipath*, *Ldainv*.

Выходные данные: *ainv*.

Параметры описаны в табл. 6.4.

Подпрограмма завершается с ошибкой, если *A* имеет на диагонали равный нулю элемент.

Пример:

```
program inverse
 use msimsl
 integer(4), parameter :: ipath = 1, n = 3, Lda = n, Ldainv = n
 real(4) :: a(Lda, Lda), ainv(Ldainv, Ldainv)
```

```
 a = reshape((/        2.0,     0.0,     0.0,                    &
                       2.0,    -1.0,     0.0,                    &
                      -4.0,     2.0,     5.0    /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 ! Находим A-1 – матрицу, обратную матрице A
 call Linrt(n, a, Lda, ipath, ainv, Ldainv)     ! Нижняя треугольная матрица (ipath = 1)
 call wrrrn('ainv', n, n, ainv, Lda, 0)         ! Вывод результата
end program inverse
```

Результат:

ainv

	1	2	3
1	0.500	0.000	0.000
2	1.000	-1.000	0.000
3	0.000	0.400	0.200

6.2.7.4. Решение треугольной системы линейных уравнений

Находится подпрограммой LSLRT (DLSLRT):

CALL LSLRT(*n*, *a*, *Lda*, *b*, *ipath*, *x*)

Входные данные: *n*, *a*, *Lda*, *ipath*, *b*.

Выходные данные: *x*.

Параметры описаны в табл. 6.4.

Дополнительные сведения о параметре *ipath*. Если

- *ipath* = 1, то решается система $Ax = b$, где A – нижняя треугольная матрица (ТМ);
- *ipath* = 2, то решается система $Ax = b$, где A – верхняя ТМ;
- *ipath* = 3, то решается система $A^Tx = b$, где A – нижняя ТМ;
- *ipath* = 4, то решается система $A^Tx = b$, где A – верхняя ТМ.

Подпрограмма завершается с ошибкой, если A имеет на диагонали равный нулю элемент.

Прототипом LSLRT является процедура STRSL пакета LINPACK, приведенная в [34].

Пример:

```
program findX
 use msimsl
 integer(4), parameter :: ipath = 1, n = 3, Lda = n, Ldainv = n
 real(4) :: a(Lda, Lda), b(Lda), x(Lda)
 a = reshape((/        2.0,     0.0,     0.0,                    &
                       2.0,    -1.0,     0.0,                    &
                      -4.0,     2.0,     5.0    /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
 b = (/ 2.0, 5.0, 0.0 /)                        ! Решаем линейную систему Ax = b
 call Lslrt(n, a, Lda, b, ipath, x)             ! Нижняя треугольная матрица (ipath = 1)
 call wrrrn('x', 1, n, x, 1, 0)                 ! Вывод результата
end program findX
```

Результат:

```
              x
 1.000      -3.000      2.000
```

6.2.7.5. Решение треугольной (возможно, вырожденной) системы линейных уравнений и вычисление обобщенного обращения верхней треугольной матрицы

Находится подпрограммой GIRTS (DGIRTS):

CALL GIRTS(*n*, *r*, *Ldr*, *nb*, *b*, *Ldb*, *ipath*, *irank*, *x*, *Ldx*, *rinv*, *Ldrinv*)

Входные данные: *n*, *r*, *Ldr*, *nb*, *b*, *Ldb*, *ipath*, *Ldx*, *Ldrinv*.

Выходные данные: *irank*, *x*, *rinv*.

Вещественные данные в случае GIRTS имеют тип REAL(4), в случае DGIRTS – REAL(8); целочисленные – INTEGER(4).

n – порядок матрицы *R*.

r – вещественный массив формы (*Ldr*, *n*), содержащий верхнюю треугольную *n*×*n*-матрицу *R*. Если *r* в некоторой строке содержит на диагонали нуль, все последующие элементы этой строки должны равняться нулю. Адресуется только верхняя треугольная часть *R*.

Ldr – ведущий размер массива *r* по первому измерению.

nb – число столбцов в массиве *b*. Если *nb* = 0, то система линейных уравнений не решается.

b – вещественный массив формы (*Ldb*, *nb*), содержащий *nb* правых частей системы линейных уравнений. Если *nb* = 0, то *b* не адресуется и может быть вектором размера 1.

Ldb – ведущий размер массива *b*.

ipath – параметр типа INTEGER(4), задающий выполняемые GIRTS действия. Если

- *ipath* = 1, то решается система $Rx = b$;
- *ipath* = 2, то решается система $R^T x = b$;
- *ipath* = 3, то решается система $Rx = b$ и вычисляется *rinv*;
- *ipath* = 4, то решается система $R^T x = b$ и вычисляется *rinv*.

irank – ранг матрицы *R*.

x – вещественный массив формы (*Ldx*, *nb*), содержащий решения системы линейных уравнений с несколькими правыми частями, записанными в массиве *b*. Если *b* после вычислений не нужен, то *x* и *b* могут быть одним массивом. Если *nb* = 0, то *x* не адресуется и может быть вектором размера 1.

Ldx – ведущий размер массива *x* по первому измерению.

rinv – вещественный массив формы (*Ldrinv*, *n*), содержащий верхнюю треугольную *n*×*n*-матрицы R^{-1}, когда матрица *R* не является вырожденной. Вычисляется, когда *ipath* = 3 или 4. Когда *R* – вырожденная матрица, то *rinv*

содержит $g3$-обратную матрицу. Если *ipath* = 1 или 2, *riniv* не адресуется и может быть вектором размера 1. Если *ipath* = 3 или 4 и *r* после вызова GIRTS не нужен, то *r* и *rinv* могут быть одним массивом.

Ldrinv – ведущий размер массива *rinv*.

Замечания:

1. При вызовах GIRTS, если система линейных уравнений несовместна, возникнет информационная ошибка тина 3 с кодом 1. Несовместность возникает, когда строка матрицы состоит из нулей, а соответствующий элемент в правой части недостаточно близок к нулю. Так, если в приводимом ниже примере в четвертой строке матрицы разместить одни нули, то возникнет предупреждение

```
*** WARNING ERROR 1 from GIRTS. The linear system of equations is
***          inconsistent within the computed tolerance. Elements of row 4
***          are zero, but B(4,1) = 3.000000E+00.  X(4,1) is set to zero.
```

2. Подпрограмма GIRTS может работать с вырожденной матрицей *R*, если вырожденность представляется нулевыми строками матрицы. Никаких других форм вырожденности *R* не допускается.

Описание:

Подпрограмма GIRTS решает систему линейных уравнений, если $nb > 0$, с треугольной матрицей коэффициентов *R* и несколькими правыми частями и находит обратную или $g3$-обратную матрицу R^{-1}. Если матрица *R* содержит в некоторой строке нуль, а хотя бы один последующий элемент этой строки отличен от нуля, то возникнет завершающая ошибка с кодом 15:

```
*** TERMINAL ERROR 15 from GIRTS. R(3,3) = .000000E+00. Remaining elements
***          for the row must also be zero, but R(3,4) = 1.000000E+00 is not.
Stop – Program terminated.
```

Если решается система линейных уравнений ($nb > 0$) и строка *i* матрицы коэффициентов состоит из нулей, то *i*-я строка решения *x* принимается равной нулю и,

- если *ipath* = 1 или 3 и *i*-я строка правой части (матрицы *B*) не состоит из нулей, выдается предупреждение;
- если *ipath* = 2 или 4 и *i*-я строка сжатой правой части, получаемой после удалений первых $i - 1$ значений в строке *i*, отличается от нуля на величину, большую заданного допуска, выдается предупреждение.

Если вычисляется обратная матрица R^{-1} и строка *i* состоит из нулей, то и в строку и в столбец с номером *i* матрицы R^{-1} (*rinv*) записываются нули. Результирующая обратная матрица называется *g3-обратной матрицей*.

Матрица *G* является $g3$-обратной матрицей *A*, если *G* удовлетворяет условиям 1, 2 и 3 Мура – Пенроуза и обычно не удовлетворяет условию 4.

Матрица G является Мура – Пенроуза обобщенной обратной матрицей A, если:

1. $AGA = A$;
2. $GAG = G$;
3. AG является симметрической матрицей;
4. GA также является симметрической матрицей.

Подробное описание алгоритма изложено в [68, Section 2].

Пример. Решается система линейных уравнений $Rx = b$ и вычисляется $g3$-обратная матрица R [62, p. 102-105].

```
program g3
 use msimsl
 integer(4), parameter :: n = 4, nb = 1, Ldb = n, Ldr = n, Ldrinv = n, Ldx = n
 integer(4) :: ipath, irank
 real(4) :: b(Ldb, nb), r(Ldr, n), rinv(Ldrinv, n), x(Ldx, nb)
 r = reshape((/      6.0,    2.0,    5.0,    1.0,                &
                     0.0,    4.0,   -2.0,    2.0,                &
                     0.0,    0.0,    0.0,    0.0,                &
                     0.0,    0.0,    0.0,    3.0 /), shape = (/ Ldrinv, n /), order = (/ 2, 1 /))
 b(:, 1) = (/ 3.0, 4.0, 0.0, 3.0 /); ipath = 3
 call girts(n, r, Ldr, nb, b, Ldb, ipath, irank, x, Ldx, rinv, Ldrinv)
 call wrrrn('rinv', n, n, rinv, Ldrinv, 0); call wrrrn('x', 1, Ldx, reshape(x, shape = (/ Ldx /)), 1, 0)
end program g3
```

Результат:

```
                    rinv
            1        2        3        4
   1   0.1667  -0.0833   0.0000   0.0000
   2   0.0000   0.2500   0.0000   0.1667
   3   0.0000   0.0000   0.0000   0.0000
   4   0.0000   0.0000   0.0000   0.3333

                    x
  0.167     0.500     0.000     1.000
```

6.2.8. Процедуры с вещественной несимметрической разреженной матрицей

Рассматриваемые в разделе процедуры перечислены в табл. 6.1.

6.2.8.1. Вычисление LU-разложения разреженной матрицы

Выполняет подпрограмма LFTXG (DLFTXG):

CALL LFTXG(*n, nz, a, irow, jcol, iparam, rparam, nfac, nl, fac, irfac, jcfac, ipvt, jpvt*)

Входные данные: *n, nz, a, irow, jcol*.

Входные/выходные данные: *iparam, rparam, nfac, irfac, jcfac, ipvt, jpvt*.

Выходные данные: *nl*, *fact*.

Смысл параметров процедуры поясняется в табл. 6.5.

Автоматически для решения предоставляется память:

- $15n + 5maxnz$ байт в случае LFTXG;
- $15n + 6maxnz$ байт в случае DLFTXG, где *maxnz* – максимальное число ненулевых элементов на стадиях работы алгоритма исключения Гаусса. При отсутствии информации рекомендуется принять $maxnz = 3nz$.

Установка в *iparam* и *rparam* задаваемых по умолчанию параметров происходит, если задать *iparam*(1) = 0. Если надо установить иные параметры, выполните до вызова LFTXG вызов

CALL L4LXG(*iparam*, *rparam*)

Описание векторов *iparam* и *rparam*:

Вектор *iparam*:

Входные величины: *iparam*(1), *iparam*(2), *iparam*(3), *iparam*(5).

Входные/выходные величины: *iparam*(4).

iparam(1) – флаг инициализации.

iparam(2) – стратегия перестановок; если в *iparam*(2) записать

- 1, то ведущий элемент ищется методом Марковица по рядам;
- 2, то ведущий элемент ищется методом Марковица по столбцам;
- 3, то выполняется симметрический поиск Марковица.

Значение по умолчанию – 3.

iparam(3) – число рядов, в которых количество ненулевых элементов наименьшее и в которых выполняется поиск ведущих элементов; значение по умолчанию – 3.

iparam(4) – максимальное число ненулевых элементов в матрице A на любом шаге метода исключения Гаусса (выходная величина).

iparam(5) – максимальный размер рабочей области (должен превышать $19n + 5nz$ байт); если

- *iparam*(5) = 0, то выделяется заданная по умолчанию память;
- *iparam*(5) > 0, то рабочая область будет равна *iparam*(5).

iparam(6) -- в LFTXG не используется.

Вектор *rparam*:

Входные величины: *rparam*(1), *rparam*(2), *rparam*(3).

Выходные величины: *rparam*(4), *rparam*(5).

rparam(1) – верхний предел коэффициента роста; вычисления будут прекращены, если коэффициент роста превысит *rparam*(1); значение по умолчанию – 1016.

rparam(2) – коэффициент стабильности; абсолютное значение ведущего элемента должно превышать значение модуля элемента в ряду матрицы A, разделенного на *rparam*(2); значение по умолчанию – 10.0.

rparam(3) – допуск; элемент нижней треугольной матрицы L из нее удаляется (приравнивается нулю), если его абсолютное значение в процессе реализации метода исключения Гаусса станет меньше допуска; значение по умолчанию – 0.0.

rparam(4) – коэффициент роста; принимается равным наибольшему по абсолютной величине элементу матрицы A на всех этапах выполнения метода исключения Гаусса, деленному на наибольший по абсолютной величине элемент в первоначальной матрице A. Большое значение коэффициента роста означает, что решение может содержать ощутимую ошибку.

rparam(5) – значение наименьшего по абсолютной величине ведущего элемента.

Тип *rparam* в случае DL4LXG – REAL(8).

Описание

Пусть решается система линейных уравнений $Ax = b$, в которой A – разреженная $n \times n$-матрица. Пусть число ненулевых элементов в A равно nz. Для хранения координат ненулевых элементов используем 2 целочисленных вектора *irow* и *jcol* размера nz, устроив их так, что $A_{irow(i),jcol(i)} = a(i)$, $i = 1, ..., nz$.

Подпрограмма LFTXG выполняет LU-разложение матрицы A. По умолчанию для выбора ведущих элементов используется симметрическая стратегия Марковица [33], направленная как на снижение числа ненулевых элементов на промежуточных этапах метода исключения, так и на стабилизацию коэффициента роста. Для задач, с ярко выраженным сосредоточением ненулевых элементов в рядах или столбцах матрицы A, предусмотрены задаваемые *iparam*(2) стратегии. Результат разложения описывается равенством $PAQ = LU$, в котором P и Q – соответственно матрицы перестановок в рядах и столбцах матрицы, выполняемых в процессе LU-разложения, A – исходная матрица, L и U – соответственно нижняя и верхняя треугольные матрицы разложения [38].

После получения LU-разложения решение системы $Ax = b$ выполняется в 3 этапа: из системы $Lz = Pb$ определяется вектор z, далее решается система $Uy = z$ и находится вектор y, в завершение вычисляется $x = Qy$.

Пример. Ищется LU-разложение 6×6-матрицы

$$A=\begin{pmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & -3 & -1 & 0 & 0 \\ 0 & 0 & 15 & 0 & 0 & 0 \\ -2 & 0 & 0 & 10 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{pmatrix} \qquad (6.1)$$

```
program sparse                                  ! Поиск LU-разложения разреженной матрицы
 use msimsl, nouse => fac
 integer(4), parameter :: n = 6, nz = 15
 integer(4) :: iparam(6), irow(nz), jcol(nz), nfac, nl, irfac(3 * nz), jcfac(3 * nz), ipvt(n), jpvt(n)
 real(4) :: rparam(5), a(nz), fac(3 * nz)
 ! Координаты ненулевых элементов разместим в векторах irow(1:15) и jcol(1:15):
 irow    = (/  1,  2,  2,  2,  3,  4,  4,  4,  5,  5,  5,  5,  6,  6,  6  /)
 jcol    = (/  1,  2,  3,  4,  3,  1,  4,  5,  1,  4,  5,  6,  1,  2,  6  /)
 a       = (/ 10, 10, -3, -1, 15, -2, 10, -1, -1, -5,  1, -3, -1, -2,  6  /)
 nfac = 3 * nz; iparam(1) = 0                   ! Используем заданные по умолчанию параметры
 call Lftxg(n, nz, a, irow, jcol, iparam, rparam, nfac, nl, fac, irfac, jcfac, ipvt, jpvt)
 call wrrrn(' fac ', 1, nfac, fac, 1, 0)        ! Вывод результата
 call wrirn(' irfac ', 1, nfac, irfac, 1, 0);  call wrirn(' jcfac ', 1, nfac, jcfac, 1, 0)
 call wrirn(' p ', 1, n, ipvt, 1, 0);          call wrirn(' q ', 1, n, jpvt, 1, 0)
end program sparse
```

Результат:

```
                                   Fac
   1       2       3       4       5       6       7       8       9      10
 -.20   -1.00    -.10    -.20  -10.00   -2.00    -.10   98.00   10.00   -1.00
  11      12      13      14      15      16
5.00   -3.00   10.00   -1.00   10.00   15.00

                                  Irfac
 3   4   4   5   6   6   6   6   5   5   4   4   3   3   2   1
                                  Jcfac
 2   3   2   1   5   4   2   6   6   5   5   4   5   3   2   1

            p                                   q
 3   1   4   5   2   6                   3   1   5   6   4   2
```

6.2.8.2. Решение системы линейных уравнений с применением ранее найденного LU-разложения

Выполняется подпрограммой LFSXG (DLFSXG):

CALL LFSXG(*n*, *nfac*, *nl*, *fac*, *irfac*, *jcfac*, *ipvt*, *jpvt*, *b*, *ipath*, *x*)

Входные данные: *n*, *nfac*, *nl*, *fac*, *irfac*, *jcfac*, *ipvt*, *jpvt*, *ipath*, *b*.

Выходные данные: *x*.

Смысл параметров процедуры пояснен в табл. 6.5.

Применяемый алгоритм описан в предшествующем разделе.

Пример. Решим систему с приведенной в предшествующем разделе разреженной матрицей (6.1), имеющей две правые части $b^{(1)} = (10, 7, 45, 33, -34, 31)^T$ и $b^{(2)} = (60, 35, 60, 16, -22, -10)^T$.

```
program sparse2
 use msimsl, nouse => fac
 integer(4), parameter :: n = 6, nz = 15, ipath = 1
 integer(4) :: iparam(6), irow(nz), jcol(nz), nfac, nl, irfac(3 * nz), jcfac(3 * nz), ipvt(n), jpvt(n)
 real(4) :: rparam(5), a(nz), fac(3 * nz), x(n), b(n, 2)
 character(2) :: title(2) = (/ 'x1', 'x2' /)
 character(10) :: rlabel(1) = (/ 'none' /), clabel(1) = (/ 'none' /)
 ! Координаты ненулевых элементов разместим в векторах irow(1:15) и jcol(1:15):
 irow    = (/   1,   2,   2,   2,   3,   4,   4,   4,   5,   5,   5,   5,   6,   6,   6 /)
 jcol    = (/   1,   2,   3,   4,   3,   1,   4,   5,   1,   4,   5,   6,   1,   2,   6 /)
 a       = (/  10,  10,  -3,  -1,  15,  -2,  10,  -1,  -1,  -5,   1,  -3,  -1,  -2,   6 /)
 b(:, 1) = (/  10,   7,  45,  33, -34,  31 /)
 b(:, 2) = (/  60,  35,  60,  16, -22, -10 /)
 nfac = 3 * nz; iparam(1) = 0                    ! Используем заданные по умолчанию параметры
 ! Находим LU-разложение
 call Lftxg(n, nz, a, irow, jcol, iparam, rparam, nfac, nl, fac, irfac, jcfac, ipvt, jpvt)
 do i = 1, 2                                     ! Решаем линейные системы Ax = b(1) и Ax = b(2)
  call Lfsxg(n, nfac, nl, fac, irfac, jcfac, ipvt, jpvt, b(1, i), ipath, x)
  call wrrrl(title(i), 1, n, x, 1, 0, '(f4.1)', rlabel, clabel)          ! Вывод результата
 end do
end program sparse2
```

Результат:

```
                x1
  1.0   2.0   3.0   4.0   5.0   6.0
                x2
  6.0    50   4.0   3.0   2.0   1.0
```

6.2.8.3. Решение системы линейных уравнений

Выполняется подпрограммой LSLXG (DLSLXG):

CALL LSLXG(*n*, *nz*, *a*, *irow*, *jcol*, *b*, *ipath*, *iparam*, *rparam*, *x*)

Входные данные: *n*, *nz*, *a*, *irow*, *jcol*, *b*, *ipath*.

Входные/выходные данные: *iparam*, *rparam*.

Выходные данные: *b*.

Смысл параметров процедуры поясняется в табл. 6.5. Описание параметров *iparam* и *rparam* и способы изменения заданных по умолчанию настроек даны в разд. 6.2.8.1. Процедурой используется также и значение *iparam*(6), которое следует задать отличным от нуля, с тем чтобы выполнялось итерационное уточнение решения. Значение по умолчанию – 0.

Описание:

Первоначально LSLXG использует LFTXG и находит *LU*-разложение матрицы коэффициентов решаемой системы уравнений. Само же решение вычисляется LSLXG в результате вызова подпрограммы LFSXG. Подробно алгоритм изложен в [33] и в [38].

Пример. Решим систему с приведенной в разд. 6.2.8.1 разреженной матрицей (6.1).

```
program sparse3
 use msimsl
 integer(4), parameter :: n = 6, nz = 15, ipath = 1
 integer(4) :: iparam(6), irow(nz), jcol(nz)
 real(4) :: rparam(5), a(nz), x(n), b(n)
 ! Координаты ненулевых элементов разместим в векторах irow(1:15) и jcol(1:15):
 irow  = (/  1,  2,  2,  2,  3,  4,  4,  4,  5,  5,  5,  5,  6,  6,  6 /)
 jcol  = (/  1,  2,  3,  4,  3,  1,  4,  5,  1,  4,  5,  6,  1,  2,  6 /)
 a     = (/ 10, 10, -3, -1, 15, -2, 10, -1, -1, -5,  1, -3, -1, -2,  6 /)
 b     = (/ 10,  7, 45, 33, -34, 31 /)
 call L4lxg (iparam, rparam)                ! Меняем заданные по умолчанию настройки
 ! Задаем явно размер рабочей области; iparam(5) > 19n + 5nz = 189
 iparam(5) = 200                            ! Решаем линейную систему Ax = b
 call Lslxg (n, nz, a, irow, jcol, b, ipath, iparam, rparam, x)
 call wrrrn(' x ', 1, n, x, 1, 0)           ! Вывод результата
end program sparse3
```

Результат:

```
                    x1
1.000    2.000    3.000    4.000    5.000    6.000
```

6.3. ПРОЦЕДУРЫ С ВЕЩЕСТВЕННОЙ СИММЕТРИЧЕСКОЙ МАТРИЦЕЙ

6.3.1. Список процедур

Раздел содержит описания процедур с вещественной симметрической матрицей, которая может быть:

- положительно определенной;
- положительно определенной ленточной;
- положительно определенной разреженной;
- неотрицательно определенной;
- неопределенной;
- произвольной.

Процедуры перечислены в табл. 6.6. В табл. 6.7 дан синтаксис их вызова. Все процедуры являются подпрограммами. Как и для несимметрической

матрицы, приводится имя процедуры, работающей с одинарной точностью. При этом следует помнить, что каждой процедуре (кроме LSCXD) соответствует процедура, использующая двойную точность. Ее имя начинается с префикса D. Также опускаются имена процедур 2-го уровня.

Таблица 6.6. Процедуры с вещественной симметрической матрицей

Подпрограмма	*Назначение*	*Аналог*
	Симметрическая положительно определенная матрица	
LFCDS	Вычисляет R^TR-разложение матрицы и выполняет оценку ее числа обусловленности	LFCRG
LFTDS	Вычисляет R^TR-разложение матрицы без оценки ее числа обусловленности	LFTRG
LFSDS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное подпрограммой LFCDS или LFTDS R^TR-разложение матрицы A	LFSRG
LFIDS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^TR-разложение матрицы A, и выполняет итерационное уточнение решения	LFIRG
LSLDS	Вычисляет R^TR-разложение матрицы A и решает систему линейных уравнений $Ax = b$	LSLRG
LSADS	Вычисляет R^TR-разложение матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	LSARG
LINDS:	Вычисляет R^TR-разложение матрицы A, а затем находит матрицу A^{-1}, обратную матрице A	LINRG
	Симметрическая положительно определенная ленточная матрица	
LSLPB	Ищет R^TDR-разложение матрицы и решает систему $Ax = b$	-
LFCQS	Вычисляет R^TR-разложение матрицы и выполняет оценку ее числа обусловленности	LFCRG
LFTQS	Вычисляет R^TR-разложение матрицы без оценки ее числа обусловленности	LFTRG
LFSQS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное подпрограммой LFCQS или LFTQS R^TR – разложение матрицы A	LFSRG
LFIQS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^TR-разложение матрицы A, и выполняет итерационное уточнение решения	LFIRG
LSLQS	Вычисляет R^TR-разложение матрицы A и решает систему линейных уравнений $Ax = b$	LSLRG

LSAQS	Вычисляет R^TR-разложение матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	LSARG
Симметрическая положительно определенная разреженная матрица		
PCGRC	Решает систему линейных уравнений $Ax = b$ методом сопряженных градиентов с аппроксимацией и обратной связью	-
JCGRC	Решает систему линейных уравнений $Ax = b$ методом сопряженных градиентов с аппроксимацией Якоби и обратной связью	PCGRC
LSCXD	Вычисляет структуру данных для представления нижней треугольной матрицы L из LL^T-разложения Холецкого матрицы A	-
LNFXD	Находит числовое LL^T-разложение Холецкого матрицы A	-
LFSXD	Решает систему линейных уравнений $Ax = b$ с использованием ранее найденного LL^T-разложения Холецкого матрицы A	-
LSLXD	Решает систему линейных уравнений $Ax = b$ методом Холецкого	-
Симметрическая неотрицательно определенная матрица		
CHFAC	Вычисляет R^TR-разложение Холецкого матрицы A	-
LCHRG	Вычисляет R^TR-разложение Холецкого матрицы A с применением схемы частичного выбора	-
Симметрическая неопределенная матрица		
LFCSF	Вычисляет UDU^T-разложение матрицы A и выполняет оценку ее числа обусловленности	LFCRG
LFTSF	Вычисляет UDU^T-разложение матрицы A без оценки ее числа обусловленности	LFTRG
LFSSF	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное подпрограммой LFCSF или LFTSF UDU^T-разложение матрицы A	LFSRG
LFISF	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное UDU^T-разложение матрицы A, и выполняет итерационное уточнение решения	LFIRG
LSLSF	Вычисляет UDU^T-разложение матрицы A и решает систему линейных уравнений $Ax = b$	LSLRG
LSASF	Вычисляет UDU^T-разложение матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	LSARG
Симметрическая произвольная матрица		
MCHOL	Вычисляет R^TR-разложение Холецкого матрицы A с добавлением (при необходимости) к ней диагональной матрицы	-

Таблица 6.7. Синтаксис вызова подпрограмм из табл. 6.6

Синтаксис вызова	*Автоматически выделяемая память*
Симметрическая положительно определенная матрица	
CALL LFCDS(*n*, *a*, *Lda*, *fac*, *Ldfac*, *rcond*)	*n*
CALL LFTDS(*n*, *a*, *Lda*, *fac*, *Ldfac*)	-
CALL LFSDS(*n*, *fac*, *Ldfac*, *b*, *x*)	-
CALL LFIDS(*n*, *a*, *Lda*, *fac*, *Ldfac*, *b*, *x*, *res*)	-
CALL LSLDS(*n*, *a*, *Lda*, *b*, *x*)	$n^2 + n$
CALL LSADS(*n*, *a*, *Lda*, *b*, *x*)	$n^2 + 2n$
CALL LINDS(*n*, *a*, *Lda*, *ainv*, *Ldainv*)	*n*
Симметрическая положительно определенная ленточная матрица	
CALL LSLPB(*n*, *a*, *Lda*, *ncoda*, *ijob*, *u*)	*ncoda*
CALL LFCQS(*n*, *a*, *Lda*, *ncoda*, *fac*, *Ldfac*, *rcond*)	*n*
CALL LFTQS(*n*, *a*, *Lda*, *ncoda*, *fac*, *Ldfac*)	-
CALL LFSQS(*n*, *fac*, *Ldfac*, *ncoda*, *b*, *x*)	-
CALL LFIQS(*n*, *a*, *Lda*, *ncoda*, *fac*, *Ldfac*, *b*, *x*, *res*)	-
CALL LSLQS(*n*, *a*, *Lda*, *ncoda*, *b*, *x*)	$n(ncoda + 1) + n$
CALL LSAQS(*n*, *a*, *Lda*, *ncoda*, *b*, *x*)	То же
Симметрическая положительно определенная разреженная матрица	
CALL PCGRC(*ido*, *n*, *x*, *p*, *r*, *z*, *relerr*, *itmax*)	$8itmax$
CALL JCGRC(*ido*, *n*, *diag*, *x*, *p*, *r*, *z*, *relerr*, *itmax*)	"
CALL LSCXD(*n*, *nz*, *irow*, *jcol*, *ijob*, *itwksp*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *ilnz*, *iper*, *invper*, *ispace*)	$10n + 12nz + 7$
CALL LNFXD(*n*, *nz*, *a*, *irow*, *jcol*, *ijob*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *ilnz*, *iper*, *invper*, *ispace*, *itwksp*, *diag*, *rlnz*, *rparam*)	$3n + 3nz$
CALL LFSXD(*n*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *rlnz*, *ilnz*, *diag*, *iper*, *b*, *x*)	-
CALL LSLXD(*n*, *nz*, *a*, *irow*, *jcol*, *b*, *itwksp*, *x*)	$18n + 21nz + 9$
Симметрическая неотрицательно определенная матрица	
CALL CHFAC(*n*, *a*, *Lda*, *tol*, *irank*, *r*, *Ldr*)	-
CALL LCHRG(*n*, *a*, *Lda*, *pivot*, *ipvt*, *fac*, *Ldfac*)	-
Симметрическая неопределенная матрица	
CALL LFCSF(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *rcond*)	*n*
CALL LFTSF(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*)	"

CALL LFSSF(*n*, *fac*, *Ldfac*, *ipvt*, *b*, *x*)	-
CALL LFISF(*n*, *a*, *Lda*, *fac*, *Ldfac*, *ipvt*, *b*, *x*, *res*)	-
CALL LSLSF(*n*, *a*, *Lda*, *b*, *x*)	$n^2 + 2n$
CALL LSASF(*n*, *a*, *Lda*, *b*, *x*)	То же
Симметрическая произвольная матрица	
CALL MCHOL(*n*, *a*, *Lda*, *tol*, *irank*, *r*, *Ldr*, *dmax*, *ind*)	-

Из списка параметров видно, что все подпрограммы с симметрической матрицей не имеют параметра *ipath*, позволяющего при работе с нессиметрическими матрицами решать либо систему $Ax = b$, либо систему $A^T x = b$. Для симметрической матрицы *ipath* не имеет смысла, поскольку по определению для нее $A = A^T$.

6.3.2. Возможные ошибки процедур с вещественной симметрической матрицей

В табл. 6.8 указаны ошибки, которые могут возникать при вызове процедур, работающих с вещественной симметрической матрицей. Для экономии места в последнем столбце табл. 6.8 приводится подпрограмма, соответствующая типу REAL(4); имена сопутствующих ей подпрограмм, в которых возникают те же ошибки, опускаются.

Таблица 6.8. Возможные ошибки процедур из табл. 6.6

Тип	*Код*	*Причина ошибки*	*Где возникает*
3	1	Входная матрица алгоритмически вырожденная	LFCSG, LFCSF
3	1	Входная матрица плохо обусловлена; решение может быть неточным	LSLSG, LSASG, LSLDS, LSADS, LINDS, LSLQS, LSAQS
3	1	Входная матрица не является неотрицательно определенной с допуском, заданным параметром *tol*	CHFAC
3	2	Входная матрица плохо обусловлена; итерационное уточнение не может быть эффективным	LFISG, LFIDS
3	3	Входная матрица алгоритмически вырожденная	LFCQS
3	4	Входная матрица плохо обусловлена; итерационное уточнение не может быть эффективным	LFIQS
4	1	Входная матрица является вырожденной	LFSDS, LFSQS, LSCXD, LFSXD

4	1	Аппроксимирующая матрица является вырожденной	PCGRC
4	1	Диагональ входной матрицы содержит нуль	JCGRC
4	1	Входная матрица не является положительно определенной	LNFXD, LSLXD
4	2	Аппроксимирующая матрица не является положительно определенной	PCGRC
4	2	Диагональные элементы матрицы имеют разные знаки	JCGRC
4	2	Входная матрица является вырожденной	LFCSG, LFTSG, LSLSG, LSASG
4	2	Входная матрица не является положительно определенной	LFCDS, LFTDS, LSLDS, LSADS, LINDS, LSLPB, LFCQS, LFTQS, LSLQS, LSAQS
4	2	В матрице коэффициентов есть столбец, в котором одни нули	LNFXD, LSLXD
4	3	Входная матрица не является положительно определенной	PCGRC, JCGRC
4	4	Входная матрица является вырожденной	PCGRC, JCGRC
4	5	Система не сошлась после выполнения *itmax* итераций	PCGRC

6.3.3. Параметры процедур с вещественной симметрической и комплексной эрмитовой матрицами

Параметры процедур с вещественной симметрической и комплексной эрмитовой полными и ленточными положительно определенными матрицами и неопределенными матрицами совпадают по смыслу с одноименными параметрами процедур-аналогов и приведены в табл. 6.4.

Параметры процедур с вещественной симметрической положительно определенной разреженной матрицей даны в разделах с описаниями этих процедур.

Процедуры с комплексной эрмитовой положительно определенной разреженной матрицей имеют те же параметры, что и их вещественные аналоги.

6.3.4. Процедуры с вещественной симметрической положительно определенной матрицей

6.3.4.1. Об использовании процедур

Процедуры с вещественной симметрической положительно определенной матрицей выполняют и/или используют R^TR-разложение матрицы по методу Холецкого. Список процедур приведен в табл. 6.6. В последней гра-

фе таблицы указаны подпрограммы-аналоги, применяемые для работы с вещественной несимметрической общего вида матрицей.

Между аналогами и подпрограммами табл. 6.6 есть отличия:

- подпрограммы с симметрической положительно определенной матрицей используют ее R^TR-разложение, а подпрограммы-аналоги – LU-разложение матрицы общего вида;
- при выполнении разложения подпрограммы из табл. 6.6, созданные для положительно определенной матрицы, не применяют перестановки строк и столбцов матрицы; поэтому их вызовы не содержат параметров *pivot* и *ipvt*;
- отсутствует параметр *ipath*;
- подпрограммы LFCDS, LFTDS, LSLDS, LSADS и LINDS, выполняющие R^TR-разложение симметрической положительно определенной матрицы, прекращают работу, когда входная матрица не является положительно определенной.

Смысл используемых с подпрограммами параметров поясняется в разделах, содержащих описания их аналогов, и в табл. 6.4.

6.3.4.2. Примеры для процедур с симметрической положительно определенной матрицей

Пример для LFCDS. Вычисляется оценка числа обусловленности симметрической $n \times n$-матрицы Гильберта, в которой $a_{ij} = 1/(i + j - 1)$, для разных значений n.

```
program Gilb
 use msimsl
 integer(4):: n, i, j
 real(4), allocatable :: a(:, :)
 real(4) :: rcond
 do n = 2, 7
  allocate(a(n, n))                    ! Выделяем память под массив a
  do j = 1, n                          ! Задаем матрицу Гильберта
   a(:, j) = (/ (1.0 / real(i + j - 1), i = 1, n) /)
   end do
 ! Находим RTR-разложение и оценку числа обусловленности матрицы A
  call Lfcds(n, a, n, a, n, rcond)
  deallocate(a)
  print *, 'cond1(A) = ', 1.0 / rcond  ! Вывод оценки числа обусловленности матрицы A
 end do
end program Gilb
```

Результат:

n	2	3	4	5	6	7
cond1(A)	22.83	687.15	21672.2	697940.5	2.246E+07	4.416E+08

Начиная с $n = 6$ матрица становится алгоритмически вырожденной, поэтому работа LFCDS станет сопровождаться сообщениями типа

```
*** WARNING ERROR 1 from LFCDS. The matrix is algorithmically singular.
***        An estimate of the reciprocal of its L1 condition number is
***        RCOND = 4.452485E-08.
```

При $n = 10$ матрица перестает быть положительно определенной, и LFCDS выдаст сообщение и прекратит работу.

```
*** FATAL ERROR 2 from LFCDS. The leading 10 by 10 submatrix of the
***        input matrix is not positive definite.
```

Пример для LFCDS и LFIDS. Процедурой LFCDS вычисляется R^TR-разложение симметрической положительно определенной матрицы A и выполняется оценка ее числа обусловленности. Процедура LFIDS применяется для нахождения A^{-1}.

Порядок вычислений:

1) найти, используя LFCDS, R^TR-разложение матрицы A;
2) вызвать n раз LFIDS, находя в каждом вызове очередной столбец A^{-1}.

```
program useRRT
 use msimsl, nouse => fac
 integer(4), parameter :: n = 3, Lda = n, Ldfac = n
 integer(4) :: j
 real(4) :: a(Lda, Lda), ainv(Lda, Lda), fac(Ldfac, Ldfac), rcond, res(n), rj(n)
 a = reshape((/      1.0,     -3.0,      2.0,                    &
                    -3.0,     10.0,     -5.0,                    &
                     2.0,     -5.0,      6.0    /), shape = (/ Lda, Lda /))
 ! Находим RTR-разложение и оценку числа обусловленности матрицы A
 call Lfcds(n, a, Lda, fac, Ldfac, rcond)
 print *, 'cond1(A) = ', 1.0 / rcond            ! Вывод оценки числа обусловленности матрицы A
 rj = 0.0
 do j=1, n
  rj(j) = 1.0          ! rj(j) – столбец с номером j единичной матрицы, поэтому LFIDS
                       ! разместит столбец j обратной матрицы A-1 в j-й столбец массива ainv
  call Lfids(n, a, Lda, fac, Ldfac, rj, ainv(1, j), res)
  rj(j) = 0.0
 end do
 call wrrrn('ainv', n, n, ainv, Lda, 0)         ! Вывод результата
end program useRRT
```

Результат:

```
cond1(A) =    673.838600
```

```
                ainv
              1          2          3
  1      35.000      8.000     -5.000
  2       8.000      2.000     -1.000
  3      -5.000     -1.000      1.000
```

Пример для LFTDS и LFSDS. Решить систему линейных уравнений с m правыми частями.

Порядок решения:

1) найти, используя LFTDS, R^TR-разложение симметрической матрицы A;
2) вызвать m раз LFSDS, находя в каждом вызове решение системы для очередной правой части.

```
program useRRT2
 use msimsl, nouse => fac
 integer(4), parameter :: n = 3, Lda = n, Ldfac = n, m = 2
 integer(4) :: j
 real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), b(n, m), x(n, m)
  a = reshape((/      1.0,     -3.0,      2.0,                    &
                     -3.0,     10.0,     -5.0,                    &
                      2.0,     -5.0,      6.0    /), shape = (/ Lda, Lda /))
  b = reshape((/      1.0,      4.0,     -1.0,                    &
                     10.0,     14.0,      9.0    /), shape = (/ n, m /))
 call Lftds(n, a, Lda, fac, Ldfac)              ! Находим R^TR-разложение матрицы A
 do j=1, m                                      ! Решаем систему с несколькими правыми частями
   call Lfsds(n, fac, Ldfac, b(1, j), x(1, j))
 end do
 call wrrrn('x', n, m, x, n, 0)                 ! Вывод результата
end program useRRT2
```

Результат (решение содержит каждый столбец результата):

```
                x
              1          2
  1      72.000    417.500
  2      17.000     99.000
  3     -10.000    -55.000
```

Пример для LFIDS, решающей с применением ранее найденного R^TR-разложения симметрической матрицы A систему линейных уравнений $Ax = b$, выполняющей итерационное уточнение решения и вычисляющей невязку.

В примере несколько раз решается система уравнений с одинаковой левой и разными правыми частями. Каждая новая правая часть получается в результате увеличения второго элемента вектора b на 0.2. Сразу после решения выводится вычисленная в LFIDS невязка *res*.

```
program useRRT3
 use msimsl, nouse => fac
```

```
integer(4), parameter :: n = 3, Lda = n, Ldfac = n
integer(4) :: j
real(4) :: a(Lda, Lda), b(n), x(n, 3), fac(Lda, Lda), res(n, 3), rcond
 a = reshape((/      1.0,     -3.0,      2.0,                  &
                    -3.0,     10.0,     -5.0,                  &
                     2.0,     -5.0,      6.0    /), shape = (/ Lda, Lda /))
b = (/ 4.1, -4.7, 6.5 /)
! Находим R^TR-разложение и оценку числа обусловленности матрицы A
call Lfcds(n, a, Lda, fac, Ldfac, rcond)
print *, 'cond1(A) = ', 1.0 / rcond          ! Вывод оценки числа обусловленности матрицы A
do j =1, 3                                   ! Решаем систему с разными правыми частями
  call Lfids(n, a, Lda, fac, Lda, b, x(1, j), res(1, j))
  b(2) = b(2) + 0.2
end do
call wrrrn('x', n, 3, x, n, 0)               ! Вывод результата
call wrrrn('res', n, 3, res, n, 0)           ! Вывод невязки
end program useRRT3
```

Результат (каждый столбец первой таблицы содержит решение, а второй – соответствующие ему невязки):

```
cond1(A) =     673.838600

                    x
              1          2          3
   1      73.40      75.00      76.60
   2      16.90      17.30      17.70
   3      -9.30      -9.50      -9.70

                          res
                  1                  2
   1      3.338E-06          3.338E-06          3.338E-06
   2      0.000E+00          0.000E+00          0.000E+00
   3      4.768E-07          4.768E-07          4.768E-07
```

Пример для LSLDS и LSADS, вычисляющих R^TR-разложение симметрической матрицы A и решающих систему линейных уравнений $Ax = b$. Дополнительно LSADS выполняет итерационное уточнение решения.

```
program useLsl
 use msimsl
 integer, parameter :: Lda = 3, n = 3
 real :: a(Lda, Lda), b(n), x(n)
  a = reshape((/      1.0,     -3.0,      2.0,                  &
                     -3.0,     10.0,     -5.0,                  &
                      2.0,     -5.0,      6.0    /), shape = (/ Lda, Lda /))
b = (/ 4.1, -4.7, 6.5 /)
 call Lslds(n, a, Lda, b, x)                 ! Решаем линейную систему Ax = b
```

```
! Тот же результат для заданных данных даст и вызов LSADS
! call Lsads(n, a, Lda, b, x)
 call wrrrn('x', 1, n, x, 1, 0)                    ! Вывод результата
end program useLsl
```

Результат:

```
                    x
   73.40          16.90          -9.30
```

Пример для LINDS, вычисляющей R^TR-разложение симметрической матрицы A и находящей затем A^{-1}.

```
program useLinds
 use msimsl
 integer, parameter :: Lda = 3, Ldainv = 3, n = 3
 real :: a(Lda, Lda), ainv(Ldainv, Ldainv)
  a = reshape((/      1.0,    -3.0,     2.0,                    &
                     -3.0,    10.0,    -5.0,                    &
                      2.0,    -5.0,     6.0    /), shape = (/ Lda, Lda /))
 call Linds(n, a, Lda, ainv, Ldainv)            ! Вычисляем A-1
 call wrrrn('ainv', n, n, ainv, Ldainv, 0)      ! Вывод результата
end program useLinds
```

Результат:

```
                    ainv
                 1          2          3
   1         35.00       8.00      -5.00
   2          8.00      2.000     -1.000
   3         -5.00     -1.000      1.000
```

6.3.5. Процедуры с вещественной симметрической положительно определенной ленточной матрицей

6.3.5.1. Об использовании процедур

Процедуры с вещественной симметрической положительно определенной ленточной матрицей выполняют и/или используют R^TR-разложение матриц по Холецкому. Список процедур приведен в табл. 6.7. Между аналогами и подпрограммами раздела существуют те же, что и перечисленные в разд. 6.3.4.1, отличия.

Смысл используемых с подпрограммами параметров поясняется в разделах, содержащих описания их аналогов, и в табл. 6.4. В процедурах, кроме LSLPB, массивы *a* и *fac* содержат соответственно ленточное представление симметрической матрицы и ее R^TR-разложение; массив *a* имеет форму (Lda, n), а *fac* – $(Ldfac, n)$, где $Lda \geq ncoda + 1$ и $Ldfac \geq ncoda + 1$.

Как и ранее, процедуры, имеющие аналоги, иллюстрируются лишь примерами.

6.3.5.2. R^TDR-разложение Холецкого симметрической положительно определенной ленточной матрицы и решение системы $Ax = b$

Выполняется подпрограммой LSLPB (DLSLPB):

CALL LSLPB(*n*, *a*, *Lda*, *ncoda*, *ijob*, *u*)

Входные данные: *n*, *Lda*, *ncoda*, *ijob*.

Входные/выходные данные: *a*, *ipvt*, *rcond*.

Выходные данные: *u*.

n – порядок матрицы *A*; $n > 0$.

a – массив, содержащий кодиагональное ленточное представление положительно определенной симметрической $n \times n$-матрицы *A* и правую часть (элементы вектора *b*) уравнения $Ax = b$. Число столбцов массива должно быть не менее *ncoda* + 2. Вектор *b* хранится в столбце *ncoda* + 2 массива *a* и решение – вектор *x* – замещает в массиве *a* значения *b*.

Lda – ведущий размер массива *a*; $Lda \geq n + ncoda$.

ncoda – число верхних кодиагоналей матрицы *A*; $0 \leq ncoda < n$.

ijob – флаг, управляющей работой подпрограммы; если *ijob* равняется

- 1, то выполняется разложение матрицы *A* и решается система $Ax = b$;
- 2, то разложение матрицы *A* должно быть уже выполнено и подпрограмма, используя это разложение, решает систему $Ax = b$;
- 3, то осуществляется разложение матрицы; система $Ax = b$ не решается;
- 4, 5 или 6, то выполняется то же, что и при *ijob* = 3, но для повышения быстродействия не выполняется проверка значений *Lda*, *n*, *ncoda* и заполнение *u*.

u – массив флажков, указывающих на отсутствие положительной определенности главного диагонального минора; $u(i) = 0$ означает, что главный диагональный минор размера *i* не является положительно определенным, в противном случае $u(i) = 1$.

Описание:

Подпрограмма LSLPB находит разложение матрицы $A = R^TDR$, где *R* – верхняя треугольная матрица с единичной диагональю, а *D* – диагональная матрица, и решает систему линейных уравнений $Ax = b$. Для эффективности вычисляются и хранятся обратные величины диагональных элементов *D*. Если *D* имеет неположительный диагональный элемент, то возникнет ошибка. Ее причиной может быть либо близость матрицы *A* к вырожденной, либо отсутствие у нее положительной определенности.

Подпрограмма LSLPB эффективна при незначительной ширине ленты, поскольку для значений *ncoda* = 0, 1 и 2 в LSLPB созданы специальные быстрые циклы. При решении трехдиагональных систем с *ncoda* = 1 альтернативой LSLPB является LSLCR. Скорее всего, LSLCR окажется более произ-

водительной, чем LSLPB, на векторных или параллельных вычислителях. Но на скалярных и даже на параллельных компьютерах без оптимизирующих компиляторов LSLCR может уступать LSLPB.

Пример. Решается линейная система из четырех уравнений. Матрица коэффициентов является положительно определенной и имеет вещественную кодиагональ.

```
program useRTDR
 use msimsl
 integer(4), parameter :: n = 4, ncoda = 2, Lda = n + ncoda
 integer(4) :: ijob
 real(4) :: a(Lda, ncoda + 2), u(n)
 ! Кодиагональное ленточное представление симметрической матрицы A;
 ! в последнем столбце массива a – коэффициенты вектора b
 a = reshape((/      0.0,    0.0,    0.0,    0.0,                &
                     0.0,    0.0,    0.0,    0.0,                &
                     2.0,    0.0,    0.0,    6.0,                &
                     4.0,    0.0,    0.0,  -11.0,                &
                     7.0,    2.0,   -1.0,  -11.0,                &
                     3.0,   -1.0,    1.0,   19.0 /), shape = (/ Lda, ncoda + 2 /), order = (/ 2, 1 /))
 ! Находим R^TDR-разложение матрицы A и решаем линейную систему Ax = b.
 ijob = 1; call Lslpb(n, a, Lda, ncoda, ijob, u)
 ! Вывод результата – последнего столбца массива a
 call wrrrn('x', 1, n, a(ncoda + 1, ncoda + 2), 1, 0)
end program useRTDR
```

Результат:

```
                    ainv
     4.000       -6.000        2.000        9.000
```

6.3.5.3. Примеры для процедур с симметрической положительно определенной ленточной матрицей

Пример для LFCQS и LFTQS, вычисляющих R^TR-разложение симметрической матрицы A. Вдобавок LFCQS выполняет оценку числа обусловленности матрицы. Вычисляется A^{-1} – матрица, обратная симметрической $n\times n$-матрице A.

Порядок вычислений:

1) найти, используя LFCQS или LFTQS, R^TR-разложение матрицы A;
2) вызвать n раз LFIQS или LFSQS, находя в каждом вызове очередной столбец A^{-1}.

```
program useRRTB
 use msimsl, nouse => fac
 integer(4), parameter :: Lda = 2, Ldfac = 2, n = 4, ncoda = 1
 integer(4) :: j
```

```
real(4) :: a(Lda, n), ainv(n, n), fac(Ldfac, n), rcond, res(n), rj(n)
! Ленточное представление матрицы A
a = reshape((/        0.0,    1.0,    1.0,    1.0,                    &
                      2.0,    2.5,    2.5,    2.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
! Находим R^TR-разложение и оценку числа обусловленности матрицы A
call Lfcqs(n, a, Lda, ncoda, fac, Ldfac, rcond)
! То же разложение без оценки числа обусловленности даст LFTQS
! call Lftqs(n, a, Lda, ncoda, fac, Ldfac)
if(rcond > 0.0) then                    ! Если работаем с LFCQS
  print *, 'cond1(A) = ', 1.0 / rcond   ! Вывод оценки числа обусловленности матрицы A
end if
rj = 0.0
do j=1, n
  rj(j) = 1.0          ! rj(j) – столбец с номером j единичной матрицы, поэтому LFIQS
                       ! разместит столбец j обратной матрицы A^-1 в j-й столбец массива ainv
  call Lfiqs(n, a, Lda, ncoda, fac, Ldfac, rj, ainv(1, j), res)
  ! Те же результаты для ainv даст и LFSQS
  ! call Lfsqs(n, fac, Ldfac, ncoda, rj, ainv(1,j))
  rj(j) = 0.0
end do
call wrrrn('ainv', n, n, ainv, n, 0)    ! Вывод результата
end program useRRTB
```

Результат:

```
cond1(A) =      6.247648
```

ainv

	1	2	3	4
1	0.6667	-0.3333	0.1667	-0.0833
2	-0.3333	0.6667	-0.3333	0.1667
3	0.1667	-0.3333	0.6667	-0.3333
4	-0.0833	0.1667	-0.3333	0.6667

Пример для LFTQS и LFSQS. Решается система линейных уравнений с m правыми частями.

Порядок решения:

1) найти, используя LFTQS, R^TR-разложение симметрической матрицы A;
2) вызвать m раз LFSQS, находя в каждом вызове решение системы для очередной правой части.

```
program useRRT2
use msimsl, nouse => fac
integer(4), parameter :: Lda = 3, Ldfac = 3, n = 4, ncoda = 1, m = 4
integer(4) :: j
real(4) :: a(Lda, n), fac(Ldfac, n), b(n, m), x(n, m)
! Ленточное представление матрицы A
```

```
a = reshape((/      0.0,    0.0,   -1.0,    1.0,              &
                    1.0,    3.0,    2.0,    1.0,              &
                    2.0,    4.0,    7.0,    8.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
! Четыре правые части
b = reshape((/      4.0,   -3.0,    9.0,   11.0,              &
                    6.0,   10.0,   29.0,    3.0,              &
                   15.0,   12.0,   11.0,    6.0,              &
                   -7.0,    1.0,   14.0,    2.0 /), shape = (/ n, m /))
call Lftqs(n, a, Lda, ncoda, fac, Ldfac)      ! Находим R^TR-разложение матрицы A
do j=1, m                                     ! Решаем систему с несколькими правыми частями
  call Lfsqs(n, fac, Ldfac, ncoda, b(1, j), x(1, j))
end do
call wrrrn('x', n, m, x, n, 0)                ! Вывод результата
end program useRRT2
```

Результат (решение содержит каждый столбец результата):

	x			
	1	2	3	4
1	4.00	6.00	15.00	-7.00
2	-2.50	18.00	8.58	6.50
3	-4.50	44.50	13.50	18.50
4	15.50	-41.50	-7.50	-16.50

Пример для LFIQS, решающей с применением ранее найденного R^TR-разложения симметрической матрицы A систему линейных уравнений $Ax = b$, выполняющей итерационное уточнение решения и вычисляющей невязку.

В примере несколько раз решается система уравнений с одинаковой левой и разными правыми частями. Каждая новая правая часть получается в результате увеличения второго элемента вектора b на 0.2. Сразу после решения выводится вычисленная в LFIQS невязка *res*.

```
program useRRT3
 use msimsl, nouse => fac
integer(4), parameter :: Lda = 2, Ldfac = 2, n = 4, ncoda = 1
 integer(4) :: j
 real(4) :: a(Lda, n), fac(Ldfac, n), rcond, b(n), res(n, 3), x(n, 3)
 ! Ленточное представление матрицы A
 a = reshape((/      0.0,    1.0,    1.0,    1.0,              &
                     2.0,    2.5,    2.5,    2.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 b = (/ 3.0, 5.0, 7.0, 4.0 /)
 ! Находим R^TR-разложение и оценку числа обусловленности матрицы A
 call Lfcqs(n, a, Lda, ncoda, fac, Ldfac, rcond)
 print *, 'cond1(A) = ', 1.0 / rcond          ! Вывод оценки числа обусловленности матрицы A
 do j =1, 3                                   ! Решаем систему с разными правыми частями
  call Lfiqs(n, a, Lda, ncoda, fac, Lda, b, x(1, j)), res(1, j))
  b(2) = b(2) + 0.2
```

```
 end do
 call wrrrn('x', n, 3, x, n, 0)                    ! Вывод результата
 call wrrrn('res', n, 3, res, n, 0)                ! Вывод невязки
end program useRRT3
```

Результат (каждый столбец первой таблицы содержит решение, а второй – соответствующие ему невязки):

```
cond1(A) =      6.247648

                    x
              1          2          3
   1      1.167      1.100      1.033
   2      0.667      0.800      0.933
   3      2.167      2.100      2.033
   4      0.917      0.950      0.983

                          res
                 1                  2
   1      7.947E-08          7.947E-08          1.987E-08
   2      1.987E-08          7.947E-08         -3.974E-08
   3      7.947E-08          7.947E-08          7.947E-08
   4     -3.974E-08         -3.974E-08         -3.974E-08
```

Пример для LSLQS и LSAQS, вычисляющих R^TR-разложение симметрической матрицы A и решающих систему линейных уравнений $Ax = b$. Дополнительно LSAQS выполняет итерационное уточнение решения.

```
program useLsl
 use msimsl
 integer(4), parameter :: Lda = 3, Ldfac = 3, n = 4, ncoda = 1
 real(4) :: a(Lda, n), b(n), x(n)
 ! Ленточное представление матрицы A
 a = reshape((/      0.0,    0.0,   -1.0,    1.0,        &
                     1.0,    3.0,    2.0,    1.0,        &
                     2.0,    4.0,    7.0,    8.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 b = (/ 4.0, -3.0, 9.0, 11.0 /)
 call Lslqs(n, a, Lda, ncoda, b, x)                ! Решаем линейную систему Ax = b
 ! Тот же результат для заданных данных даст и вызов LSAQS
 ! call Lsaqs(n, a, Lda, ncoda, b, x)
 call wrrrn('x', 1, n, x, 1, 0)                    ! Вывод результата
end program useLsl
```

Результат:

```
                    x
   4.000        -2.50        -4.50        15.50
```

6.3.6. Процедуры с вещественной симметрической положительно определенной разреженной матрицей

6.3.6.1. Метод сопряженных градиентов с аппроксимацией и обратной связью

Решение системы линейных уравнений $Ax = b$ с симметрической положительно определенной матрицей A методом сопряженных градиентов с аппроксимацией (с предварительным приведением системы к виду, удобному для итераций) и обратной связью выполняется подпрограммой PCGRC(DPCGRC):

CALL PCGRC(*ido*, *n*, *x*, *p*, *r*, *z*, *relerr*, *itmax*)

Входные данные: *n*, *z*, *relerr*, *itmax*.

Входные/выходные данные: *ido*, *x*, *r*.

Выходные данные: *p*.

ido – флаг, задающий решаемую задачу. При первом вызове *ido* должен равняться нулю. Если подпрограмма завершается с *ido* = 1, то задается $z = Ap$, где A – матрица, и PCGRC вызывается повторно до тех пор, пока не вернет *ido*, равное двум. Если подпрограмма вернет *ido* = 2, то в z устанавливается решение системы $Mz = r$, где M – аппроксимирующая матрица, и PCGRC вызывается вновь. Завершение с *ido* = 3 означает сходимость итераций, и в этом случае x содержит решение.

n – порядок системы линейных уравнений.

x – вектор размера n. На входе x содержит начальные приближения, на выходе – решение системы.

p – вектор размера n. Используется для задания $z = Ap$.

r – вектор размера n. При первом вызове содержит правую часть системы. На выходе – невязку.

z – вектор размера n. Когда *ido* = 1, он содержит Ap; когда *ido* = 2, он содержит решение системы $Mz = r$; когда *ido* = 3, он игнорируется.

relerr – задаваемая на входе относительная ошибка.

itmax – максимально допустимое число итераций.

Автоматически для решения предоставляется память:

- 8*itmax* байт в случае PCGRC;
- 15*itmax* байт в случае DPCGRC.

Описание:

Подпрограмма PCGRC использует матрицу M, которая аппроксимирует матрицу A. Причем система $Mz = r$ легко решается. Заметим, однако, что хорошая аппроксимация и простота решения $Mz = r$ – конфликтующие требования.

Число итераций зависит от входной матрицы и заданной точности *relerr*. Когда $n >> 1$, можно задать $itmax = n^{0.5}$.

Запишем алгоритм, полагая, что M – аппроксимирующая матрица, b, p, r, x и z – векторы, а τ – заданная относительная погрешность.

1. $\lambda = -1$; $p_0 = x_0$; $r1 = b - Ap$
2. С параметром $k = 1$, *itmax* выполнить:

$z_k = M^{-1} r_k$

2.1. Если $k = 1$, то

$b_k = 1$; $p_k = z_k$

иначе

$\beta_k = z_k^T r_k / z_{k-1}^T r_{k-1}$; $p_k = z_k + \beta_k p_k$

конец если 2.1

$z_k = Ap$; $\alpha_k = z_{k-1}^T r_{k-1} / z_k^T p_k$

$x_k = x_k + \alpha_k p_k$; $r_k = r_k - \alpha_k z_k$

2.2. Если $\|z_k\|_2 \le \tau(1-\lambda)\|x_k\|_2$, то

Вычислить повторно λ.

Если $\|z_k\|_2 \le \tau(1-\lambda)\|x_k\|_2$, то выход из цикла 2.

конец если 2.2

конец цикла 2

Алгоритм детально описан в [6] и [48, Chapter 7]. В нем λ является оценкой наибольшего собственного значения итерационной матрицы $G = I - M^{-1}A$. Критерий завершения

$$\frac{\|x_k - x\|_M}{\|x\|_M} \le \frac{1}{1-\lambda_{\max}(G)} \frac{\|z_k\|_M}{\|x_k\|_M},$$

где $\|x\|_M^2 = x^T M x$. Он базируется на результатах, приведенных в [48, p. 148-151].

Относительно собственных значений известно, что $\lambda_{\max}(T_1) \le \lambda_{\max}(T_2) \le \ldots \le \lambda_{\max}(G) < 1$, где T – симметрическая трехдиагональная матрица:

$$T_n = \begin{pmatrix} \mu_1 & \omega_2 & & & \\ \omega_2 & \mu_2 & \omega_3 & & \\ & \omega_3 & \mu_3 & \omega_4 & \\ & & \ddots & \ddots & \ddots \end{pmatrix};$$

здесь $\mu_k = 1 - \beta_k / \alpha_k - 1/\alpha_k$, $\mu_1 = 1 - 1/\alpha_1$ и $\omega_k = \sqrt{\beta_k} / \alpha_k$.

Наибольшее собственное значение матрицы T_k находится подпрограммой EVASB. Обычно это значение нужно лишь для небольшого числа итераций.

Замечание. Процедура PCGRC применима для любых вещественных симметрических положительно определенных матриц: полных, ленточных и разреженных.

Пример. Решается система линейных уравнений. Матрица коэффициентов задана как полная матрица. Аппроксимирующей матрицей является диагональная матрица, диагональ которой взята из матрицы A. Такая аппроксимация называется аппроксимацией Якоби и используется в подпрограмме JCGRC.

```
program GRC1
 use msimsl
 integer(4), parameter :: Lda = 3, n = 3
 integer(4) :: ido, itmax
 real(4) :: a(Lda, n), b(n), p(n), r(n), relerr, x(n), z(n)
 a = reshape((/      1.0,    -3.0,     2.0,          &
                    -3.0,    10.0,    -5.0,          &
                     2.0,    -5.0,     6.0   /),     shape = (/ Lda, n /))
 b = (/ 27.0, -78.0, 64.0 /)                  ! Правая часть системы Ax = b
 r = b; x = b                                 ! x = b – начальное приближение
 itmax  = 100; relerr = 1.0e-5                ! Число итераций и точность
 ido = 0                                      ! Для первого вызова
 do while(ido /= 3)
   call pcgrc(ido, n, x, p, r, z, relerr, itmax)
   select case(ido)
   case(1)
    z = matmul(a, p)                          ! Задаем z = Ap
   case(2)
    do j=1, n                                 ! Используем диагональ матрицы A в
     z(j) = r(j) / a(j, j)                    ! качестве аппроксимирующей матрицы M
    end do                                    ! и устанавливаем z = M^-1 r
   end select
 end do
 call wrrrn('x', 1, n, x, 1, 0)               ! Вывод результата
end program GRC1
```

Результат:

```
                x
  1.000       -4.000        7.000
```

Пример. Используется более сложная аппроксимирующая матрица для решения системы линейных уравнений, возникающей в результате аппроксимации вторых производных в уравнении Лапласа с помощью отношений конечных разностей на сетке 4×4. Матрица системы

$$A=\begin{pmatrix} 4 & -1 & 0 & -1 & & & & & \\ -1 & 4 & -1 & 0 & -1 & & & & \\ 0 & -1 & 4 & -1 & 0 & -1 & & & \\ -1 & 0 & -1 & 4 & -1 & 0 & -1 & & \\ & -1 & 0 & -1 & 4 & -1 & 0 & -1 & \\ & & -1 & 0 & -1 & 4 & -1 & 0 & -1 \\ & & & -1 & 0 & -1 & 4 & -1 & 0 \\ & & & & -1 & 0 & -1 & 4 & -1 \\ & & & & & -1 & 0 & -1 & 4 \end{pmatrix}.$$

В качестве аппроксимирующей матрицы M используем симметрическую трехдиагональную часть A:

$$M=\begin{pmatrix} 4 & -1 & & & & & & & \\ -1 & 4 & -1 & & & & & & \\ & -1 & 4 & -1 & & & & & \\ & & -1 & 4 & -1 & & & & \\ & & & -1 & 4 & -1 & & & \\ & & & & -1 & 4 & -1 & & \\ & & & & & -1 & 4 & -1 & \\ & & & & & & -1 & 4 & -1 \\ & & & & & & & -1 & 4 \end{pmatrix}.$$

В приводимой ниже программе для решения системы $Mz = r$ используется найденное до цикла DO WHILE(ido /= 3) R^TR-разложение симметрической аппроксимирующей матрицы M.

```
program GRC2
 use msimsl
 integer(4), parameter :: n = 9, ncoda = 3, ncom = 1, Lda = 2 * ncoda + 1, Ldm = ncom + 1
 integer(4) :: ido, itmax
 real(4) :: a(Lda, n), p(n), m(Ldm, n), mfac(Ldm, n), r(n), rcond, relerr, x(n), z(n)
 ! Ленточное представление матрицы A
 a = reshape((/      0.0,    0.0,    0.0,    4.0,   -1.0,    0.0,   -1.0,     &
                     0.0,    0.0,   -1.0,    4.0,   -1.0,    0.0,   -1.0,     &
                     0.0,    0.0,   -1.0,    4.0,   -1.0,    0.0,   -1.0,     &
                    -1.0,    0.0,   -1.0,    4.0,   -1.0,    0.0,   -1.0,     &
                    -1.0,    0.0,   -1.0,    4.0,   -1.0,    0.0,   -1.0,     &
                    -1.0,    0.0,   -1.0,    4.0,   -1.0,    0.0,   -1.0,     &
                    -1.0,    0.0,   -1.0,    4.0,   -1.0,    0.0,    0.0,     &
                    -1.0,    0.0,   -1.0,    4.0,   -1.0,    0.0,    0.0,     &
                    -1.0,    0.0,   -1.0,    4.0,    0.0,    0.0,    0.0      &
                  /), shape = (/ Lda, n /))
 ! Ленточное симметрическое представление матрицы M
```

```
 m = reshape((/      0.0,   -1.0,   -1.0,   -1.0,   -1.0,   -1.0,   -1.0,   -1.0,   -1.0,      &
                     4.0,    4.0,    4.0,    4.0,    4.0,    4.0,    4.0,    4.0,    4.0      &
                  /), shape = (/Ldm, n /), order = (/ 2, 1 /))
b = 1.0                                  ! Правая часть системы Ax = b
r = b; x = 0.0                           ! x = 0.0 – начальное приближение
! Находим R^TR-разложение симметрической аппроксимирующей матрицы M
call Lfcqs(n, m, Ldm, ncom, mfac, Ldm, rcond)
itmax  = 100; relerr = 1.0e-4            ! Число итераций и точность
ido = 0                                  ! Для первого вызова
do while(ido /= 3)
  call pcgrc(ido, n, x, p, r, z, relerr, itmax)
  select case(ido)
  case(1)                                ! Задаем z = Ap
    call murbv(n, a, Lda, ncoda, ncoda, n, p, 1, n, z)
  case(2)                                ! Решаем линейную систему Mz = r
    call Lslqs(n, mfac, Ldm, ncom, r, z)
  end select
end do
call wrrrn('x', 1, n, x, 1, 0)           ! Вывод результата
end program GRC2
```

Результат:

```
                                  x
   0.955     1.241     1.349     1.578     1.660     1.578     1.349     1.241      .955
```

6.3.6.2. Метод сопряженных градиентов с аппроксимацией Якоби и обратной связью

Решение системы линейных уравнений $Ax = b$ с симметрической положительно определенной матрицей A методом сопряженных градиентов с аппроксимацией Якоби и обратной связью выполняется подпрограммой JCGRC (DJCGRC). В качестве аппроксимирующей используется диагональная матрица, составленная из диагональных элементов матрицы A. Вызов JCGRC:

CALL JCGRC(*ido*, *n*, *diag*, *x*, *p*, *r*, *z*, *relerr*, *itmax*)

Входные данные: *n*, *diag*, *z*, *relerr*, *itmax*.

Входные/выходные данные: *ido*, *x*, *r*.

Выходные данные: *p*.

ido – флаг, задающий решаемую задачу. При первом вызове *ido* должен равняться нулю. Если подпрограмма завершается с *ido* = 1, то задается $z = Ap$, где A – матрица, и JCGRC вызывается повторно до тех пор, пока не вернет *ido*, равное двум. Завершение с *ido* = 2 означает сходимость итераций, и в этом случае x содержит решение.

n – порядок системы линейных уравнений.

diag – вектор размера *n*, содержащий диагональ матрицы. Все его элементы должны быть строго положительны или строго отрицательны.

x – вектор размера *n*. На входе *x* содержит начальные приближения, на выходе – решение системы.

p – вектор размера *n*. Используется для задания $z = Ap$.

r – вектор размера *n*. При первом вызове содержит правую часть системы. На выходе – невязку.

z – вектор размера *n*; когда *ido* = 1, он содержит *Ap*; когда *ido* = 0, он игнорируется.

relerr – задаваемая на входе относительная ошибка.

itmax – максимально допустимое число итераций.

Автоматически для решения предоставляется память:

- 8*itmax* байт в случае JCGRC;
- 15*itmax* байт в случае DJCGRC.

Описание:

Подпрограмма JCGRC является специальным случаем PCGRC. Она использует диагональ матрицы *A* для приведения системы к виду, удобному для итераций. Более подробно изложение алгоритма дано при описании PCGRC (разд. 6.3.6.1). Для детального ознакомления с алгоритмом можно обратиться к [6] и [48, Chapter 7].

Число итераций зависит от входной матрицы и заданной точности *relerr*. Когда $n >> 1$, можно задать $itmax = n^{0,5}$.

Пример. Решается система линейных уравнений. Матрица коэффициентов задана как полная симметрическая положительно определенная матрица.

```
program GRC3
 use msimsl
 integer(4), parameter :: Lda = 3, n = 3
 integer(4) :: ido, itmax
 real(4) :: a(Lda, n), b(n), diag(n), p(n), r(n), relerr, x(n), z(n)
 a = reshape((/      1.0,     -3.0,      2.0,                    &
                    -3.0,     10.0,     -5.0,                    &
                     2.0,     -5.0,      6.0    /),     shape = (/ Lda, n /))
 b = (/ 27.0, -78.0, 64.0 /)
 r = b; x = b                          ! x = b – начальное приближение
 call scopy(n, a, Lda + 1, diag, 1)    ! Копируем диагональ a в diag
 itmax = 100; relerr = 1.0e-5          ! Число итераций и точность
 ido = 0                               ! Для первого вызова
 do
  call jcgrc(ido, n, diag, x, p, r, z, relerr, itmax)
  if(ido == 2) exit
  z = matmul(a, p)                     ! Задаем z = Ap
```

```
 end do
 call wrrrn('x', 1, n, x, 1, 0)                    ! Вывод результата
end program GRC3
```

Результат:

```
                x
  1.000       -4.000        7.000
```

6.3.6.3. Вычисление LL^T-разложения Холецкого симметрической положительно определенной разреженной матрицы

Выполняется процедурами LSCXD, LNFXD и LFSXD.

Первоначально LSCXD находит структуру данных для представления нижней треугольной матрицы L из LL^T-разложения Холецкого симметрической разреженной матрицы A. Затем LNFXD, используя найденную структуру данных, вычисляет числовое LL^T-разложение Холецкого той же матрицы. Само же разложение и его структура используются LFSXD для решения системы линейных уравнений $Ax = b$.

Описание параметров процедур LSCXD, LNFXD (DLNFXD), LFSXD (DLFSXD) и LSLXD (DLSLXD), работающих с симметрической разреженной матрицей и получающих и/или использующих ее разложение Холецкого, приведено в табл. 6.9.

Таблица 6.9. Параметры процедур LSCXD, LNFXD, LFSXD и LSLXD

Имена	*Смысл*	*Типы*
a	Вектор размера *nz*, содержащий ненулевые элементы нижней треугольной матрицы *A*, решаемой системы $Ax = b$	REAL(4) или REAL(8)
b	Вектор размера *n*, содержащий правую часть решаемой системы линейных уравнений	То же
diag	Вектор размера *n*, содержащий диагональ разложения (матрицы *L*)	Совпадает с типом вектора *a*
ilnz	Вектор размера $n + 1$, содержащий указатели для разложения Холецкого. Внедиагональные ненулевые элементы в столбце *j* разложения хранятся по адресам от *ilnz*(*j*) до *ilnz*(*j* + 1) - 1. Векторы *ilnz*, *nzsub*, *inzsub* задают структуру данных внедиагональных ненулевых элементов в разложении Холецкого. Размещение выполняется по столбцам	INTEGER(4)
invper	Вектор размера *n*, содержащий информацию об обратных перестановках; *invper*(*k*) = *i* означает, что изначально ряд *k* был размещен на месте ряда *i*	"
inzsub	Вектор размера $n + 1$, содержащий указатели для *nzsub*. Номера рядов внедиагональных ненулевых элементов в столбце *j* хранятся в *nzsub* по адресам от *inzsub*(*j*) до *inzsub*(*j* + 1) – 1	"

iper	Вектор размера *n*, содержащий порядок, используемый *ijob*; *iper*(*i*) = *k* означает, ряд исходной матрицы *k* размещен на месте ряда *i*	"
irow и *jcol*	Векторы размера *nz*, содержащие соответственно номера рядов и столбцов ненулевых элементов в нижней треугольной части симметрической матрицы, включая ненулевые диагональные элементы; *irow*(*k*), *jcol*(*k*) – это индексы *k*-го ненулевого элемента, хранимого в координатной форме, причем *irow*(*k*) ≥ *jcol*(*k*)	"
ispace	Память в байтах, необходимая для стека фронтальных матриц	"
itwksp	Размер рабочей области в байтах; для установки размера, заданного по умолчанию, следует принять *itwksp* = 0	"
maxnz	Общее число внедиагональных ненулевых элементов в разложении Холецкого	"
maxsub	Число записей в массиве *nzsub*. На входе *maxsub* – размер массива *nzsub*. Когда используется заданный по умолчанию размер рабочей области (*itwksp* = 0), следует задать *maxsub* = 3*nz*. В противном случае (*itwksp* > 0) устанавливается *maxsub* = (*itwksp* - -10*n* - 7)/4. На выходе *maxsub* дает число записей, используемых для хранения данных *nzsub*	"
n	Порядок матрицы *A* (число уравнений)	"
nz	Общее число ненулевых элементов в нижней треугольной части симметрической матрицы, включая ненулевые диагональные элементы	"
nzsub	Вектор размера *maxsub*, содержащий номера рядов внедиагональных ненулевых элементов в разложении Холецкого	"
rlnz	Вектор размера *maxnz*, содержащий ненулевые элементы нижней треугольной матрицы *L* без ее диагональных элементов	Совпадает с типом вектора *a*
rparam	Вектор, содержащий данные о разложении; *rparam*(1) – наименьший диагональный элемент; *rparam*(2) – наибольший диагональный элемент	То же
x	Вектор размера *n*, в который записывается решение системы линейных уравнений. Если вектор *b* для последующих вычислений не нужен, то *b* и *x* могут быть одним вектором	"

Вызов LSCXD:

CALL LSCXD(*n*, *nz*, *irow*, *jcol*, *ijob*, *itwksp*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *ilnz*, *iper*, *invper*, *ispace*)

Входные данные: *n*, *nz*, *irow*, *jcol*, *ijob*, *itwksp*, *invper*, *ispace*.

Входные/выходные данные: *maxsub*, *iper*.

Выходные данные: *nzsub*, *inzsub*, *maxnz*, *ilnz*, *invper*, *ispace*.

Описание параметров, кроме *ijob*, см. в табл. 6.9.

ijob – целочисленный параметр, задающий способ выполнения перестановок в матрице. Если

- *ijob* = 0, то выбирается заданный пользователем в *iper* порядок; причем он изменяется таким образом, что в числовом разложении может быть использован мультифронтальный метод;
- *ijob* = 1, то выбирается заданный пользователем в *iper* порядок;
- *ijob* = 2, то используется минимальный уровень перестановок;
- *ijob* = 3, то используется минимальный уровень перестановок, пригодный для применения в числовом разложении мультифронтального метода.

Автоматически для решения предоставляется память, равная 10n + + 12nz + 7 байт.

Описание:

Пусть задана система линейных уравнений $Ax = b$, в которой A – симметрическая положительно определенная разреженная матрица. Для хранения ее элементов используются 1 вещественный и 2 целых вектора. Вещественный вектор a содержит все nz ненулевых элементов нижней треугольной части симметрической матрицы A, включая ненулевые диагональные элементы. Векторы *irow* и *jcol* таковы, что $A_{irow(i),jcol(i)} = a(i)$, $i = 1, ..., nz$, причем $irow(i) \geq jcol(i)$ для $i = 1, ..., nz$.

Подпрограмма LSCXD находит структуру данных для представления матрицы L из LL^T-разложения Холецкого матрицы A, используя минимальное или заданное пользователем переупорядочивание. Затем подпрограмма LNFXD вычисляет значения элементов L. Причем $PAP^T = LL^T$, где P – матрица перестановок.

Числовое LL^T-разложение можно выполнять, используя либо стандартный метод разложения для разреженных матриц, либо многофронтальный. Последний работает быстрее, но требует больше памяти.

Многофронтальный метод основан на процедуре, приведенной в [59]. Его подробное изложение дано в [58, 60, 61], [39, 40], [22] и [23]. Стандартный метод полностью изложен в [43].

Пример. Находится структура данных матрицы L, являющейся разложением симметрической разреженной матрицы

$$A = \begin{pmatrix} 10 & 0 & 1 & 0 & 2 \\ 0 & 20 & 0 & 0 & 3 \\ 1 & 0 & 30 & 4 & 0 \\ 0 & 0 & 4 & 40 & 5 \\ 2 & 3 & 0 & 5 & 50 \end{pmatrix} \qquad (6.2)$$

с $nz = 10$ ненулевыми элементами, представляемой векторами

irow	1	2	3	3	4	4	5	5	5	5,
jcol	1	2	1	3	3	4	1	2	4	5,
a	10	20	1	30	4	40	2	3	5	50.

```
program LSF
 use msimsl
 integer(4), parameter :: n = 5, nz = 10
 integer(4) :: ijob, ilnz(n + 1), invper(n), inzsub(n + 1), iper(n), irow(nz), &
              ispace, itwksp, jcol(nz), maxnz, maxsub, nzsub(3 * nz)
 irow = (/ 1, 2, 3, 3, 4, 4, 5, 5, 5, 5 /); jcol = (/ 1, 2, 1, 3, 3, 4, 1, 2, 4, 5 /)
 ! Минимальный уровень упорядочивания, применимый для многофронтального метода
 ijob = 3
 ! Используем заданное по умолчанию рабочее пространство
 itwksp = 0; maxsub = 3 * nz                ! Определяем структуру разложения
 call Lscxd(n, nz, irow, jcol, ijob, itwksp, maxsub, nzsub, &
            inzsub, maxnz, ilnz, iper, invper, ispace)
 call wrirn('Iper', 1, n, iper, 1, 0)       ! Вывод результата
 call wrirn('Invper', 1, n, invper, 1, 0);   call wrirn('Nzsub', 1, maxsub, nzsub, 1, 0)
 call wrirn('Inzsub', 1, n + 1, inzsub, 1, 0)  call wrirn('Ilnz', 1, n + 1, ilnz, 1, 0)
end program LSF
```

Результат:

```
          Iper                                  Invper
 2     1     5     4     3                 2     1     5     4     3

       Nzsub                                     Inzsub
 3     5     4     5                 1     1     3     4     4     4

                 Ilnz
 1     2     4     6     7     7
```

Числовое разложение Холецкого симметрической положительно определенной разреженной матрицы выполняется подпрограммой LNFXD (DLNFXD), использующей ранее найденную LSCXD структуру данных разложения:

CALL LNFXD(*n*, *nz*, *a*, *irow*, *jcol*, *ijob*, *maxsub*, *nzsub*, *inzsub*,
maxnz, *ilnz*, *iper*, *invper*, *ispace*, *itwksp*, *diag*, *rlnz*, *rparam*)

Входные данные: *n*, *nz*, *a*, *irow*, *jcol*, *ijob*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *ilnz*, *iper*, *invper*, *ispace*, *itwksp*.

Выходные данные: *diag*, *rlnz*, *rparam*.

Описание параметров, кроме *ijob*, дано в табл. 6.9. Причем входные для LNFXD параметры *maxsub*, *nzsub*, *inzsub*, *maxnz*, *ilnz*, *iper*, *invper*, *ispace* являются результатом работы LSCXD.

ijob – целочисленный параметр, задающий метод разложения. Если

- *ijob* = 1, то выполняется обычное разложение Холецкого;
- *ijob* = 2, то применяется многофронтальный метод разложения.

Автоматически, если *itwksp* = 0, для решения предоставляется память:

- $3n + 3nz$ байт в случае LNFXD;
- $4n + 6nz$ байт в случае DLNFXD.

Если *itwksp* > 0, то предоставляется рабочая область размера *itwksp*.

Описание:

Пусть задана система линейных уравнений $Ax = b$, в которой A – разреженная положительно определенная симметрическая матрица. Подпрограмма LNFXD, используя результаты LSCXD, находит матрицу L LL^T-разложения Холецкого матрицы PAP^T. Причем $PAP^T = LL^T$, где P – матрица перестановок. Диагональ L хранится в векторе *diag* и строго нижняя ненулевая часть L – в *rlnz*. Причем ненулевые элементы столбца j матрицы L размещены по адресам *rlnz*(i), ..., *rlnz*($i + k$), где i = *ilnz*(j) и k = *ilnz*($j + 1$) - *ilnz*(j) - 1. Номера рядов размещены в векторе *nzsub* по адресам от *inzsub*(j) до *inzsub*($j + 1$) - 1. Числовое LL^T-разложение можно выполнять, используя либо стандартный метод разложения для разреженных матриц, либо многофронтальный. Последний работает быстрее, но требует больше памяти.

Пример. Подпрограмма LNFXD, используя результаты LSCXD, находит числовое LL^T-разложение PAP^T, где A – описанная в примере для LSCXD разреженная симметрическая матрица (6.2).

```
program LNF
 use msimsl
 integer(4), parameter :: n = 5, nz = 10, nrlnz = 10
 integer(4) :: ijob, ilnz(n + 1), invper(n), inzsub(n + 1), iper(n), irow(nz), &
              ispace, itwksp, jcol(nz), maxnz, maxsub, nzsub(3 * nz)
 real(4) :: a(nz), diag(n), rlnz(nrlnz), rparam(2)
 a = (/ 10.0, 20.0, 1.0, 30.0, 4.0, 40.0, 2.0, 3.0, 5.0, 50.0 /)
 irow = (/ 1, 2, 3, 3, 4, 4, 5, 5, 5, 5 /)
 jcol = (/ 1, 2, 1, 3, 3, 4, 1, 2, 4, 5 /)
 ! Минимальный уровень упорядочивания в многофронтальном методе
 ijob = 3
 ! Используем заданное по умолчанию рабочее пространство
 itwksp = 0; maxsub = 3 * nz                ! Определяем структуру разложения
 call Lscxd(n, nz, irow, jcol, ijob, itwksp, maxsub, nzsub, &
            inzsub, maxnz, ilnz, iper, invper, ispace)
 if(nrlnz >= maxnz) then                    ! Проверяем значение nrlnz
  ijob = 2                                  ! Выбираем многофронтальный метод
  ! Числовое разложение разреженной симметрической матрицы
  call Lnfxd(n, nz, a, irow, jcol, ijob, maxsub, nzsub, inzsub, &
             maxnz, ilnz, iper, invper, ispace, itwksp, diag, rlnz, rparam)
  call wrrrn('Diag', 1, n, diag, 1, 0); call wrrrn('Rlnz', 1, maxnz, rlnz, 1, 0)
```

```
 end if
end program LNF
```

Результат:

```
                    Diag
     1        2        3        4        5
 4.472    3.162    7.011    6.284    5.430

                    Rlnz
     1        2        3        4        5        6
0.6708   0.6325   0.3162   0.7132  -0.0285   0.6398
```

6.3.6.4. Решение системы линейных уравнений с симметрической положительно определенной разреженной матрицей с использованием ранее найденного LL^T-разложения Холецкого

Выполняется подпрограммой LFSXD (DLFSXD):

CALL LFSXD(*n*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *rlnz*, *ilnz*, *diag*, *iper*, *b*, *x*)

Входные данные: *n*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *rlnz*, *ilnz*, *diag*, *iper*, *b*.

Выходные данные: *x*.

Описание параметров см. в табл. 6.9. Параметры *n*, *nz*, *maxsub*, *nzsub*, *inzsub*, *maxnz*, *ilnz*, *iper* являются результатом работы LSCXD. Параметры *rlnz* и *diag* являются результатом работы LNFXD.

Описание:

Система $Ax = b$ с симметрической положительно определенной разреженной матрицей решается в 3 этапа. На первом LSCXD возвращает структуру матрицы L из LL^T-разложения Холецкого; на втором LNFXD, используя результаты LSCXD, находит численные значения элементов матрицы L, сохраняя их в векторах *diag* и *rlnz*. На третьем LFSXD, используя результаты LSCXD и LNFXD, определяет решение – вектор x. Порядок получения решения (P – матрица перестановок, представляемая вектором *iper*):

1) решаем $Ly1 = Pb$ относительно $y1$;
2) решаем $L^Ty2 = y1$ относительно $y2$;
3) вычисляем $x = P^Ty2$.

Процедура LFSXD имеет те же источники, что и LSCXD.

Пример. Найдем решение системы $Ax = b$ для двух правых частей $b1 = (23.0, 55.0, 107.0, 197.0, 278.0)^T$ и $b2 = (55.0, 83.0, 103.0, 97.0, 82.0)^T$, полученных в результате умножения $Ax1$ и $Ax2$, где $x1 = (1.0, 2.0, 3.0, 4.0, 5.0)^T$ и $x2 = (5.0, 4.0, 3.0, 2.0, 1.0)^T$ – ожидаемые решения. Здесь A – описанная в примере для LSCXD разреженная симметрическая матрица (6.2).

```
program sxd
 use msimsl
 integer(4), parameter :: n = 5, nz = 10, nrlnz = 10
```

```
integer(4) :: ijob, ilnz(n + 1), invper(n), inzsub(n + 1), iper(n), irow(nz), &
          ispace, itwksp, jcol(nz), maxnz, maxsub, nzsub(3 * nz)
real(4) :: a(nz), b1(n), b2(n), diag(n), rlnz(nrlnz), rparam(2), x(n)
b1 = (/ 23.0, 55.0, 107.0, 197.0, 278.0 /); b2 = (/ 55.0, 83.0, 103.0, 97.0, 82.0 /)
a = (/ 10.0, 20.0, 1.0, 30.0, 4.0, 40.0, 2.0, 3.0, 5.0, 50.0 /)
irow = (/ 1, 2, 3, 3, 4, 4, 5, 5, 5, 5 /); jcol = (/ 1, 2, 1, 3, 3, 4, 1, 2, 4, 5 /)
ijob = 3          ! Минимальный уровень упорядочивания в многофронтальном методе
                  ! Используем заданное по умолчанию рабочее пространство
itwksp = 0; maxsub = 3 * nz              ! Определяем структуру разложения
call Lscxd(n, nz, irow, jcol, ijob, itwksp, maxsub, nzsub, &
        inzsub, maxnz, ilnz, iper, invper, ispace)
if(nrlnz >= maxnz) then                  ! Проверяем значение nrlnz
  ijob = 2                               ! Выбираем многофронтальный метод
  ! Числовое разложение разреженной симметрической матрицы
  call Lnfxd(n, nz, a, irow, jcol, ijob, maxsub, nzsub, inzsub, &
          maxnz, ilnz, iper, invper, ispace, itwksp, diag, rlnz, rparam)
  ! Решаем Ax1 = b1
  call Lfsxd(n, maxsub, nzsub, inzsub, maxnz, rlnz, ilnz, diag, iper, b1, x)
  call wrrrn(' x1 ', 1, n, x, 1, 0)      ! Вывод x1
  ! Решаем Ax2 = b2
  call Lfsxd(n, maxsub, nzsub, inzsub, maxnz, rlnz, ilnz, diag, iper, b2, x)
  call wrrrn(' x2 ', 1, n, x, 1, 0)      ! Вывод x2
end if
end program sxd
```

Результат:

```
                    x1
 1.000     2.000     3.000     4.000     5.000
                    x2
 5.000     4.000     3.000     1.000     1.000
```

6.3.6.5. Решение системы линейных уравнений с симметрической положительно определенной разреженной матрицей методом Холецкого

Находится подпрограммой LSLXD (DLSLXD):

CALL LSLXD(n, nz, a, $irow$, $jcol$, b, $itwksp$, x)

Входные данные: n, nz, a, $irow$, $jcol$, b, $itwksp$.

Выходные данные: x.

Описание параметров дано в табл. 6.9. Параметры a, b, $rlnz$ и $diag$ являются результатом LNFXD.

Автоматически, если $itwksp = 0$, для решения предоставляется память:

- $18n + 21nz + 9$ байт в случае LSLXD;
- $20n + 27nz + 9$ байт в случае DLSLXD.

Если $itwksp > 0$, то предоставляется рабочая область размера $itwksp$.

Память можно выделить явно, вызвав L2LXD (DL2LXD):

CALL L2LXD(*n*, *nz*, *a*, *irow*, *jcol*, *b*, *x*, *iper*, *iparam*, *rparam*, *wk*, *Lwk*, *iwk*, *Liwk*)

iper – вектор размера *n*, содержащий информацию о перестановках.

iparam – целочисленный вектор размера 4 (входной/выходной).

rparam – вещественный вектор размера 2 (выходной).

wk – вещественный вектор размера *Lwk*.

Lwk – размер *wk*; должен быть не менее $2n + 6nz$.

iwk – целочисленный вектор размера *Liwk*.

Liwk – размер *iwk*; должен быть не менее $15n + 15nz + 9$.

Замечание. Параметр *itwksp* подпрограммы LSLXD не является аргументом L2LXD.

Если для L2LXD желательно задать принятые по умолчанию настройки, то установите *iparam*(1) = 1 и вызовите L2LXD. Если необходимы иные настройки, то до обращения к L2LXD выполняется вызов

CALL L4LXD(*iparam*, *rparam*)

который устанавливает в *iparam* и *rparam* заданные для них по умолчанию значения. Затем в них устанавливаются необходимые величины.

Смысл элементов *iparam*:

iparam(1) – флаг инициализации;

iparam(2) – метод числового разложения; если *iparam*(2) = 0, то применяется многофронтальное разложение; если *iparam*(2) = 1, то – стандартный метод разложения для разреженных матриц;

iparam(3) – параметр, задающий способ выполнения перестановок в матрице. Если *iparam*(3) = 0 (значение по умолчанию), то используется минимальный уровень перестановок; *iparam*(3) = 1, то выбирается заданный пользователем в *iper* порядок перестановок;

iparam(4) – является выходным и содержит общее число ненулевых элементов в матрице разложения.

Смысл элементов *rparam*:

rparam(1) – наименьший диагональный элемент в разложении Холецкого;

rparam(2) – наибольший диагональный элемент в разложении Холецкого.

При работе с двойной точностью вызывается DL4LXD и *rparam* имеет тип REAL(8).

Описание:

Пусть задана система линейных уравнений $Ax = b$, в которой A – разреженная положительно определенная симметрическая матрица. Как и ранее, для ее представления используются векторы *a*, *irow* и *jcol* (разд. 6.3.6.3). Подпрограмма LSLXD первоначально использует LSCXD для поиска структуры, получаемой в результате разложения матрицы, затем LSLXD вызыва-

ет LNFXD, которая находит числовое представление разложения, и затем обращается к LFSXD, которая в 3 приема решает исходную систему:

1) $Ly1 = Pb$;
2) $L^T y2 = y1$;
3) $x = P^T y2$.

Подпрограмма LFSXD принимает b, матрицу L и матрицу перестановок P, представляемую массивом *iper*, и возвращает x.

Процедура LSLXD имеет те же источники, что и LSCXD.

Пример. Найдем решение системы $Ax = b$ для $b = (23.0, 55.0, 107.0, 197.0, 278.0)^T$, полученного в результате умножения Ax, где $x = (1.0, 2.0, 3.0, 4.0, 5.0)^T$ – ожидаемое решение. Здесь A – описанная в примере для LSCXD разреженная симметрическая матрица (6.2).

```
program LSL
 use msimsl
 integer(4), parameter :: n = 5, nz = 10
 integer(4) :: irow(nz), jcol(nz), itwksp
 real(4) :: a(nz), b(n), x(n)
 b = (/ 23.0, 55.0, 107.0, 197.0, 278.0 /)
 a = (/ 10.0, 20.0, 1.0, 30.0, 4.0, 40.0, 2.0, 3.0, 5.0, 50.0 /)
 irow = (/ 1, 2, 3, 3, 4, 4, 5, 5, 5, 5 /); jcol = (/ 1, 2, 1, 3, 3, 4, 1, 2, 4, 5 /)
 itwksp = 0                    ! Используем заданное по умолчанию рабочее пространство
 call Lslxd(n, nz, a, irow, jcol, b, itwksp, x)  ! Решаем Ax = b
 call wrrrn(' x ', 1, n, x, 1, 0)    ! Вывод x
end program LSL
```

Результат:

```
                    x
 1.000      2.000      3.000      4.000      5.000
```

6.3.7. Разложение Холецкого симметрической неотрицательно определенной матрицы

Выполняется подпрограммой CHFAC (DCHFAC):

CALL CHFAC(*n*, *a*, *Lda*, *tol*, *irank*, *r*, *Ldr*)

Входные данные: *n*, *a*, *Lda*, *tol*, *Ldr*.

Выходные данные: *irank*, *r*.

n – порядок симметрической $n \times n$-матрицы A.

a – массив, представляющий предназначенную для разложения симметрическую неотрицательно определенную матрицу. Подпрограмма CHFAC обращается только к верхней треугольной части A.

Lda – ведущий размер массива *a*.

tol – допуск, используемый для определения линейной зависимости уравнений системы. Обычно принимают $tol = 100 * \varepsilon_м$, где $\varepsilon_м$ = EPSILON(*tol*).

irank – ранг матрицы A.

r – массив, содержащий верхнюю треугольную $n \times n$-матрицу R, полученную в результате R^TR-разложения Холецкого матрицы A; к нижней треугольной части R программа не обращается. Элементы соответствующих n - *irank* рядов R получают значение 0, если обнаружена линейная зависимость столбцов. Если массив a после вычислений не нужен, то r и a могут быть одним массивом.

Ldr – ведущий размер массива r.

Описание:

Подпрограмма CHFAC находит R^TR-разложение Холецкого симметрической $n \times n$-матрицы A. Матрица R вычисляется как верхняя треугольная. Диагональные элементы R неотрицательны. Вычисления производятся по столбцам. Если A – вырожденная, то n – *irank* рядов в матрице R будут целиком состоять из нулей. Столбец i матрицы A считается линейно зависимым от первых i - 1 столбцов, если

$$\left| a_{ii} - \sum_{j=1}^{i-1} r_{ji}^2 \right| \le \varepsilon \left| a_{ii} \right| ,$$

где ε – значение входного параметра *tol*. Если линейная зависимость обнаружена, то все элементы ряда i получат значение 0.

Подпрограмма CHFAC проверяет, является ли входная матрица неотрицательно определенной, и выдает ошибку с кодом 1, если выполняется любое из условий:

1) $a_{ii} - \sum_{j=1}^{i-1} r_{ji}^2 \le \varepsilon \left| a_{ii} \right|$;

2) $r_{ii} = 0$ и $a_{ik} - \sum_{j=1}^{i-1} r_{ji} r_{jk} \le \varepsilon \sqrt{a_{ii} a_{kk}}, k > i$.

Чтобы A и R могли занимать одну область памяти, в условии 2 вместо a_{ii} используется сумма $\sum_{j=1}^{i-1} r_{ji}^2$.

Алгоритм основан на работе [52]. Из работ [41] и [25] взят метод определения неотрицательной определенности матрицы. Проблема вычисления R^TR-разложение Холецкого симметрической неотрицательно определенной матрицы также обсуждается в [62, p. 85-86].

Пример. Выполняется разложение Холецкого симметрической неотрицательно определенной 5×5-матрицы.

```
program cfa
 use msimsl
 integer(4), parameter :: n = 5, Lda = n, Ldr = n
```

```
integer(4) :: irank
real(4) :: a(Lda, n), r(Ldr, n), tol = 0.00001
a = reshape((/      36.0,    12.0,    30.0,     6.0,    18.0,          &
                    12.0,    20.0,     2.0,    10.0,    22.0,          &
                    30.0,     2.0,    29.0,     1.0,     7.0,          &
                     6.0,    10.0,     1.0,    14.0,    20.0,          &
                    18.0,    22.0,     7.0,    20.0,     40.0 /), shape = (/ Lda, n /))
call chfac(n, a, Lda, tol, irank, r, Ldr)
print '(a, i3)', ' irank = ', irank; call wrrrn('R', n, n, r, Ldr, 0)
end program cfa
```

Результат:

irank = 4

R

	1	2	3	4	5
1	6.000	2.000	5.000	1.000	3.000
2	0.000	4.000	-2.000	2.000	4.000
3	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	3.000	3.000
5	0.000	0.000	0.000	0.000	2.449

Если задать на входе симметрическую неопределенную матрицу, например вместо диагональных элементов 14.0 и 40.0 проставить соответственно 1.4 и 0.4, то в той же программе *cfa* подпрограмма CHFAC выдаст предупреждение

```
*** WARNING ERROR 1 from CHFAC. The leading 4 by 4 submatrix of the input
***                 matrix is not nonnegative definite within the tolerance defined
***                 by TOL = 1.000000E-05.
```

и *результат*, в котором матрица *R* имеет ранг, равный двум, и 3 состоящих из нулей ряда:

irank = 2

R

	1	2	3	4	5
1	6.000	2.000	5.000	1.000	3.000
2	0.000	4.000	-2.000	2.000	4.000
3	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.000	0.000
5	0.000	0.000	0.000	0.000	0.000

6.3.8. Разложение Холецкого симметрической неотрицательно определенной матрицы с применением симметричного выбора

Выполняет подпрограмма LCHRG (DLCHRG):

CALL LCHRG(*n*, *a*, *Lda*, *pivot*, *ipvt*, *fac*, *Ldfac*)

Входные данные: *n*, *a*, *Lda*, *pivot*, *ipvt*, *Ldfac*.

Входные/выходные данные: *ipvt*.

Выходные данные: *fac*.

Тип *a* и *fac* – REAL(4); остальных параметров – INTEGER(4).

n – порядок симметрической матрицы *A*.

a – массив, представляющий симметрическую неотрицательно определенную $n \times n$-матрицу *A*. Подпрограмма LCHRG обращается только к верхней треугольной части *A*.

Lda – ведущий размер массива *a*.

pivot – логический параметр; если *pivot* = .TRUE., то в процессе разложения разрешаются перестановки столбцов; если *pivot* = .FALSE., то перестановки запрещены.

ipvt – целочисленный вектор размера *n*, содержащий на входе информацию, управляющую перестановками строк и столбцов, а на выходе – о выполненных перестановках. На входе, если $ipvt(k) > 0$, столбец *k* матрицы *A* является *начальным*; если $ipvt(k) = 0$, столбец *k* матрицы *A* является *свободным* и он может меняться местами с другими свободными столбцами; если $ipvt(k) < 0$, то столбец *k* матрицы *A* является *завершающим*. На выходе $ipvt(k)$ содержит индекс диагонального элемента *A*, который был перемещен в диагональную позицию *k*. Вектор *ipvt* используется, лишь когда *pivot* = .TRUE..

fac – массив формы (*Ldfac*, *n*), содержащий разложение матрицы *A* – верхнюю треугольную матрицу *R*. Если после вызова LCHRG массив *a* не нужен, то вместо *fac* можно использовать *a*.

Ldfac – ведущий размер массива *fac*.

Описание:

В процессе разложения, когда *pivot* = .TRUE., применяется симметричный выбор диагонального элемента (разд. 5.3.3.3). При этом перед разложением начальные элементы перемещаются в начало массива *a*, а завершающие – в его конец. Во время разложения могут перемещаться элементы, соответствующие свободным столбцам. Результатом разложения являются верхняя треугольная матрица *R* и матрица перестановок *P*, удовлетворяющие равенству $P^TAP = R^TR$. Таким образом, матрица $A = PR^TRP^{-1}$ и система $Ax = b$ эквивалентны системе $R^TR(P^{-1}x) = P^{-1}b$ или системе $R^TRy = P^{-1}b$, где $y = P^{-1}x$.

Прототипом LCHRG является процедура SCHDC пакета LINPACK, приведенная в [34].

Замечание. Матрица *P* представляется вектором *ipvt*, она ортогональна: $P^TP = I$. Для восстановления *P* по *ipvt* применим код

```
integer(4), parameter :: n = 3
integer(4) :: ipvt(n), p(n, n)
```

```
ipvt = (/ 2, 3, 1 /); p = 0
do i = 1, n
  p(ipvt(i), i) = 1
end do
! Проверим равенство P^TP = I; WRIRN напечатает единичную матрицу
call wrirn('I', n, n, matmul(transpose(p), p), n, 0)
```

В случае неотрицательно определенной матрицы A для решения системы $Ax = b$ используется последовательность операторов

```
pivot = .true.                          ! Разрешаем перестановки
ipvt = 0                                ! Используем для перестановки все столбцы
call Lchrg(n, a, Lda, pivot, ipvt, fac, Ldfac) ! Разложение по Холецкому
! Переставляем элементы в b в соответствии с порядком,
! заданным вектором ipvt; результат храним в x
call permu(n, b, ipvt, 1, x)            ! x = P^-1b
call Lfsds(n, fac, Ldfac, x, y)         ! Находим y как решение R^TRy = P^-1b
call permu(n, y, ipvt, 2, x)            ! Находим x = Py
```

В этой последовательности LCHRG находит разложение матрицы A по Холецкому ($A = PR^TRP^{-1}$) и размещает его в *fac*, записывая информацию о перестановках столбцов в *ipvt*. Далее подпрограмма PERMU записывает элементы b в x в порядке, заданном вектором *ipvt*, выполняя преобразование $x = P^{-1}b$. Затем LFSDS, используя ранее найденное разложение матрицы A, решает систему $R^TRy = x$ относительно y, а PERMU выполняет преобразование $x = Py$, находя в итоге решение системы $Ax = b$.

Пример. Подпрограмма LCHRG используется совместно с PERMU и LFSDS для решения системы линейных уравнений $Ax = b$ с неотрицательно определенной матрицей A.

```
program chol3
 use msimsl, nouse => fac
 integer(4), parameter :: n = 5, Lda = n, Ldfac = n
 integer(4) :: ipvt(n)
 real(4) :: a(Lda, n), fac(Ldfac, n), b(n), x(n)
 logical(4) :: pivot = .true.                  ! Разрешаем перестановки
 a = reshape((/      36.0,    12.0,    30.0,     6.0,    18.0,                &
                     12.0,    20.0,     2.0,    10.0,    22.0,                &
                     30.0,     2.0,    29.0,     1.0,     7.0,                &
                      6.0,    10.0,     1.0,    14.0,    20.0,                &
                     18.0,    22.0,     7.0,    20.0,     40.0 /), shape = (/ Lda, n /))
 b = (/ 0.002, 0.003, 0.004, 50.0, 20.0 /)
 ipvt = 0              ! Используем для перестановок все диагональные элементы
 call Lchrg(n,a,Lda,pivot,ipvt,fac,Ldfac)      ! Разложение Холецкого
 print '(1x, a, 10i3)', 'ipvt =', ipvt         ! Вектор перестановок
 call permu(n, b, ipvt, 1, x)                  ! Переставляем элементы в b, результат храним в x
```

```
call wrrrn('R', n, n, fac, Lda, 1)          ! Верхняя треугольная матрица R^TR-разложения
call Lfsds(n, fac, Ldfac, x, x)             ! Находим x
call permu(n, x, ipvt, 2, x)                ! Восстанавливаем порядок элементов в x
call wrrrn('x', 1, n, x, 1, 0)              ! Вывод результата и невязок
print '(1x, a, 10f8.5)', 'Residuals: ', b - matmul(a, x)
end program chol3
```

Результат:

```
ipvt =  5  1  2  4  3
                          R
              1        2        3        4        5
   1       6.325    2.846    3.479    3.162    1.107
   2       0.000    5.282    0.398   -0.568    5.083
   3       0.000    0.000    2.782   -0.278   -1.391
   4       0.000    0.000    0.000    1.897    0.000
   5       0.000    0.000    0.000    0.000    0.001

                          x
  -1639.5      819.9    1640.3     10.6      -5.0
Residuals: -0.00377  0.00034 -0.00336  0.00027  0.00173
```

Рассматривая полученный вектор невязок и сравнивая его с вектором b, можно сделать вывод, что полученному решению доверять нельзя – величина первых трех невязок соизмерима с соответствующими элементами вектора b. Применение двойной точности не даст улучшений:

```
                                   x
  -2.627D+12     1.314D+12     2.627D+12     1.056D+01     -5.001D+00
Residuals:  -0.02194 -0.00356 -0.01568  0.00409 -0.00003
```

Это неудивительно, поскольку, как это обнаруживает CHFAC, входная матрица является неполноранговой, а оценка ее числа обусловленности, возвращаемая DLFCRG, равна 6.7765E+016. Таким образом, полезно перед применением LCHRG оценить ранг матрицы A и применять разложение, например в LFSDS, лишь в случае полноты ранга.

Если в матрицу A внести те же, что и в предшествующем примере, изменения, т. е. вместо значений диагональных элементов 14.0 и 40.0 проставить соответственно 1.4 и 0.4, то вызов LCHRG приведет к ошибке, сопровождаемой сообщением

```
*** TERMINAL ERROR 3 from LSLRT. The input triangular matrix is singular.
***      The index of the first zero diagonal element is equal to 3.
         Here is a traceback of subprogram calls in reverse order:
         Routine name          Error type  Error code
         ------------          ----------  ----------
```

```
LSLRT          5    3  (Called internally)
LFSDS          0    0
USER           0    0
Stop – Program terminated.
```

Та же ошибка возникнет и при запрете перестановок:

```
pivot = .false.                    ! Запрещаем перестановки
ipvt = (/ (i, i = 1, n) /)         ! Естественный порядок столбцов
```

При этом, разумеется, изменится и матрица R:

R

	1	2	3	4	5
1	6.00	2.00	5.00	1.00	3.00
2	0.00	4.00	-2.00	2.00	4.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	9.00	9.00
5	0.00	0.00	0.00	0.00	15.00

6.3.9. Процедуры с вещественной симметрической неопределенной матрицей

6.3.9.1. Об использовании процедур с симметрической неопределенной матрицей

Поскольку процедуры с симметрической неопределенной матрицей имеют ранее рассмотренные аналоги, то они будут проиллюстрированы лишь примерами. Существенных различий в применении подпрограмм из табл. 6.1 и их аналогов нет. Более того, в подпрограмме для неопределенной симметрической матрицы автоматически выделяется тот же, что в и аналоге, объем памяти и генерируются все присущие ее аналогу ошибки. Впрочем, есть отличия:

- процедуры для симметрической неопределенной матрицы используют ее UDU^T-разложение, а процедуры-аналоги – LU-разложение матрицы общего вида;
- процедуры с симметрической матрицей не имеют параметра *ipath*.

В рассматриваемых процедурах симметрическая неопределенная матрица представляется, например, подпрограммой LFCSF в виде UDU^T-разложения, где U – верхняя унитреугольная матрица, а D – диагональная матрица, состоящая из блоков размера 1×1 или 2×2 (разд. 5.3.4). Затем, используя это разложение, можно найти обратную матрицу или, применив, например, LFSSF, решить систему линейных уравнений $Ax = b$. Процедуры применимы и для положительно и для неотрицательно определенных матриц, но требуют больших временных ресурсов, чем процедуры с такими матрицами, выполняющими их R^TR-разложение.

Смысл используемых с подпрограммами параметров поясняется в разделах, содержащих описания их аналогов, и в табл. 6.4.

6.3.9.2. Примеры для процедур с симметрической неопределенной матрицей

Пример для LFCSF, вычисляющей UDU^T-разложение симметрической неопределенной матрицы и выполняющей оценку ее числа обусловленности. Вычислить A^{-1} – матрицу, обратную симметрической $n \times n$-матрице A.

Порядок вычислений:

1) найти, используя LFCSF, UDU^T-разложение матрицы A;
2) вызвать n раз LFISF, находя в каждом вызове очередной столбец A^{-1}.

```
program useUDUT
 use msimsl, nouse => fac
 integer(4), parameter :: n = 3, Lda = n, Ldfac = n
 integer(4) :: ipvt(n), j
 real(4) :: a(Lda, Lda), ainv(Lda, Lda), fac(Ldfac, Ldfac), rcond, res(n), rj(n)
 a = reshape((/        1.0,    -2.0,     1.0,          &
                      -2.0,     3.0,    -2.0,          &
                       1.0,    -2.0, 3.0 /), shape = (/ Lda, Lda /))
 ! Находим UDUT-разложение и оценку числа обусловленности матрицы A
 call Lfcsf(n, a, Lda, fac, Ldfac, ipvt, rcond)
 print *, 'cond1(A) = ', 1.0 / rcond            ! Вывод оценки числа обусловленности матрицы A
 rj = 0.0
 do j=1, n
  rj(j) = 1.0          ! rj(j) – столбец с номером j единичной матрицы, поэтому LFISF
                       ! разместит столбец j обратной матрицы A-1 в j-й столбец массива ainv
  call Lfisf(n, a, Lda, fac, Ldfac, ipvt, rj, ainv(1, j), res)
  rj(j) = 0.0
 end do
 call wrrrn('ainv', n, n, ainv, Lda, 0)          ! Вывод результата
end program useUDUT
```

Результат:

```
cond1(A) =    29.750000
                    ainv
              1          2          3
   1       -2.500     -2.000     -0.500
   2       -2.000     -1.000      0.000
   3       -0.500      0.000      0.500
```

Пример для LFTSF и LFSSF. Решить систему линейных уравнений с m правыми частями.

Порядок решения:

1) найти, используя LFTSF, UDU^T-разложение симметрической матрицы A;

2) вызвать m раз LFSSF, находя в каждом вызове решение системы для очередной правой части.

```
program useUDUT2
 use msimsl, nouse => fac
 integer(4), parameter :: n = 3, Lda = n, Ldfac = n, m = 2
 integer(4) :: ipvt(n), j
 real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), b(n, m), x(n, m)
 a = reshape((/        1.0,    -2.0,     1.0,            &
                      -2.0,     3.0,    -2.0,            &
                       1.0,    -2.0,     3.0 /), shape = (/ Lda, Lda /))
 b = reshape((/        1.0,     4.0,    -1.0,            &
                      10.0,    14.0,     9.0 /), shape = (/ n, m /))
 call Lftsf(n, a, Lda, fac, Ldfac, ipvt)        ! Находим UDU^T-разложение матрицы A
 do j=1, m                                      ! Решаем систему с несколькими правыми частями
  call Lfssf(n, fac, Ldfac, ipvt, b(1, j), x(1, j))
 end do
 call wrrrn('x', n, m, x, n, 0)                 ! Вывод результата
end program useUDUT2
```

Результат (каждый столбец результата содержит решение):

```
          x
          1          2
  1   -10.000    -57.500
  2    -6.000    -34.000
  3    -1.000      -.500
```

Пример для LFISF, решающей с применением ранее найденного UDU^T-разложения симметрической матрицы A систему линейных уравнений $Ax = b$, выполняющей затем итерационное уточнение решения и вычисляющей невязку.

В примере несколько раз решается система уравнений с одинаковой левой и разными правыми частями. Каждая новая правая часть получается в результате увеличения второго элемента вектора b на 0.2. Сразу после решения выводится вычисленная в LFISF невязка *res*.

```
program useUDUT3
 use msimsl, nouse => fac
 integer(4), parameter :: n = 3, Lda = n, Ldfac = n
 integer(4) :: ipvt(n), j
 real(4) :: a(Lda, Lda), b(n), x(n, 3), fac(Lda, Lda), res(n, 3), rcond
 a = reshape((/        1.0,    -2.0,     1.0,            &
                      -2.0,     3.0,    -2.0,            &
                       1.0,    -2.0,     3.0    /), shape = (/ Lda, Lda /))
 b = (/ 4.1, -4.7, 6.5 /)
 ! Находим UDU^T-разложение и оценку числа обусловленности матрицы A
 call Lfcsf(n, a, Lda, fac, Ldfac, ipvt, rcond)
```

```
print *, 'cond1(A) = ', 1.0 / rcond          ! Вывод оценки числа обусловленности матрицы A
do j =1, 3                                   ! Решаем систему с разными правыми частями
  call Lfisf(n, a, Lda, fac, Lda, ipvt, b, x(1, j), res(1, j))
  b(2) = b(2) + 0.2
end do                                       ! Вывод результата и невязки
 call wrrrn('x', n, 3, x, n, 0); call wrrrn('res', n, 3, res, n, 0)
end program useUDUT3
```

Результат (каждый столбец первой таблицы содержит решение, а второй – соответствующие ему невязки):

```
cond1(A) =    29.750000

                x
            1        2        3
  1    -4.100   -4.500   -4.900
  2    -3.500   -3.700   -3.900
  3     1.200    1.200    1.200

                        res
                1                2
  1     -2.384e-07       -2.384e-07       -2.384e-07
  2     -2.384e-07       -2.384e-07       -2.384e-07
  3      0.000e+00        0.000e+00        0.000e+00
```

Пример для LSLSF и LSASF, вычисляющих UDU^T-разложение симметрической матрицы A и решающих систему линейных уравнений $Ax = b$. Дополнительно LSASF выполняет итерационное уточнение решения.

```
program useLsl
 use msimsl
 integer, parameter :: Lda = 3, n = 3
 real :: a(Lda, Lda), b(n), x(n)
 a = reshape((/      1.0,    10.0,    20.0,              &
                    10.0,     1.0,    30.0,              &
                    20.0,    30.0,     1.0    /), shape = (/ Lda, Lda /))
 b = (/ 4.1, -4.7, 6.5 /)
 call Lslsf(n, a, Lda, b, x)                 ! Решаем линейную систему Ax = b
 ! Тот же результат для заданных данных даст и вызов LSASF
 ! call Lsasf(n, a, Lda, b, x)
 call wrrrn('x', 1, n, x, 1, 0)              ! Вывод результата
end program useLsl
```

Результат:

```
                x
   -0.4376      0.5093      -0.0278
```

6.3.10. Разложение симметрической матрицы с добавлением к ней диагональной матрицы

Используя метод Холецкого, матрицу $A + D$ (A – симметрическая в общем случае неопределенная матрица; D – диагональная матрица, определяемая в процессе разложения таким образом, что $A + D$ является положительно определенной) можно представить в виде R^TR-разложения. Это выполняется подпрограммой MCHOL (DMCHOL):

CALL MCHOL(*n*, *a*, *Lda*, *tol*, *irank*, *r*, *Ldr*, *dmax*, *ind*)

Входные данные: *n*, *a*, *Lda*, *tol*, *Ldr*.

Выходные данные: *irank*, *r*, *dmax*, *ind*.

n – порядок симметрической $n \times n$-матрицы A.

a – массив, представляющий симметрическую матрицу, предназначенную для разложения по методу Холецкого. Подпрограмма MCHOL обращается только к верхней треугольной части A.

Lda – ведущий размер массива *a*.

tol – допуск, используемый для определения линейной зависимости уравнений системы. Обычно принимают для MCHOL *tol* = 100 * AMACH(4), а для DMCHOL *tol* = 100 * DMACH(4) (AMACH(4) и DMACH(4) возвращают машинную точность соответственно для разновидностей типов KIND = 4 и KIND = 8).

irank – ранг матрицы $A + D$.

r – массив, содержащий верхнюю треугольную $n \times n$-матрицу R, полученную в результате R^TR-разложения по методу Холецкого матрицы $A + D$; к нижней треугольной части R программа не обращается. Если массив *a* после вычислений не нужен, то *r* и *a* могут быть одним массивом.

Ldr – ведущий размер массива *r*.

dmax – наибольший диагональный элемент матрицы D. Если *dmax* = 0.0, то A – неотрицательно определенная матрица и R является ее разложением по методу Холецкого. Если *dmax* > 0.0, то A является неопределенной, т. е. имеет по крайней мере одно отрицательное собственное значение. В этом случае *dmax* является верхней по абсолютной величине границей для минимального собственного значения.

ind – индекс для последующих вычислений в направлении спуска при обнаружении седловой точки s. Если *ind* = 0, то A – неотрицательно определенная матрица. Если *ind* > 0, то решение $Rs = e$ относительно s, где e – единичный вектор с элементом $e_{ind} = 1$ и с остальными элементами, равными нулю, является направлением спуска, т. е. произведение s^TAs отрицательно.

Описание:

Подпрограмма MCHOL находит R^TR-разложения Холецкого матрицы $A + D$, где A – симметрическая матрица, а D – диагональная матрица с отно-

сительно большими элементами, такими, что $A + D$ является положительно определенной матрицей.

Алгоритм MCHOL подобен приведенному в [44, p. 108-111]. Однако в MCHOL матрица $A + D$ может быть и вырожденной.

Вычисления производятся по рядам. Если $A + D$ – вырожденная матрица, то $n - irank$ рядов в матрице R будут целиком состоять из нулей. Ряд i матрицы $A + D$ оказывается линейно зависимым от первых i - 1 рядов, если выполняются два следующих условия:

1) $\left| a_{ii} - \sum_{j=1}^{i-1} r_{ji}^2 \right| \le \varepsilon \left| a_{ii} \right|$,

2) $\left| a_{ik} - \sum_{j=1}^{i-1} r_{ji} r_{jk} \right| \le \varepsilon \sqrt{\left| a_{ii} a_{kk} \right|}, k > i$,

где ε – значение входного параметра *tol*.

Подпрограмма MCHOL часто используется совместно с подпрограммой GIRTS для определения направления спуска в задаче поиска минимума. Пусть A и g – соответственно текущие Гессиан и градиент в некоторой задаче минимизации. Решение линейной системы $As = -g$ относительно s может не дать направление спуска, если A не является положительно определенной. Оно будет гарантированно определено, если искать s как решение $(A + D)s = -g$, где $A + D$ – положительно определенная матрица. Направление спуска s определяется GIRTS, которая получает верхнюю треугольную матрицу R, вычисленную MCHOL в результате разложения $A + D$ по методу Холецкого. Подпрограмма GIRTS определяет s, решая последовательно две треугольные системы линейных уравнений $R^T x = -g$ и $Rs = x$, первую – относительно x, а вторую – относительно s. Также, если $g = 0$ и A не является положительно определенной, т. е. текущее решение является седловой точкой, GIRTS можно использовать для нахождения s из системы $Rs = e$, где e – единичный вектор, в котором $e_i = 1$, если $i = ind$, и $e_i = 0$ – в противном случае.

Пример 1. Выполняется $R^T R$-разложение Холецкого симметрической неотрицательно определенной 5×5-матрицы.

```
program chol
 use msimsl
 integer(4), parameter :: n = 5, Lda = n, Ldr = n
 integer(4) :: ind, irank
 real(4) :: a(Lda, n), dmax, r(Ldr, n), tol = 0.00001
 a = reshape((/      36.0,   12.0,   30.0,    6.0,   18.0,          &
                     12.0,   20.0,    2.0,   10.0,   22.0,          &
                     30.0,    2.0,   29.0,    1.0,    7.0,          &
```

```
                    6.0,   10.0,    1.0,   14.0,   20.0,                &
                   18.0,   22.0,    7.0,   20.0,    40.0 /), shape = (/ Lda, n /))
 call mchol(n, a, Lda, tol, irank, r, Ldr, dmax, ind)
 print '(a, i3 / a, f10.3 / a, i3)', ' irank = ', irank, ' dmax =  ', dmax, ' ind =   ', ind
 call wrrrn('R', n, n, r, Ldr, 0)
end program chol
```

Результат:

irank = 4

dmax = 0.000E+00

ind = 0

R

	1	2	3	4	5
1	6.000	2.000	5.000	1.000	3.000
2	0.000	4.000	-2.000	2.000	4.000
3	0.000	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	3.000	3.000
5	0.000	0.000	0.000	0.000	2.449

Из примера следует, что неотрицательно определенные матрицы при выполнении их R^TR-разложения подпрограммой MCHOL не модифицируются.

Пример 2. Выполняется R^TR-разложение Холецкого симметрической неопределенной 3×3-матрицы. Также подпрограммой GIRTS находится решение $Rs = e$. Произведение s^TAs отрицательно; этот факт проверяется функцией BLINF, возвращающей значение квадратичной формы s^TAs.

```
program chol2
 use msimsl
 integer(4), parameter :: n = 3, Lda = n, Ldr = n
 integer(4) :: ind, irank
 real(4) :: a(Lda, n), dmax, e(n), r(Ldr, n), s(n), tol = 0.00001
 a = reshape((/        1.0,    1.0,    2.0,                  &
                       1.0,    1.0,    3.0,                  &
                       2.0,    3.0,     1.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call mchol(n, a, Lda, tol, irank, r, Ldr, dmax, ind)
 print '(a, i3 / a, f10.3 / a, i3)', ' irank = ', irank, ' dmax =  ', dmax, ' ind =   ', ind
 call wrrrn('R', n, n, r, Ldr, 0)
 if (ind > 0) then
   e = 0.0;  e(ind) = 1.0
   call girts(n, r, Ldr, 1, e, n, 1, irank, s, n, r, Ldr)
   call wrrrn('s', 1, n, s, n, 0)
   print '(a, f10.3)', ' trans(s)*A*s = ', blinf(n, n, a, Lda, s, s)
 end if
end program chol2
```

Результат:

```
irank =  3
dmax  =    5.016
ind   =  3
                    R
              1          2          3
   1       1.942      0.515      1.030
   2       0.000      2.398      1.030
   3       0.000      0.000      1.059

                    s
  -0.393       1.061          0.000
trans(s)*A*s =    -2.254
```

6.3.11. Комментарий

Более продуктивно функционируют процедуры для положительно определенной симметрической матрицы. Если же при вызове такой процедуры устанавливается, что матрица не является положительно определенной, то возникнет информационная ошибка уровня 4 с кодом 2, которую можно зафиксировать функциями IERCD и N1RTY (разд. 3.4.4) и вызвать подходящую процедуру для неотрицательно определенной или неопределенной симметрической матрицы. (Действия, связанные с выводом сообщений и остановом программы, регулируются рассмотренной в разд. 3.4.3 подпрограммой ERSET.) Аналогичным образом можно приспособить программу для обработки иных серьезных ошибок, с тем чтобы, избежав ее останова, продолжить вычисления и получить требуемый результат.

6.4. СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ С ВЕЩЕСТВЕННОЙ МАТРИЦЕЙ ТЕПЛИЦА

IMSL содержит процедуру для решения линейных систем с одним видом персимметрических матриц – с теплицевыми матрицами. На каждой диагонали матрицы Теплица расположены равные элементы:

$$A=\begin{pmatrix} p_0 & p_1 & p_2 & p_3 \\ p_{-1} & p_0 & p_1 & p_2 \\ p_{-2} & p_{-1} & p_0 & p_1 \\ p_{-3} & p_{-2} & p_{-1} & p_0 \end{pmatrix}.$$

Для решения систем с матрицей Теплица следует вызвать подпрограмму LSLTO (DLSLTO):

CALL LSLTO(*n, a, b, ipath, x*)

Входные данные: n, a, b, *ipath*.

Выходные данные: x.

Описание параметров, кроме a, такое же, как и в табл. 6.4.

a – вектор типа REAL(4) размера $2n - 1$, содержащий первый ряд коэффициентов матрицы A, после которых располагается первый столбец этой матрицы, в котором отсутствует первый его элемент. Например, для системы

$$\begin{pmatrix} 2 & -3 & -1 & 6 \\ 1 & 2 & -3 & -1 \\ 4 & 1 & 2 & -3 \\ 3 & 4 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 16 \\ -29 \\ -7 \\ 5 \end{pmatrix}$$

вектор a определяется так:

```
real(4) :: a(9) = (/ 2, -3, -1, 6, 1, 4, 3 /)        ! или real(8)
```

Автоматически для решения предоставляется память:

- $2n - 2$ байт в случае LSLTO;
- $4n - 4$ байт в случае DLSLTO.

Алгоритм описан в [70] и в [6].

Пример. Решить приведенную в начале раздела систему уравнений.

```
program Toeplitz
 use msimsl
 integer(4), parameter :: n = 4
 real(4) :: a(2 * n - 1), b(n), x(n)
 a = (/ 2.0, -3.0, -1.0, 6.0, 1.0, 4.0, 3.0 /)
 b = (/ 16.0, -29.0, -7.0, 5.0 /)
 call Lslto(n, a, b, 1, x)                    ! Решаем систему линейных уравнений Ax = b
 call wrrrn('x', 1, n, x, 1, 0)               ! Вывод результата
end program Toeplitz
```

Результат:

```
                    x
   -2.000      -1.000       7.000       4.000
```

6.5. СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ С КОМПЛЕКСНОЙ МАТРИЦЕЙ

Практически все процедуры IMSL для комплексных матриц имеют вещественные аналоги. Например, система линейных уравнений $Ax = b$ с комплексной матрицей Теплица решается процедурой

CALL LSLTC(n, a, b, *ipath*, x)

а ее вещественным аналогом является рассмотренная в разд. 6.4 процедура

CALL LSLTO(n, a, b, *ipath*, x)

Причем и LSLTO и LSLTC содержат один и тот же набор входных параметров, имеющих одинаковый смысл. Различие – в типе, используемом для параметров *a*, *b* и *x*. В случае LSLTO их тип REAL(4), а в случае LSLTC – COMPLEX(4).

Подобные аналогии справедливы и для иных, кроме LSLCC, процедур IMSL с комплексной матрицей. Поэтому ниже подробно разбирается только не имеющая аналогов процедура LSLCC; для остальных объем приводимых сведений ограничивается описаниями назначения процедур, указанием их вещественных аналогов и синтаксиса вызова. Приводится также ряд примеров работы с рассматриваемыми процедурами. Чтобы вспомнить смысл переменных и алгоритмы процедур, достаточно обратиться к описанию их вещественных аналогов. Как и ранее, подпрограммы, имеющие дело с двойной точностью, с типом COMPLEX(8), начинаются с префикса D. Подпрограммы, не имеющие такого префикса, оперируют типом COMPLEX(4).

6.5.1. Процедуры с комплексной неэрмитовой матрицей

6.5.1.1. Список процедур

Описание подпрограмм, использующих комплексную неэрмитову матрицу, дадим в табл. 6.10, предварительно заметив, что, обращаясь к описанию аналога (как это рекомендуется в таблице), следует при прочтении в табл. 6.1 графы "Назначение" изменять имена, на которые в этой графе есть ссылки. Порядок изменения имен следует из табл. 6.10. Например, имя LFCRG меняется на LFCCG. То есть новое имя получается в результате замены в старом буквы R на букву C. Правда, в некоторых именах выполняются иные замены, например имена подпрограмм для разреженных матриц образуются в результате замены X на Z. Но в любом случае изменяется одна буква. Как и ранее, не приводятся подпрограммы 2-го уровня.

Таблица 6.10. Подпрограммы с комплексной неэрмитовой матрицей

Подпрограмма	*Назначение*	*Аналог*
	Циклическая матрица и матрица Теплица	
LSLCC	Решает систему линейных уравнений с циклической матрицей	-
LSLTC	Решает систему линейных уравнений с матрицей Теплица	LSLTO
	Комплексная неэрмитова матрица общего вида	
LFCCG	См. описание аналога в табл. 6.1	LFCRG
LFTCG	То же	LFTRG
LFSCG	"	LFSRG

LFICG	См. описание аналога в табл. 6.1	LFIRG
LSLCG	То же	LSLRG
LSACG	"	LSARG
LINCG	"	LINRG
Комплексная неэрмитова ленточная матрица		
LFCCB	См. описание аналога в табл. 6.1	LFCRB
LFTCB	То же	LFTRB
LFSCB)	"	LFSRB
LFICB	"	LFIRB
LSLCB	"	LSLRB
LSACB	"	LSARB
Комплексная неэрмитова трехдиагональная матрица		
LSLCQ	См. описание аналога в табл. 6.1	LSLCR
LSLTQ	То же	LSLTR
Комплексная неэрмитова треугольная матрица		
LFCCT	См. описание аналога в табл. 6.1	LFCRT
LINCT	То же	LINRT
LSLCT	"	LSLRT
Комплексная неэрмитова разреженная матрица		
LFTZG	См. описание аналога в табл. 6.1	LFTXG
LFSZG	То же	LFSXG
LSLZG	"	LSLXG

Вызовы подпрограмм из табл. 6.10 аналогичны вызовам их аналогов. Для формирования соответствующего вызова следует вместо имени аналога в соответствующей строке табл. 6.2 записать из табл. 6.10 имя подпрограммы.

Подпрограммы из табл. 6.10 имеют те же, что и их аналоги, ошибки. Сведения о них приведены в табл. 6.3.

6.5.1.2. Системы линейных уравнений с циклической матрицей

Опишем не имеющие вещественных аналогов подпрограммы LSLCC и DLSLCC) решающие системы линейных уравнений с *циклической матрицей*, имеющей вид

$$A=\begin{pmatrix} p_1 & p_2 & p_3 & p_4 \\ p_4 & p_1 & p_2 & p_3 \\ p_3 & p_4 & p_1 & p_2 \\ p_2 & p_3 & p_4 & p_1 \end{pmatrix}.$$

Вызов LSLCC:

CALL LSLCC(n, a, b, $ipath$, x)

Входные данные: n, a, b, $ipath$.

Выходные данные: x.

n – порядок матрицы A.

a – комплексный вектор размера n, содержащий первый ряд коэффициентов матрицы A. Поскольку матрица является циклической, то для ее задания достаточно первого ряда.

b – комплексный вектор размера n, содержащий правую часть системы линейных уравнений $Ax = b$.

$ipath$ – целочисленный флаг. Если $ipath = 1$, то решается система $Ax = b$; если $ipath = 2$, то решается система $A^T x = b$.

x – комплексный вектор размера n, содержащий решение системы линейных уравнений.

Автоматически для решения предоставляется память:

- $8n + 15$ байт в случае LSLCC;
- $16n + 30$ байт в случае DLSLCC.

Описание:

Пусть дана система $Ax = b$, в которой A – циклическая матрица. Для ее решения находится вектор q, такой, что $(Ax)_j = \sum_{i=1}^{n} q_{i-j+1} x_i = (q \cdot x)$, где $(q \cdot x)$ - свертка векторов q и x.

В соответствии с теоремой о свертке, если $(q \cdot x) = b$, то $\hat{q} * \hat{x} = \hat{b}$, где $\hat{q}, \hat{x}$ и $\hat{b}$ – дискретные преобразования Фурье соответственно векторов q, x и b, которые вычисляются процедурой FFTCF библиотеки IMSL. Тогда, получив $\hat{x} = \hat{b} / \hat{q}$, восстановим, применив FFTCB, по $\hat{x}$ вектор x.

Пример. Решается система линейных уравнений $Ax = b$.

```
program circ
 use msimsl
 integer(4), parameter :: n = 4, ipath = 1
 complex(4) :: a(n), b(n), x(n)
 ! Первый ряд матрицы A
 a = (/ (2.0, 2.0), (-3.0, 0.0), (1.0, 4.0), (6.0, -2.0) /)
 b = (/ (6.0, 65.0), (-41.0, -10.0), (-8.0, -30.0), (63.0, -3.0) /)
 call Lslcc(n, a, b, ipath, x)          ! Решаем линейную систему Ax = b
 call wrcrn('x', 1, n, x, 1, 0)         ! Вывод результата
end program circ
```

Результат:

```
                              x
 (-2.000, 0.000)  (-1.000, -5.000)   ( 7.000, 2.000)   ( 0.000, 4.000)
```

6.5.1.3. Примеры процедур с комплексной неэрмитовой матрицей

Приведем по одному примеру для каждого представленного в табл. 6.9 вида матриц.

Пример для LSLTC, работающей с матрицей Теплица. Решается система линейных уравнений.

```
program Ltc
 use msimsl
 integer(4), parameter :: n = 4, ipath = 1
 complex(4) :: a(2 * n – 1), b(n), x(n)
 ! Элементы матрицы A
 !      2+2i      -3      1+4i      6-2i
 !        i      2+2i      -3      1+4i
 !      4+2i       i      2+2i      -3
 !      3-4i     4+2i       i      2+2i
 ! Первый ряд и первый столбец (без первого элемента) матрицы A
 a = (/ (2.0,2.0), (-3.0,0.0), (1.0,4.0), (6.0,-2.0), (0.0,1.0), (4.0,2.0), (3.0,-4.0) /)
 b = (/ (6.0,65.0), (-29.0,-16.0), (7.0,1.0), (-10.0,1.0) /)
 call Lsltc(n, a, b, ipath, x)                ! Решаем линейную систему Ax = b
 call wrcrn('x', 1, n, x, 1, 0)               ! Вывод результата
end program Ltc
```

Результат:

```
                                  x
 (-2.000, 0.000)  (-1.000, -5.000)   ( 7.000, 2.000)   ( 0.000, 4.000)
```

Пример для LFICG, работающей с комплексной матрицей общего вида и решающей систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение матрицы A, и выполняющей итерационное уточнение решения. Решается система с несколькими правыми частями. Каждая новая правая часть получается путем прибавления после получения решения величины $0.5 + 0.5i$ ко второму элементу вектора b.

```
program Ltc
 use msimsl, nouse => fac
 integer(4), parameter :: Lda = 3, Ldfac = 3, n = Lda, ipath = 1
 integer(4) :: ipvt(n), j
 real(4) :: rcond
 complex(4) :: a(Lda, n), b(n), fac(Ldfac, n), x(n), res(n)
 a = reshape((/      (1.0, 1.0),   (2.0, 3.0),  (3.0, -3.0),          &
                     (2.0, 1.0),   (5.0, 3.0),  (7.0, -5.0),          &
                    (-2.0, 1.0),  (-4.0, 4.0),   (5.0, 3.0) /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 b = (/(3.0,5.0), (22.0,10.0), (-10.0,4.0) /)
 call Lfccg(n, a, Lda, fac, Ldfac, ipvt, rcond)! Находим LU-разложение матрицы A
 print *, ' cond1(A) = ', 1.0 / rcond          ! Оценка числа обусловленности
 do j=1, 3                                     ! Решаем линейную систему Ax = b
```

```
  call Lficg(n, a, Lda, fac, Ldfac, ipvt, b, ipath, x, res)
  call wrcrn('x', 1, n, x, 1, 0)                ! Вывод решения
  b(2) = b(2) + cmplx(0.5, 0.5)                 ! Изменяем b(2)
 end do
end program Ltc
```

Результат:

```
cond1(A) =      42.799400

                          x
  ( 1.000,-1.000)     ( 2.000, 4.000)     ( 3.000, 0.000)
                          x
  ( 0.910,-1.061)     ( 1.986, 4.175)     ( 3.123, 0.071)
                          x
  ( 0.821,-1.123)     ( 1.972, 4.349)     ( 3.245, 0.142)
```

Пример для LFSCB, работающей с ленточной комплексной матрицей и решающей систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение матрицы A. Вычисляется обратная матрица для 4×4-матрицы с одной верхней и одной нижней кодиагоналями. Предполагается, что входная матрица хорошо обусловлена, и поэтому для ее разложения применяется LFTCB, а не LFCCB.

```
program tcb
 use msimsl, nouse => fac
 integer(4), parameter :: Lda = 3, Ldfac = 4, n = Ldfac, ipath = 1, nlca = 1, nuca = 1
 integer(4) :: ipvt(n), j
 complex(4) :: a(Lda, n), ainv(n, n), fac(Ldfac, n), rj(n)
 ! Ленточное представление матрицы A
 a = reshape((/     (0.0, 0.0),  (4.0, 0.0),  (-2.0, 2.0),  (-4.0, -1.0),          &
                   (-2.0, -3.0), (-0.5, 3.0), (3.0, -3.0),  (1.0, -1.0),          &
                    (6.0, 1.0),  (1.0, 1.0),  (0.0, 2.0),    (0.0, 0.0)/), shape=(/Lda,n/), order=(/2,1/))
 ! Находим LU-разложение матрицы A
 call Lftcb(n, a, Lda, nlca, nuca, fac, Ldfac, ipvt)
 rj = (0.0, 0.0)
 do j=1, n                                      ! Вычисляем обратную матрицу
  ! rj(j) – столбец с номером j единичной матрицы, поэтому LFSCB
  ! разместит столбец j обратной матрицы A^-1 в j-й столбец массива ainv
  rj(j) = (1.0, 0.0)
  call Lfscb(n, fac, Ldfac, nlca, nuca, ipvt, rj, ipath, ainv(1, j))
  rj(j) = (0.0, 0.0)
 end do
 call wrcrn('ainv', n, n, ainv, n, 0)           ! Вывод обратной матрицы
end program tcb
```

Результат:

ainv

	1	2	3	4
1	(0.165,-0.341)	(0.376,-0.094)	(-0.282, 0.471)	(-1.600, 0.000)
2	(0.588,-0.047)	(0.259, 0.235)	(-0.494, 0.024)	(-0.800,-1.200)
3	(0.318, 0.271)	(0.012, 0.247)	(-0.759,-0.235)	(-0.550,-2.250)
4	(0.588,-0.047)	(0.259, 0.235)	(-0.994, 0.524)	(-2.300,-1.200)

Пример для LSLTQ, решающей трехдиагональную линейную систему из четырех уравнений.

```
program Ltq
 use msimsl
 integer(4), parameter :: n = 4
 complex(4) :: b(n), c(n), d(n), e(n)
 character(8) :: clabel(1) = 'number', fmt = '(e11.4)', rlabel(1) = 'none'
 ! Векторы c, d, e и b содержат соответственно верхнюю кодиагональ,
 ! диагональ, нижнюю кодиагональ и правую часть системы уравнений
 c = (/ (0.0, 0.0), (-9.0, 3.0), (2.0, 7.0), (7.0, -4.0) /)
 d = (/ (3.0, -5.0), (4.0, -9.0), (-5.0, -7.0), (-2.0, -3.0) /)
 e = (/ (-9.0, 8.0), (1.0, 8.0), (8.0, 3.0), (0.0, 0.0) /)
 b = (/ (-16.0, -93.0), (128.0, 179.0), (-60.0, -12.0), (9.0,- 108.0) /)
 ! Решаем линейную систему Ax = b; результат размещается в b
 call Lsltq(n, c, d, e, b)
 call wrcrl('b', 1, n, b, 1, 0, fmt, rlabel, clabel)
end program Ltq
```

Результат:

```
                    ainv
                    1                          2
( -.4000E+01, -.7000E+01)    ( -.7000E+01,  .4000E+01)
                    3                          4
( .7000E+01, -.7000E+01)    ( .9000E+01,  .2000E+01)
```

Пример для LFCCT, выполняющей оценку числа обусловленности матрицы. Оценивается число обусловленности нижней треугольной 3×3-матрицы коэффициентов.

```
program cct
 use msimsl
 integer(4), parameter :: Lda = 3, n = Lda, ipath = 1
 real(4) :: rcond
 complex(4) :: a(Lda, n)
 ! Представление нижней треугольной матрицы A
 a = reshape((/    (-3.0, 2.0),  (-2.0, -1.0),  (-1.0, 3.0),          &
                   (0.0, 0.0),   (0.0, 6.0),    (1.0, -5.0),          &
                   (0.0, 0.0),   (0.0, 0.0),    (-4.0, 0.0) /), shape = (/ Lda, n /))
```

```
 call Lfcct(n, a, Lda, ipath, rcond)
 print *, ' cond1(A) = ', 1.0 / rcond              ! Оценка числа обусловленности
end program cct
```

Результат:

```
cond1(A) =      5.223334
```

Пример для LFSZG, решающей систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение матрицы A. Решается система уравнений с разреженной 6×6-матрицей

$$A=\begin{pmatrix} 10+7i & 0 & 0 & 0 & 0 & 0 \\ 0 & 3+2i & -3+0i & -1+2i & 0 & 0 \\ 0 & 0 & 4+2i & 0 & 0 & 0 \\ -2-4i & 0 & 0 & 1+6i & -1+3i & 0 \\ -5+4i & 0 & 0 & -5+0i & 12+2i & -7+7i \\ -1+12i & -2+8i & 0 & 0 & 0 & 3+7i \end{pmatrix}.$$

Пусть

$$x^{(1)} = (1 + i, 2 + 2i, 3 + 3i, 4 + 4i, 5 + 5i, 6 + 6i)^T,$$

тогда

$$b^{(1)} = Ax^{(1)} = (3 + 17i, -19 + 5i, 6 + 18i, -38 + 32i, -63 + 49i, -57 + 83i)^T.$$

Пусть также

$$x^{(2)} = (6 + 6i, 5 + 5i, 4 + 4i, 3 + 3i, 2 + 2i, 1 + i)^T,$$

тогда

$$b^{(2)} = Ax^{(2)} = (18 + 102i, -16 + 16i, 8 + 24i, -11 -11i, -63 + 7i, -132 + 106i)^T.$$

Разреженные координаты матрицы A содержат векторы *irow* и *jcol*.

```
program szg
 use msimsl, nouse => fac
 integer(4), parameter :: n = 6, nz = 15
 integer(4) :: iparam(6), ipath = 1, ipvt(n), irfac(3 * nz), irow(nz),   &
            jcfac(3 * nz), jcol(nz), jpvt(n), nfac, nl
 real(4) :: rparam(5)
 complex(4) :: a(nz), b(n, 2), fac(3 * nz), x(n)
 character(2) :: title(2) = (/ 'x1', 'x2' /)
 a(:) = (/          (3.0,7.0),    (3.0,2.0),  (-3.0,0.0),    (-1.0,3.0),    (4.0,2.0),     &
                   (10.0,7.0),   (-5.0,4.0),   (1.0,6.0),   (-1.0,12.0),   (-5.0,0.0),     &
                   (12.0,2.0),   (-2.0,8.0),  (-2.0,-4.0),    (-1.0,2.0),   (-7.0,7.0)/)
 b(:, 1) = (/      (3.0,17.0),  (-19.0,5.0),  (6.0,18.0), (-38.0,32.0), (-63.0,49.0),    (-57.0,83.0) /)
 b(:, 2) = (/    (18.0,102.0), (-16.0,16.0),  (8.0,24.0), (-11.0,-11.0),  (-63.0,7.0), (-132.0,106.0) /)
 irow = (/ 6, 2, 2, 4, 3, 1, 5, 4, 6, 5, 5, 6, 4, 2, 5 /)
 jcol = (/ 6, 2, 3, 5, 3, 1, 1, 4, 1, 4, 5, 2, 1, 4, 6 /)
 nfac = 3 * nz; iparam(1) = 0                    ! Используем заданные по умолчанию установки
 ! Выполняем LU-разложение матрицы A
```

```
call Lftzg(n, nz, a, irow, jcol, iparam, rparam, nfac, nl, fac, irfac, jcfac, ipvt, jpvt)
do i = 1, 2                                  ! Решаем Ax(i) = b(i)
  call Lfszg(n, nfac, nl, fac, irfac, jcfac, ipvt, jpvt, b(1, i), ipath, x)
  call wrcrn(title(i), 1, n, x, 1, 0)        ! Вывод результата
end do
end program szg
```

Результат:

```
                                    x1
( 1.00, 1.00)  ( 2.00, 2.00)  ( 3.00, 3.00)  ( 4.00, 4.00)  ( 5.00, 5.00)  ( 6.00, 6.00)

                                    x2
( 6.00, 6.00)  ( 5.00, 5.00)  ( 4.00, 4.00)  ( 3.00, 3.00)  ( 2.00, 2.00)  ( 1.00, 1.00)
```

6.5.2. Процедуры с комплексной эрмитовой матрицей

6.5.2.1. Список процедур

Описание подпрограмм, использующих комплексную эрмитову матрицу, дано в табл. 6.11. Вновь, как и для процедур с комплексной неэрмитовой матрицей, сведения о назначении, синтаксисе вызова и ошибках следует почерпнуть из табл. 6.6-6.8, содержащих описание аналогов, работающих с вещественной симметрической матрицей. Заметим, однако, что разложение Холецкого эрмитовой матрицы имеет вид $R^H R$ (а не $R^T R$), в котором R – комплексная верхняя треугольная матрица, а R^H – матрица сопряженная с матрицей R. То же справедливо и для разложений LL^T и UDU^T: первое заменяется на LL^H, второе – на UDU^H.

Таблица 6.11. Подпрограммы с комплексной эрмитовой матрицей

Подпрограмма	*Назначение*	*Аналог*
Комплексная эрмитова положительно определенная матрица		
LFCDH	См. описание аналога в табл. 6.6	LFCDS
LFTDH	То же	LFTDS
LFSDH	"	LFSDS
LFIDH	"	LFIDS
LSLDH	"	LSLDS
LSADH	"	LSADS
Комплексная эрмитова положительно определенная ленточная матрица		
LSLQB	См. описание аналога в табл. 6.6	LSLPB
LFCQH	То же	LFCQS
LFTQH	"	LFTQS
LFSQH	"	LFSQS
LFIQH	"	LFIQS

LSLQH	См. описание аналога в табл. 6.6	LSLQS
LSAQH	То же	LSAQS
Комплексная эрмитова положительно определенная разреженная матрица		
LNFZD	См. описание аналога в табл. 6.6	LNFXD
LFSZD	То же	LFSXD
LSLZD	"	LSLXD
Комплексная эрмитова неопределенная матрица		
LFCHF	См. описание аналога в табл. 6.6	LFCSF
LFTHF	То же	LFTSF
LFSHF	"	LFSSF
LFIHF	"	LFISF
LSLHF	"	LSLSF
LSAHF	"	LSASF

Замечание. В подпрограммах с комплексной эрмитовой матрицей адресуется только ее верхняя треугольная часть.

6.5.2.2. Возможные ошибки процедур

В табл. 6.12 указаны ошибки, которые могут возникнуть при вызове процедур из табл. 6.11 и которых нет в табл. 6.8, содержащей описание информационных ошибок процедур – аналогов.

Таблица 6.12. Некоторые возможные ошибки процедур из табл. 6.11

Тип	*Код*	*Причина ошибки*	*Где возникает*
3	4	Входная матрица не является эрмитовой. Она имеет на диагонали элемент с маленькой мнимой частью	LFCDH, LFTDH, LSLDH, LSADH, LFCQH, LFTQH, LSLQH, LSAQH
4	4	Входная матрица не является эрмитовой. Она имеет на диагонали элемент с мнимой частью	LFCDH, LFTDH, LSLDH, LSADH, LFCQH, LFTQH, LSLQH, LSAQH

6.5.2.3. Примеры для процедур с комплексной эрмитовой матрицей

Приведем по одному примеру для каждого представленного в табл. 6.12 вида матриц.

Пример для LFIDH, работающей с комплексной эрмитовой матрицей и решающей систему линейных уравнений $Ax = b$, применяя ранее найденное UDU^H-разложение эрмитовой матрицы A, и выполняющей итерационное уточнение решения. Решается система с несколькими правыми частями. Каждая новая правая часть получается путем прибавления после получения решения величины $0.5 + 0.5i$ ко второму элементу вектора b.

```
program ihf
 use msimsl, nouse => fac
 integer(4), parameter :: Lda = 3, Ldfac = 3, n = Lda
 integer(4) :: ipvt(n), j
 real(4) :: rcond
 complex(4) :: a(Lda, n), b(n), fac(Ldfac, n), x(n), res(n)
 a = reshape((/      (3.0, 0.0),  (1.0, -1.0),  (4.0, 0.0),                &
                     (1.0, 1.0),  (2.0, 0.0),  (-5.0, 1.0),                &
                     (4.0, 0.0),      (-5.0, -  (-2.0, 0.0) /), shape = (/ Lda, n /), order = (/ 2, 1 /))
                                        1.0),
 b = (/ (7.0, 32.0), (-39.0, -21.0), (51.0, 9.0) /)
 call Lfchf(n, a, Lda, fac, Ldfac, ipvt, rcond) ! Находим UDU^H-разложение матрицы A
 print *, ' cond1(A) = ', 1.0 / rcond           ! Оценка числа обусловленности
 do j=1, 3                                      ! Решаем линейную систему Ax = b
   call Lfihf(n, a, Lda, fac, Ldfac, ipvt, b, x, res)
   call wrcrn('x', 1, n, x, 1, 0)               ! Вывод решения
   b(2) = b(2) + cmplx(0.5, 0.5)                ! Изменяем b(2)
 end do
end program ihf
```

Результат:

```
cond1(A) =      4.175095

                                  x
     ( 2.00,  1.00)        (-10.00, -1.00)        ( 3.00,  5.00)
                                  x
     ( 2.040, 1.080)       (-9.927, -.927)        ( 2.933, 4.940)
                                  x
     ( 2.080, 1.160)       (-9.853, -.853)        ( 2.867, 4.880)
```

Примеры для LFSDH, LSAQH и LSLQB используют одну и ту же эрмитову положительно определенную ленточную матрицу

$$A=\begin{pmatrix} 2+0i & -1+i & 0+0i & 0+0i & 0+0i \\ -1-i & 4+0i & 1+2i & 0+0i & 0+0i \\ 0+0i & 1-2i & 10+0i & 0+4i & 0+0i \\ 0+0i & 0+0i & 0-4i & 6+0i & 1+i \\ 0+0i & 0+0i & 0+0i & 1-i & 9+0i \end{pmatrix} \tag{6.3}$$

с числом кодиагоналей $ncoda = 1$. Однако в LFSDH матрица задается массивом, содержащим все ее элементы, в LSAQH – массивом, содержащим ее ленточное представление, а в LSLQB – массивом, содержащим ее кодиагональное ленточное представление.

Пример для LFSDH, работающей с комплексной эрмитовой положительно определенной матрицей и решающей систему линейных уравнений $Ax = b$, применяя ранее найденное R^HR-разложение матрицы A. В примере

LFTDH вычисляет R^HR-разложение матрицы A, затем в цикле LFSDH решает систему линейных уравнений с эрмитовой 5×5-матрицей. Предполагается, что входная матрица хорошо обусловлена, в противном случае для разложения матрицы следует вызывать LFCDH.

```
program sdh
 use msimsl, nouse => fac
 integer(4), parameter :: Lda = 5, Ldfac = 5, n = Ldfac
 integer(4) :: i
 complex(4) :: a(Lda, n), b(n, 3), fac(Ldfac, n), x(n, 3)
 ! Задает только верхнюю треугольную часть матрицы A, поскольку нижнюю
 ! треугольную часть LFTDH не адресует
 a = reshape((/   (2.0, 0.0),  (-1.0, 1.0),  (0.0, 0.0),  (0.0, 0.0),  (0.0, 0.0),        &
                  (0.0, 0.0),  (4.0, 0.0),   (1.0, 2.0),  (0.0, 0.0),  (0.0, 0.0),        &
                  (0.0, 0.0),  (0.0, 0.0),  (10.0, 0.0),  (0.0, 4.0),  (0.0, 0.0),        &
                  (0.0, 0.0),  (0.0, 0.0),   (0.0, 0.0),  (6.0, 0.0),  (1.0, 1.0),        &
                  (0.0, 0.0),  (0.0, 0.0),   (0.0, 0.0),  (0.0, 0.0),  (9.0, 0.0) /),     &
                shape = (/ Lda, n /), order = (/ 2, 1 /))
 b = reshape((/                                                                           &
        (3.0,3.0),      (5.0,-5.0),      (5.0,4.0),      (9.0,7.0),     (-22.0,1.0),      &
        (4.0,0.0),     (15.0,-10.0),   (-12.0,-56.0),  (-12.0,10.0),     (3.0,-1.0),     &
      (29.0,-9.0),    (-36.0,-17.0),   (-15.0,-24.0),  (-23.0,-15.0),  (-23.0,-28.0) /),  &
                shape = (/ n, 3 /))
 call Lftdh(n, a, Lda, fac, Ldfac)          ! Находим R^HR-разложение матрицы A
 do i=1, 3                                  ! Вычисляем 3 решения
  call Lfsdh(n, fac, Ldfac, b(:, i), x(:, i))
 end do
 call wrcrn('x', n, 3, x, n, 0)             ! Вывод результата
end program sdh
```

Результат (решение содержит каждый столбец результата):

```
                          x
                  1                2                3
 1      ( 1.00,  0.00)   ( 3.00, -1.00)   ( 11.00, -1.00)
 2      ( 1.00, -2.00)   ( 2.00,  0.00)   ( -7.00,  0.00)
 3      ( 2.00,  0.00)   ( -1.00, -6.00)  ( -2.00, -3.00)
 4      ( 2.00,  3.00)   ( 2.00,  1.00)   ( -2.00, -3.00)
 5      ( -3.00, 0.00)   ( 0.00,  0.00)   ( -2.00, -3.00)
```

Пример для LSAQH, вычисляющей R^HR-разложение матрицы A, решающей систему линейных уравнений $Ax = b$ и выполняющей итерационное уточнение решения. Решается система уравнений с эрмитовой ленточной 5×5-матрицей (6.3).

```
program aqh
 use msimsl
```

```
integer(4), parameter :: Lda = 2, n = 5, ncoda = 1
complex(4) :: a(Lda, n), b(n), x(n)
! Ленточное представление эрмитовой матрицы A
a = reshape((/      (0.0, 0.0),  (-1.0, 1.0),  (1.0, 2.0),  (0.0, 4.0),  (1.0, 1.0),      &
                    (2.0, 0.0),  (4.0, 0.0),  (10.0, 0.0),  (6.0, 0.0),  (9.0, 0.0) /),   &
                  shape = (/ Lda, n /), order = (/ 2, 1 /))
b = (/ (1.0, 5.0), (12.0, -6.0), (1.0,- 16.0), (-3.0, -3.0), (25.0, 16.0) /)
call Lsaqh(n, a, Lda, ncoda, b, x)
call wrcrn('x', 1, n, x, 1, 0)                    ! Вывод результата
end program aqh
```

Результат:

```
                                  x
( 2.000, 1.000)   ( 3.000, 0.000)   (-1.000,-1.000)   ( 0.000,-2.000)   ( 3.000, 2.000)
```

Пример для LSLQB, работающей с комплексной эрмитовой ленточной положительно определенной матрицей A, вычисляющей R^HDR-разложение Холецкого и решающей систему $Ax = b$. Решается системы из пяти линейных уравнений. Матрица коэффициентов представляется массивом a, содержащим кодиагональное представление эрмитовой ленточной матрицы (6.3).

```
program useRHDR
 use msimsl
 integer(4), parameter :: n = 5, ncoda = 1, Lda = n + ncoda
 integer(4) :: ijob
 real(4) :: a(Lda, 2 * ncoda + 3), u(n)
 ! Кодиагональное представление эрмитовой ленточной матрицы A;
 ! в последнем столбце массива a – коэффициенты вектора b
 a = reshape((/      0.0,    0.0,    0.0,    0.0,    0.0,        &
                     2.0,    0.0,    0.0,    1.0,    5.0,        &
                     4.0,   -1.0,    1.0,   12.0,   -6.0,        &
                    10.0,    1.0,    2.0,    1.0,  -16.0,        &
                     6.0,    0.0,    4.0,   -3.0,   -3.0,        &
                     9.0,    1.0,    1.0,   25.0,   16.0 /),     &
                  shape = (/ Lda, 2*ncoda + 3 /), order = (/ 2, 1 /))
 ijob = 1                                  ! Находим R^HDR-разложение матрицы A
 call Lslqb(n, a, Lda, ncoda, ijob, u)     ! и решаем линейную систему Ax = b
 ! Вывод результата – действительной и мнимой частей. Действительная часть
 ! результата находится в предпоследнем столбце массива a, а мнимая – в последнем
 call wrrrn('real(x)', 1, n, a(ncoda + 1, 2 * ncoda + 2), 1, 0)
 call wrrrn('imag(x)', 1, n, a(ncoda + 1, 2 * ncoda + 3), 1, 0)
end program useRHDR
```

Результат:

```
                          real(x)
  2.000        3.000       -1.000        0.000        3.000
```

```
                    imag(x)
  1.000      0.000     -1.000      -2.000      2.000
```

Пример для LFSZD, решающей систему линейных уравнений $Ax = b$, применяя ранее найденное LL^H-разложение Холецкого матрицы A. Решается система уравнений с эрмитовой положительно определенной разреженной 3×3-матрицей

$$A = \begin{pmatrix} 2+0i & -1+i & 0 \\ -1-i & 4+0i & 1+2i \\ 0 & 1-2i & 10+0i \end{pmatrix}.$$

Пусть

$$x^{(1)} = (1 + i, 2 + 2i, 3 + 3i)^T,$$

тогда

$$b^{(1)} = Ax^{(1)} = (-2 + 2i, 5 + 15i, 36 + 28i)^T.$$

Пусть также

$$x^{(2)} = (3 + 3i, 2 + 2i, 1 + i)^T,$$

тогда

$$b^{(2)} = Ax^{(2)} = (2 + 6i, 7 - 5i, 16 + 8i)^T.$$

Разреженные координаты матрицы A содержат векторы *irow* и *jcol*:

irow	1	2	3	2	3,
jcol	1	2	3	1	2,
a	$2+0i$	$4+0i$	$10+0i$	$-1-i$	$1-2i$.

Число ненулевых элементов в нижней треугольной части матрицы $nz = 5$.

```
program szg
 use msimsl, nouse => fac
 integer(4), parameter :: n = 3, nz = 5, nrlnz = 5
 real(4) :: rparam(2)
 integer(4) :: ijob, ilnz(n + 1), invper(n), inzsub(n + 1), iper(n), irow(nz), ispace,  &
              itwksp, jcol(nz), maxnz, maxsub, nzsub(3 * nz)
 complex(4) :: a(nz), b1(n), b2(n), diag(n), rlnz(nrlnz), x(n)
 a = (/ (2.0, 0.0), (4.0, 0.0), (10.0, 0.0), (-1.0, -1.0), (1.0, -2.0) /)
 b1 = (/ (-2.0, 2.0), (5.0, 15.0), (36.0, 28.0) /); b2 = (/ (2.0, 6.0), (7.0, 5.0), (16.0, 8.0) /)
 irow = (/ 1, 2, 3, 2, 3 /); jcol = (/ 1, 2, 3, 1, 2 /)
 ! Минимальный уровень упорядочивания, применимый для многофронтального метода
 ijob = 3
 ! Используем заданное по умолчанию рабочее пространство
 itwksp = 0; maxsub = 3 * nz
 ! Ищем LL^H-разложение матрицы A
 ! Определяем структуру разложения
 call Lscxd(n, nz, irow, jcol, ijob, itwksp, maxsub, nzsub, &
```

```
          inzsub, maxnz, ilnz, iper, invper, ispace)
 if(nrlnz >= maxnz) then                    ! Проверяем значение nrlnz
  ijob = 2                                  ! Выбираем многофронтальный метод
  ! Числовое разложение разреженной эрмитовой матрицы
  call Lnfzd(n, nz, a, irow, jcol, ijob, maxsub, nzsub, inzsub, &
          maxnz, ilnz, iper, invper, ispace, itwksp, diag, rlnz, rparam)
  ! Решаем Ax(1) = b(1), применяя найденное LL^H-разложение
  call Lfszd(n, maxsub, nzsub, inzsub, maxnz, rlnz, ilnz, diag, iper, b1, x)
  call wrcrn(' x1 ', 1, n, x, 1, 0)         ! Вывод x1
  ! Решаем Ax(2) = b(2)
  call Lfszd(n, maxsub, nzsub, inzsub, maxnz, rlnz, ilnz, diag, iper, b2, x)
  call wrcrn(' x2 ', 1, n, x, 1, 0)         ! Вывод x2
 end if
end program szg
```

Результат:

```
                    x1
( 1.000, 1.000)   ( 2.000, 2.000)   ( 3.000, 3.000)
                    x2
( 3.000, 3.000)   ( 2.000, 2.000)   ( 1.000, 1.000)
```

6.6. ВЫЧИСЛЕНИЕ ОПРЕДЕЛИТЕЛЕЙ

6.6.1. Список процедур

Приводимые в табл. 6.13 подпрограммы вычисляют определители матриц разных видов, используя наиболее подходящие для конкретного вида матрицы разложения. В табл. 6.14 дается синтаксис вызова подпрограмм из табл. 6.13, работающих с матрицами типа REAL(4) и COMPLEX(4). Синтаксис вызова подпрограмм с матрицами типа REAL(8) и COMPLEX(8) аналогичен. Следует лишь добавить к имени подпрограммы букву D. В последней графе табл. 6.13 перечисляются подпрограммы 1-го уровня, возвращающие разложение одинарной точности.

Таблица 6.13. Подпрограммы, вычисляющие определители

Подпрограмма	*Вид матрицы*	*Разложение*	*Подпрограммы, возвращающие разложение*
Вещественная несимметрическая матрица			
LFDRG (DLFDRG)	Общего вида	*LU*	LFCRG, LFTRG
LFDRB (DLFDRB)	Ленточная	"	LFCRB, LFTRB
LFDRT (DLFDRT)	Треугольная	-	-

Вещественная симметрическая матрица			
LFDDS (DLFDDS)	Положительно определенная	R^TR	LFCDS, LFTDS
LFDQS (DLFDQS)	Положительно определенная ленточная	"	LFCQS, LFTQS
LFDSF (DLFDSF)	Неопределенная	UDU^T	LFCSF, LFTSF
Комплексная неэрмитова матрица			
LFDCG (DLFDCG)	Общего вида	LU	LFCCG, LFTCG
LFDCB (DLFDCB)	Ленточная	"	LFCCB, LFTCB
LFDCT (DLFDCT)	Треугольная	-	-
Комплексная эрмитова матрица			
LFDDH (DLFDDH)	Положительно определенная	R^HR	LFCDH, LFTDH
LFDQH (DLFDQH)	Положительно определенная ленточная	"	LFCQH, LFTQH
LFDHF (DLFDHF)	Неопределенная	UDU^H	LFCHF, LFTHF

Таблица 6.14. Синтаксис вызова подпрограмм из табл. 6.13

Синтаксис вызова
Вещественная несимметрическая матрица
CALL LFDRG(*n*, *fac*, *Ldfac*, *ipvt*, *det*1, *det*2)
CALL LFDRB(*n*, *fac*, *Ldfac*, *nlca*, *nuca*, *ipvt*, *det*1, *det*2)
CALL LFDRT(*n*, *a*, *Lda*, *det*1, *det*2)
Вещественная симметрическая матрица
CALL LFDSF(*n*, *fac*, *Ldfac*, *ipvt*, *det*1, *det*2)
CALL LFDDS(*n*, *fac*, *Ldfac*, *det*1, *det*2)
CALL LFDQS(*n*, *fac*, *Ldfac*, *ncoda*, *det*1, *det*2)
Комплексная неэрмитова матрица
CALL LFDCG(*n*, *fac*, *Ldfac*, *ipvt*, *det*1, *det*2)
CALL LFDCB(*n*, *fac*, *Ldfac*, *nlca*, *nuca*, *ipvt*, *det*1, *det*2)
CALL LFDCT(*n*, *a*, *Lda*, *det*1, *det*2)
Комплексная эрмитова матрица
CALL LFDHF(*n*, *fac*, *Ldfac*, *ipvt*, *det*1, *det*2)
CALL LFDDH(*n*, *fac*, *Ldfac*, *det*1, *det*2)
CALL LFDQH(*n*, *fac*, *Ldfac*, *ncoda*, *det*1, *det*2)

Поскольку подпрограммы вычисления определителей разных видов матриц схожи между собой, опишем подробно лишь LFDRG (DLFDRG). Далее приведем по одному примеру подпрограмм с вещественной симметрической, комплексной неэрмитовой и эрмитовой матрицами.

6.6.2. Параметры процедур

Процедуры возвращают определитель матрицы A в виде двух чисел $det1$ и $det2$, таких, что $\det A = det1 * 10^{det2}$. Если матрица не является комплексной эрмитовой, то тип $det1$ совпадает с типом матрицы A. В случае комплексной эрмитовой матрицы определитель является вещественным, поэтому для подобных матриц $det1$ – вещественный параметр с разновидностью типа, равной разновидности типа матрицы A. Параметр $det2$ в любом случае является вещественным. Разновидность типа $det2$ совпадает с разновидностью типа матрицы A. Все остальные параметры процедур являются входными. Их описание для всех видов матриц дано в табл. 6.4.

6.6.3. Описание подпрограммы LFDRG

В качестве образца приведем описание подпрограммы LFDRG (DLFDRG), использующей для вычисления определителя вещественной несимметрической общего вида матрицы A ее LU-разложение. Вызов LFDRG:

CALL LFDRG(n, fac, $Ldfac$, $ipvt$, $det1$, $det2$)

Входные данные: n, fac, $Ldfac$, $ipvt$.

Выходные данные: $det1$, $det2$.

$det1$ – скаляр, содержащий мантиссу определителя; величина $det1$ нормализуется так, что $1.0 \le |det1| < 10.0$ или $det1 = 0.0$.

$det2$ – скаляр, содержащий порядок определителя. При этом $\det A = det1 * 10^{det2}$.

Описание:

Первоначально находится LU-разложение матрицы A, например подпрограммой LFCRG. Далее определитель вычисляется по формуле $\det A = \det L * \det U$. При этом определитель треугольной матрицы U равен произведению ее диагональных элементов: $\det U = \prod_{i=1}^{n} u_{ii}$, где n – порядок матрицы; определитель матрицы L, являющейся произведением унитреугольной матрицы и матриц перестановок, равен $\det L = (-1)^k$, где k – число выполненных во время вычисления LU-разложения перестановок строк.

Пример. Вычисляется определитель вещественной общего вида 3×3-матрицы.

```
program det
 use msimsl, nouse => fac; use text_transfer
```

```
integer(4), parameter :: n = 3, Lda = n, Ldfac = n
integer(4) :: ipvt(n)
real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), det1, det2
a = reshape((/      33.0,   16.0,   71.0,                    &
                   -24.0,  -10.0,  -57.0,                    &
                    18.0,  -11.0,    7.0  /), shape = (/ Lda, Lda /), order = (/ 2, 1 /))
call Lftrg(n, a, Lda, fac, Ldfac, ipvt)          ! Находим LU-разложение матрицы A
call Lfdrg(n, fac, Ldfac, ipvt, det1, det2)      ! Вычисляем определитель (det1 и det2)
print "(1x, a, f7.3, ' * 10**', f3.1)", trim(ru_doswin('Определитель матрицы:', .false.)), det1, det2
end program det
```

Результат:

Определитель матрицы: -5.205 * 10**3.0

6.6.4. Примеры вычисления определителей

Пример 1. Вычисляется определитель вещественной симметрической положительно определенной ленточной 4×4-матрицы с двумя кодиагоналями.

```
program det02
 use msimsl, nouse => fac; use text_transfer
 integer(4), parameter :: Lda = 3, Ldfac = 3, n = 4, ncoda = 2
 real(4) :: a(Lda, n), fac(Ldfac, n), det1, det2
 ! Ленточное представление матрицы A
 a = reshape((/      0.0,    0.0,    1.0,   -2.0,          &
                     0.0,    2.0,    1.0,    3.0,          &
                     7.0,    6.0,    6.0,    8.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call Lftqs(n, a, Lda, ncoda, fac, Ldfac)          ! Находим R^T R-разложение матрицы A
 call Lfdqs(n, fac, Ldfac, ncoda, det1, det2)      ! Вычисляем определитель (det1 и det2)
 print "(1x, a, f7.3, ' * 10**', f3.1)", trim(ru_doswin('Определитель матрицы:', .false.)), det1, det2
end program det02
```

Результат:

Определитель матрицы: 1.186 * 10**3.0

Пример 2. Вычисляется определитель комплексной нижней треугольной 3×3-матрицы A. Определитель находится как произведение диагональных элементов матрицы A.

```
program det03
 use msimsl; use text_transfer
 integer(4), parameter :: Lda = 3, n = 3
 real(4) :: det2
 complex(4) :: a(Lda,Lda), det1
 a = reshape((/      (-3.0, 2.0),   (0.0, 0.0),   (0.0, 0.0),                &
                     (-2.0, -
                        1.0),       (0.0, 6.0),   (0.0, 0.0),                &
                     (-1.0, 3.0),   (1.0, -5.0),  (-4.0, 0.0) /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call Lfdct(n, a, Lda, det1, det2)                 ! Вычисляем определитель (det1 и det2)
```

```
 print "(1x, a, ' (', f4.1, ',', f4.1, ') * 10**', f3.1)",  &    ! Вывод результата
        trim(ru_doswin('Определитель матрицы:', .false.)), det1, det2
end program det03
```

Результат:

Определитель матрицы: (0.5, 0.7) * 10**2.0

Пример 3. Вычисляется определитель комплексной эрмитовой неопределенной 3×3-матрицы.

```
program det04
 use msimsl, nouse => fac; use text_transfer
 integer(4), parameter :: Lda = 3, Ldfac = 3, n = Lda
 integer(4) :: ipvt(n)
 real(4) :: det1, det2
 complex(4) :: a(Lda, n), fac(Ldfac, n)
 a = reshape((/      (3.0, 0.0),  (1.0, -1.0),  (4.0, 0.0),                  &
                     (1.0, 1.0),  (2.0, 0.0),  (-5.0, 1.0),                  &
                     (4.0, 0.0),       (-5.0, -   (-2.0, 0.0) /), shape = (/ Lda, n /), order = (/ 2, 1 /))
                                        1.0),
 call Lfthf(n, a, Lda, fac, Ldfac, ipvt)          ! Находим UDU^H-разложение матрицы A
 call Lfdhf(n, fac, Ldfac, ipvt, det1, det2)      ! Вычисляем определитель (det1 и det2)
 print "(1x, a, f7.3, ' * 10**', f3.1)", trim(ru_doswin('Определитель матрицы:', .false.)), det1, det2
end program det04
```

Результат:

Определитель матрицы: -1.500 * 10**2.0

7. МАТРИЧНЫЕ РАЗЛОЖЕНИЯ И ИХ ПЕРЕСЧЕТ

7.1. ВВЕДЕНИЕ

Процедуры библиотеки IMSL вычисляют и используют приведенные в табл. 7.1 разложения вещественных матриц. Также в таблице для каждого разложения указаны его арифметическая трудоемкость (во флопах) и основное применение.

Таблица 7.1. Матричные разложения

Разложение	*Трудоемкость*	*Применение*
LU	$2n^3/3$	Решение систем линейных уравнений с матрицей общего вида
LDU	"	Решение систем линейных уравнений с несимметрической трехдиагональной матрицей
R^TR	$n^3/3$	Решение систем линейных уравнений с симметрической положительно или неотрицательно определенной матрицей
RDR^T	"	Решение систем линейных уравнений с симметрической положительно определенной ленточной матрицей
LL^T	"	Решение систем линейных уравнений с симметрической положительно определенной разреженной матрицей
UDU^T	"	Решение систем линейных уравнений с симметрической неопределенной матрицей
QR	$4n^3/3$	Вычисление собственных значений и собственных векторов $n\times n$-матриц
QR	$2n^2(m - n/3)$	Решение методом наименьших квадратов проблемы $Ax \approx b$ с $m\times n$-матрицей ($m \geq n$)
SVD	$4m^3n + 22n^3$	Вычисление 2-нормы, числа обусловленности и ранга $m\times n$-матрицы

<u>*Замечание*</u>. Разложения пригодны и для решения иных задач, например для вычисления обратных матриц и определителей.

В настоящей главе рассматриваются имеющиеся в библиотеке IMSL процедуры получения QR- и SVD-разложений, а также пересчета R^TR- и QR-разложений. Сингулярное (SVD) разложение пересчету не поддается. Вопросы применения QR-разложений для решения указанных в табл. 7.1 задач обсуждаются в последующих главах. Методы получения разложений, используемых для решения систем линейных уравнений, описаны в гл. 5. Как и ранее, во всех приводимых в главе подпрограммах IMSL разновидность

типов вещественных и комплексных параметров при работе с одинарной точностью – KIND = 4, а с двойной – KIND = 8. Целочисленные параметры начинаются с букв *i*, *j*, *k*, *l*, *m* и *n* и имеют тип INTEGER(4). Как и ранее, опускается описание процедур 2-го уровня, к которым обращаются процедуры получения и пересчета разложений.

Изложение процедур IMSL предваряется описанием основных идей, используемых для получения *QR*-разложений матриц и их пересчета.

7.2. ПРЕОБРАЗОВАНИЯ ХАУСХОЛДЕРА И ГИВЕНСА

7.2.1. Преобразование Хаусхолдера

7.2.1.1. Матрица и вектор Хаусхолдера

Преобразованием или *отражением Хаусхолдера* называется $m \times n$-матрица

$$H = I - 2\frac{uu^T}{u^T u}, \tag{7.1}$$

где u – ненулевой вектор-столбец размера m; $u^T u$ – скалярное произведение (u, u). Матрица H и вектор u называются соответственно *матрицей* и *вектором Хаусхолдера*. Легко проверить, что матрица H является ортогональной и симметрической.

Преобразование Хаусхолдера позволяет привести произвольный вектор

$$b = (b_1, b_2, \ldots, b_n)^T$$

к вектору, в котором все элементы, кроме первого, равны нулю:

$$Hb = H(b_1, b_2, \ldots, b_n)^T = (\beta, 0, \ldots, 0)^T = \beta(1, 0, \ldots, 0)^T = \beta e^{(1)}.$$

Из приведенных соотношений найдем вектор u преобразования (7.1).

$$Hb = \left(I - 2\frac{uu^T}{u^T u}\right)b = b - 2\frac{u^T b}{u^T u}u = \beta e^{(1)}.$$

Отсюда

$$\rho u = b - \beta e^{(1)},$$

где $\rho = 2\frac{u^T b}{u^T u}$; здесь $u^T b$ – скалярное произведение (u, b). То есть вектор u коллинеарен вектору $b - \beta e^{(1)}$, и так как из (7.1) следует, что умножение u на отличную от нуля константу не изменяет преобразование Хаусхолдера, то в качестве u можно принять $b - \beta e^{(1)}$. В то же время, поскольку ортогональное преобразование сохраняет 2-норму (разд. 4.2.16), $\beta = \mu\|b\|_2$. Знак β можно

выбрать любым, однако для повышения точности он выбирается так, чтобы избежать вычитания, т. е. таким же, как и знак $b(1)$.

Пример. Выполняется преобразование Хаусхолдера вектора $b = (4, -5, 6, -7)^T$.

```
program hTest
 integer(4), parameter :: n = 4
 integer(4) :: i
 real(4) :: b(n) = (/ 4, -5, 6, -7 /), s2          ! s2 – квадрат 2-нормы вектора b
 real(4) :: u(n), dp                               ! u – вектор Хаусхолдера
 real(4), dimension(n, n) :: e, h                  ! Единичная матрица и матрица Хаусхолдера
 e = 0.0
 do i = 1, n; e(i, i) = 1.0; end do                ! Создаем единичную матрицу
 s2 = dot_product(b, b)
 u = b
 u(1) = u(1) + sign(sqrt(s2), b(1))                ! u = b - βe^(1) , где β = μ||b||_2
 dp = s2 + u(1) * u(1) - b(1) * b(1)               ! или dp = dot_product(u, u)
 ! Строим матрицу Хаусхолдера
 h = e - 2.0 * matmul(reshape(u, shape = (/ n, 1 /)), reshape(u, shape = (/ 1, n /))) / dp
 b = matmul(h, b)                                  ! Преобразование Хаусхолдера
 print '(1x, a, 10f7.3)', 'b: ', b
end program hTest
```

Результат:

```
b: -11.225  0.000  0.000  0.000
```

Замечания:

1. Действие матрицы Хаусхолдера H на вектор вызывает его отражение в n-мерном пространстве относительно гиперплоскости, проходящей через начало координат и имеющей нормальный вектор, равный u. Поэтому преобразование Хаусхолдера имеет иное название – *метод отражений*.
2. Приведенный код можно существенно оптимизировать по быстродействию и по памяти, отказавшись от явного использования матрицы Хаусхолдера.

7.2.1.2. Вычисление вектора Хаусхолдера

На практике нет необходимости строить матрицу Хаусхолдера: достаточно найти его вектор. Для упрощения алгоритмов, в которых применяется вектор Хаусхолдера, выполняется его нормировка, при которой $u(1) = 1$. В этой ситуации *существенным* является сечение $u(2:)$, которое можно записать на место обнуляемой части вектора b. Подпрограмма, возвращающая нормированный вектор Хаусхолдера u, может иметь вид:

```
subroutine hvec(n, b, u)
 integer(4) :: n
 real(4) :: b(n), u(n), s2                         ! s2 – 2-норма вектора b
```

```
 s2 = sqrt(dot_product(b, b))
 u = b
 if(s2 /= 0.0) then
  u(1) = b(1) + sign(s2, b(1))
  u(2:) = u(2:) / u(1)
 end if
 u(1) = 1.0
end subroutine Hvec
```

Возвращаемый подпрограммой *hvec* вектор u таков, что $u(1) = 1$ и все компоненты вектора $(1 - 2uu^T/u^Tu)b$, кроме первого, равны нулю.

7.2.1.3. Умножение матрицы на матрицу Хаусхолдера

Умножение матриц $\mathbf{R}^{m\times r}\times\mathbf{R}^{r\times n} \to \mathbf{R}^{m\times n}$ требует $2mnr$ флопов. Умножение на матрицу Хаусхолдера можно выполнить за $4mn$ флопов, приняв во внимание, что

$$HA=\left(I-2\frac{uu^T}{u^Tu}\right)A=A+uw^T,$$

где $w=\beta A^Tu$ и $\beta=-2/u^Tu$. Тогда произведение HA ($A \in \mathrm{R}^{m\times n}$) вернет код

```
! Подпрограмма rowHouse умножает матрицу A на матрицу Хаусхолдера слева
subroutine rowHouse(m, n, k, a, u)          ! u – вектор Хаусхолдера
 integer(4) :: m, n, k, i, j                ! k – номер обнуляемого столбца
 real(4) :: a(m, n), u(m), w(n), beta
 beta = -2.0 / dot_product(u(k:m), u(k:m))
 w = 0.0                                    ! Умножаем матрицу на вектор u и находим w
 do j = k, n                                ! Цикл вернет то же, что и MATMUL(A^T, u)
  w(j) = w(j) + dot_product(a(k:m, j), u(k:m))
 end do
 w(k:n) = beta * w(k:n)                     ! Завершаем вычисление вектора w
 do j = k, n                                ! Вычисляем A + uw^T
  do i = k, m
   a(i, j) = a(i, j) + u(i) * w(j)
  end do
 end do
end subroutine rowHouse
```

В результате вызова *rowHouse* произойдет обнуление столбца k матрицы A.

Для данных $m\times n$-матрицы и n-вектора u, где $u(1) = 1$, умножение AH, где $H = I - 2uu^T/u^Tu$, выполнит подпрограмма

```
! Подпрограмма colHouse умножает матрицу A на матрицу Хаусхолдера справа
subroutine colHouse(m, n, k, a, u)          ! u – вектор Хаусхолдера
 integer(4) :: m, n, k, i, j                ! k – номер обнуляемой строки
```

```
 real(4) :: a(m, n), u(n), w(m), beta
 beta = -2.0 / dot_product(u(k:n), u(k:n))
 w = 0.0                                  ! Умножаем матрицу на вектор u и находим w
 do i = k, m                              ! Цикл вернет то же, что и MATMUL(A, u)
  w(i) = w(i) + dot_product(a(i, k:n), u(k:n))
 end do
 w(k:m) = beta * w(k:m)                   ! Завершаем вычисление вектора w
 do j = k, n                              ! Вычисляем A + uw^T
  do i = k, m
   a(i, j) = a(i, j) + u(j) * w(i)
  end do
 end do
end subroutine colHouse
```

В результате вызова *colHouse* произойдет обнуление строки k матрицы A.

Алгоритмы преобразований Хаусхолдера, реализованные в colHouse и rowHouse, требуют 4mn флопов.

Пример 1. Обнуляются все, начиная со второго, элементы первого столбца матрицы A.

```
program a1
 use msimsl                               ! Для вызова WRRRN
 integer(4), parameter :: m = 4, n = 3
 real(4) :: a(m, n), u(m)
 a = reshape((/      1.0,    2.0,    4.0,                &
                     1.0,    4.0,   16.0,                &
                     1.0,    6.0,   36.0,                &
                     1.0,    8.0,   64.0  /), shape = (/ m, n /), order = (/ 2, 1 /))
 call hvec(m, a(1:, 1), u)                ! Вычисляем нормированный вектор Хаусхолдера
 call rowHouse(m, n, 1, a, u)             ! Находим произведение HA
 call wrrrn('HA', m, n, a, m, 0)          ! Вывод результата
end program a1                            ! Тексты подпрограмм hvec и rowHouse см. выше
```

Результат:

HA

	1	2	3
1	-2.00	-10.00	-60.00
2	0.00	0.00	-5.33
3	0.00	2.00	14.67
4	0.00	4.00	42.67

Пример 2. Обнуляются все, начиная со второго, элементы первой строки матрицы A.

```
program a2
 use msimsl                               ! Для вызова WRRRN
```

```
integer(4), parameter :: m = 3, n = 4
real(4) :: a(m, n), u(n)
a = reshape((/      1.0,    1.0,    1.0,    1.0,                &
                    2.0,    4.0,    6.0,    8.0,                &
                    8.0,   16.0,   36.0,   64.0  /), shape = (/ m, n /), order = (/ 2, 1 /))
call hvec(n, a(1, 1:), u)                    ! Вычисляем нормированный вектор Хаусхолдера
call colHouse(m, n, 1, a, u)                 ! Находим произведение AH
call wrrrn('AH', m, n, a, m, 0)              ! Вывод результата
end program a2                               ! Тексты подпрограмм hvec и colHouse см. выше
```

Результат:

AH

	1	2	3	4
1	-2.00	0.00	0.00	0.00
2	-10.00	0.00	2.00	4.00
3	-62.00	-7.33	12.67	49.67

7.2.1.4. Ошибки округления

Возвращаемый подпрограммой *hvec* вектор $\widehat{u}$ близок к точному вектору u, [15]. Причем, если матрица Хаусхолдера

$$\widehat{H} = \left(I - 2\frac{\widehat{u}\widehat{u}^T}{\widehat{u}^T\widehat{u}} \right),$$

то

$$\left\| \widehat{H} - H \right\|_2 = O(\varepsilon_м),$$

где H – точная матрица Хаусхолдера.

Преобразования Хаусхолдера с матрицей $\widehat{H}$ близки к точным преобразованиям с матрицей H:

$$fl(\widehat{H}A) = H(A + E),$$

где $\|E\|_2 = O(\varepsilon_м \|A\|_2)$.

7.2.2. Преобразование Гивенса

7.2.2.1. Матрица вращения Гивенса

Преобразованием или *вращением Гивенса*, как правило, пользуются для обнуления избранных элементов матрицы. Причем одно вращение приводит к обнулению одного элемента. Матрица вращения Гивенса является двуранговой модификацией единичной матрицы и имеет вид

$$G(i,\ k,\ \theta)=\begin{pmatrix} 1 & & & & & & & & 0 \\ & \ddots & & & & & & \iddots & \\ & & c_{ii} & & & & s_{ik} & & \\ & & & 1 & & 0 & & & \\ & & & & \ddots & & & & \\ & & & 0 & & 1 & & & \\ & & -s_{ki} & & & & c_{kk} & & \\ & \iddots & & & & & & \ddots & \\ 0 & & & & & & & & 1 \end{pmatrix},$$

где $c_{ii} = c_{kk} = c = \cos\theta$, $s_{ik} = s_{ki} = s = \sin\theta$, а θ – угол поворота, выполняемого против часовой стрелки в координатной плоскости (e_i, e_k). Если вектор y получается в результате поворота вектора x на угол θ ($y = Gx$), то его элемент y_k обратится в нуль, если

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}};\ s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}.$$

Матрица Гивенса является ортогональной и отличается от единичной четырьмя элементами, причем два из них расположены на ее диагонали.

Замечания:

1. Умножение вектора или матрицы на матрицу Гивенса называют *плоским вращением* (*преобразованием*) *Гивенса*.
2. Вращение Гивенса также именуется и *вращением Якоби*.

Пример. В цикле обнуляются последовательно все, кроме первого, элементы вектора x.

```
program giv1
 integer(4), parameter :: n = 4
 integer(4) :: i, j
 real(4) :: x(n) = (/ 4, -5, 6, -7 /), r, c, s, g(n, n)
 do i = n, 2, -1
  r = sqrt(x(i - 1)**2 + x(i)**2)
  c = x(i - 1) / r; s = -x(i) / r
  g = 0.0; do j = 1, 5;  g(j, j) = 1.0; end do
  g(i - 1, i - 1) = c;      g(i - 1, i) = s
  g(i, i - 1) = -s;         g(i, i) = c
  x = matmul(transpose(g), x)
 end do
 print '(1x, a, 10f7.3)', 'x: ', x                    ! x:  11.225  .000  .000  .000
end program giv1
```

Приведенная программа может быть серьезно улучшена. Во-первых, можно сделать более надежным вычисление квадратного корня – снизить вероятность переполнения и исчезновения порядка, вызывая функцию

```
subroutine cs(a, b, c, s)                    ! Подпрограмма возвращает c = cosθ и s =sinθ,
 real(4) :: a, b, c, s, t                    ! где θ – угол, при повороте на который обнуляется
 if(b == 0.0) then                           ! очередной элемент вектора
  c = 1.0; s = 0.0
 else if(abs(b) > abs(a)) then
  t = -a / b; s = 1.0 / sqrt(1.0 + t * t); c = s * t
 else
  t = -b / a; c = 1.0 / sqrt(1.0 + t * t); s = c * t
end if
end subroutine cs
```

В приведенном примере подпрограмма имеет вызов

```
call cs(x(i - 1), x(i), c, s)
```

Во-вторых, нет необходимости явно формировать матрицу Гивенса, так как умножение вектора на матрицу Гивенса изменяет на шаге i только два элемента вектора с номерами i - 1 и i; их новые значения вернет код

```
t1 = x(i - 1); t2 = x(i)
x(i - 1) = c * t1 - s * t2
x(i) = s * t1 + c * t2
```

Новая версия программы

```
program giv1_1
 integer(4), parameter :: n = 4
 integer(4) :: i
 real(4) :: x(n) = (/ 4, -5, 6, -7 /), c, s, t1, t2
 do i = n, 2, -1
  call cs(x(i - 1), x(i), c, s)
  t1 = x(i - 1); t2 = x(i)
  x(i - 1) = c * t1 - s * t2
  x(i) = s * t1 + c * t2
 end do
 print '(1x, a, 10f7.3)', 'x: ', x                ! x:  11.225   .000   .000   .000
end program giv1_1
```

7.2.2.2. Умножение матрицы на матрицу Гивенса

Умножение матрицы A на матрицу Гивенса $G(i, k, \theta)$ слева (здесь $i < k$) затрагивает только строки i и k. Причем элементы этих строк, расположенные в столбцах с номерами, меньшими i, не изменяются. Поэтому произведение $G(i, k, \theta)A$ ($A \in \mathbf{R}^{m\times n}$) вернет код

```
subroutine row_rot(m, n, a, i1, i2, c, s)      ! Подпрограмма возвращает GA на месте матрицы A
```

```
 integer(4) :: m, n, i1, i2, j                  ! i1 и i2 – номера изменяемых строк
 real(4) :: a(m, n), c, s, t1, t2               ! c = cosθ, s = sinθ
 do j = i1, n                                   ! Должно выполняться условие i1 < i2
   t1 = a(i1, j); t2 = a(i2, j)
   a(i1, j) = c * t1 - s * t2
   a(i2, j) = s * t1 + c * t2
 end do
end subroutine row_rot
```

Пример. Обнуляются все, начиная со второго, элементы первого столбца матрицы A.

```
program giv2
 use msimsl                                     ! Для вызова WRRRN
 integer(4), parameter :: m = 4, n = 3
 integer(4) :: i
 real(4) :: a(m, n), c, s
 a = reshape((/      1.0,    2.0,    4.0,          &
                     1.0,    4.0,   16.0,          &
                     1.0,    6.0,   36.0,          &
                     1.0,    8.0,   64.0  /), shape = (/ m, n /), order = (/ 2, 1 /))
 do i = 2, m
  call cs(a(1, 1), a(i, 1), c, s)               ! Вычисляем c и s
  call row_rot(m, n, a, 1, i, c, s)             ! Находим произведение GA
 end do
 call wrrrn('GA', m, n, a, m, 0)                ! Вывод результата
end program giv2                                ! Тексты подпрограмм cs и row_rot см. выше
```

Результат:

GA

	1	2	3
1	2.00	10.00	60.00
2	0.00	1.41	8.49
3	0.00	2.45	21.23
4	0.00	3.46	39.26

Аналогично умножение матрицы A на матрицу Гивенса $G(i, k, \theta)$ справа затрагивает только столбцы i и k. Здесь также не меняются элементы столбцов, расположенные в строках с номерами, меньшими i. Произведение $AG(i, k, \theta)$ ($A \in \mathbf{R}^{m\times n}$) вернет подпрограмма

```
subroutine col_rot(m, n, a, j1, j2, c, s)       ! Подпрограмма возвращает AG на месте матрицы A
 integer(4) :: m, n, j1, j2, i                  ! j1 и j2 – номера изменяемых столбцов
 real(4) :: a(m, n), c, s, t1, t2               ! c = cosθ, s = sinθ
 do i = j1, m                                   ! Должно выполняться условие j1 < j2
  t1 = a(i, j1); t2 = a(i, j2)
  a(i, j1) = c * t1 - s * t2
```

```
  a(i, j2) = s * t1 + c * t2
 end do
end subroutine col_rot
```

Пример. Обнуляются все, начиная со второго, элементы первой строки матрицы A.

```
program giv2_2
 use msimsl                                  ! Для вызова WRRRN
 integer(4), parameter :: m = 3, n = 4
 integer(4) :: j
 real(4) :: a(m, n), c, s
 a = reshape((/      1.0,    1.0,    1.0,    1.0,                &
                     2.0,    4.0,    6.0,    8.0,                &
                     8.0,   16.0,   36.0,   64.0  /), shape = (/ m, n /), order = (/ 2, 1 /))
 do j = 2, n
  call cs(a(1, 1), a(1, j), c, s)            ! Вычисляем c и s
  call col_rot(m, n, a, 1, j, c, s)          ! Находим произведение GA
 end do
 call wrrrn('AG', m, n, a, m, 0)             ! Вывод результата
end program giv2_2                           ! Тексты подпрограмм cs и col_rot см. выше
```

Результат:

AG

	1	2	3	4
1	2.00	0.00	0.00	0.00
2	10.00	1.41	2.45	3.46
3	62.00	5.66	19.60	38.11

7.2.2.3. Компактное представление матриц Гивенса

Матрица Гивенса $G(i, k, \theta)$ задается тремя числами – i, k и θ. Однако вместо угла поворота θ можно использовать возвращаемый функцией *rval* скаляр ρ:

```
function rval(c, s)
 real(4) :: rval, c, s
 if(c == 0.0) then
  rval = 1.0
 else if(abs(s) < abs(c)) then
  rval = sign(s / 2.0, c)
 else
  rval = sign(2.0 / c, s)
 end if
end function rval
```

По известному ρ элементы матрицы Гивенса вернет подпрограмма

```
subroutine csval2(rval, c, s)
```

```
 real(4) :: rval, c, s
 if(rval == 1.0) then
   c = 0.0; s = 1.0
 else if(abs(rval) < 1.0) then
   s = rval + rval; c = sqrt(1.0 - s * s)
 else
   c = 2.0 / rval; s = sqrt(1.0 - c * c)
 end if
end subroutine csval2
```

Употребление ρ вместо θ позволяет избежать применения тригонометрических функций для восстановления матрицы Гивенса, а также обратной тригонометрической функции, например arcsin, для вычисления θ.

При выполнении преобразований Гивенса значение $\rho^{(i)}$ записывается на место обнуляемого элемента – $a(i + 1, i)$ и может быть затем использовано для вычисления ортогональной матрицы $Q = G_t...G_1$, где t – общее количество поворотов. Матрицу Q вернет подпрограмма

```
subroutine qrest(n, a, q)
 integer(4) :: n, i
 real(4) :: a(n, n), q(n, n), c, s
 q = 0.0
 do i = 1, n; q(i, i) = 1.0; end do          ! Первоначально Q(0) = In
 do i = 1, n - 1
   call csval2(a(i + 1, i), c, s)            ! Вычисляем c и s
   call col_rot(n, n, q, i, i + 1, c, s)     ! Вычисляем Q(i) = Q(i - 1)G(i)
 end do
end subroutine qrest
```

Определенный выше скаляр ρ позволяет восстанавливать элементы матрицы Гивенса с точностью до знака, т. е. вместо матрицы G может быть восстановлена матрица $-G$. Если же важен знак матрицы, например в QR-алгоритме вычисления собственных значений, то для хранения элементов G можно использовать дополнительную память, например массив g формы (2, n). Тогда матрицу $Q = G_t...G_1$ восстановит подпрограмма

```
subroutine qrest2(n, g, q)
 integer(4) :: n, i
 real(4) :: g(2, n), q(n, n), c, s
 q = 0.0
 do i = 1, n; q(i, i) = 1.0; end do          ! Первоначально Q(0) = In
 do i = 1, n - 1
   c = g(1, i); s = g(2, i)                  ! Вычисляем c и s
   call col_rot(n, n, q, i, i + 1, c, s)     ! Вычисляем Q(i) = Q(i - 1)G(i)
 end do
end subroutine qrest2
```

7.3. МЕТОДЫ ВЫЧИСЛЕНИЯ QR-РАЗЛОЖЕНИЯ

7.3.1. QR-разложение прямоугольной матрицы с применением преобразования Хаусхолдера

Возможный способ получения QR-разложения прямоугольной $m\times n$-матрицы, в котором Q – ортогональная $m\times m$-матрица, а R – верхняя трапециевидная $m\times n$-матрица, – это использование преобразования Хаусхолдера. Возьмем, например, матрицу

$$A=\begin{pmatrix}1 & 2 & 4\\ 1 & 4 & 16\\ 1 & 6 & 36\\ 1 & 8 & 64\end{pmatrix}$$

и выполним ее QR-разложение.

Рассмотрим прежде последовательность получения матрицы R.

Согласно изложенной в разд. 7.2.1.1 схеме, чтобы обратить в нуль все, кроме первого, элементы столбца 1, необходимо в качестве вектора u, формирующего матрицу преобразования Хаусхолдера, взять

$$u=(1,1,1,1)+\sqrt{1^2+1^2+1^2+1^2}\,(1,0,0,0)^T=(3,1,1,1)^T.$$

Тогда матрица преобразования Хаусхолдера для шага 1

$$H_1=I-2\frac{uu^T}{u^Tu}=\begin{pmatrix}-0.5000 & -0.5000 & -0.5000 & -0.5000\\ -0.5000 & 0.8333 & -0.1667 & -0.1667\\ -0.5000 & -0.1667 & 0.8333 & -0.1667\\ -0.5000 & -0.1667 & -0.1667 & 0.8333\end{pmatrix},$$

а произведение H_1A равно

$$H_1A=\begin{pmatrix}-2.0 & -10.0 & -60.0\\ 0.0 & 0.0 & -5.33\\ 0.0 & 2.0 & 14.67\\ 0.0 & 4.0 & 42.67\end{pmatrix}.$$

На 2-м шаге обнуляются два нижних элемента столбца 2. Поэтому вектор

$$u=(0,2,4)^T+\sqrt{0^2+2^2+4^2}\,(1,0,0)^T=(4.47,2,4)^T,$$

а матрица преобразования

$$\overline{H}_2=\begin{pmatrix}0.0005 & -0.4472 & -0.8944\\ -0.4472 & 0.7999 & -0.4002\\ -0.8944 & -0.4002 & 0.1996\end{pmatrix}.$$

Причем к матрице H_1A применяется преобразование

$$H_2 = \begin{pmatrix} 1 & 0 \\ 0 & \overline{H_2} \end{pmatrix},$$

в результате чего получим

$$H_2 H_1 A = \begin{pmatrix} -2.0 & -10.0 & -60.0 \\ 0.0 & -4.47 & -44.72 \\ 0.0 & 0.0 & -2.95 \\ 0.0 & 0.0 & 7.44 \end{pmatrix}.$$

После обработки столбца 3 получим верхнюю трапециевидную матрицу

$$R = H_3 H_2 H_1 A = \begin{pmatrix} 1.0 & 0.0 & 0.000 & 0.000 \\ 0.0 & 1.0 & 0.000 & 0.000 \\ 0.0 & 0.0 & -0.369 & 0.930 \\ 0.0 & 0.0 & 0.930 & 0.369 \end{pmatrix} H_2 H_1 A = \begin{pmatrix} -2.0 & -10.0 & -60.0 \\ 0.0 & -4.47 & -44.72 \\ 0.0 & 0.0 & 8.00 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}.$$

Поскольку

$$R = H_3 H_2 H_1 A,$$

и матрицы H_k являются ортогональными и симметрическими, то

$$A = H_1^T H_2^T H_3^T R = QR,$$

т. е. $Q = H_1^T H_2^T H_3^T = H_1 H_2 H_3$.

Матрица R формируется в цикле

```
do k = 1, n
 call hvec(m - k + 1, a(k:m, k), u(k:m))     ! Вычисляем нормированный вектор Хаусхолдера
 call rowHouse(m, n, k, a, u)                ! Обнуляем столбец k матрицы A
 a(k + 1:m, k) = u(k + 1:m)                  ! Запоминаем вектор Хаусхолдера в обнуляемой
end do                                       ! части матрицы A
```

С целью экономии памяти ортогональная матрица Q, как правило, явно не вычисляется. Вместо этого в k-м столбце матрицы A на месте его обнуляемых элементов записываются существенные элементы соответствующего вектора Хаусхолдера, равного

$$u^{(k)} = \left(\underbrace{0,\ 0,\ \ldots,\ 0}_{k-1},\ 1,\ u_{k+1}^{(k)},\ \ldots,\ u_n^{(k)} \right).$$

Далее при необходимости данные о векторах Хаусхолдера используются для восстановления матрицы $Q = H_1 \ldots H_n$. Причем вычисление Q выполняется уже существующей подпрограммой *rowHouse*:

```
do i = 1, m; q(i, i) = 1.0; end do           ! Инициализация матрицы Q
do k = n, 1, -1                              ! Используем обратное накопление
 u(k) = 1; u(k + 1:m) = a(k + 1:m, k)
 call rowHouse(m, n, k, q, u)                ! Q = H1...HnIm
end do
```

В цикле с параметром k по известному вектору Хаусхолдера $u^{(k)}$ на каждом шаге вычисляется $Q = H_k Q$, где

$$H_k = I - 2\frac{u^{(k)}u^{(k)T}}{u^{(k)T}u^{(k)}},$$

и до начала вычислений Q инициализируется как единичная матрица.

Алгоритм требует $2n^2(m - n/3)$ флопов.

Теперь можно написать программу вычисления QR-разложения прямоугольной $m \times n$-матрицы.

```
program qrmat
 use msimsl                                  ! Для вызова WRRRN
 integer(4), parameter :: nca = 3, Lda = 4
 integer(4) :: i
 real(4) :: a(Lda, nca), u(Lda), r(Lda, nca), q(Lda, Lda)
 a = reshape((/      1.0,    2.0,    4.0,                  &
                     1.0,    4.0,   16.0,                  &
                     1.0,    6.0,   36.0,                  &
                     1.0,    8.0,   64.0  /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
 ! Поиск QR-разложения. Матрица R возвращается на место матрицы A
 call QRhous(Lda, nca, a)                    ! Возвращает матрицу R и векторы Хаусхолдера
 ! Выводим матрицу R – верхнюю треугольную часть матрицы A
 call wrrrn('R', Lda, nca, a, Lda, 1)
 ! Формируем для вычисления контрольного произведения QR матрицу Q
 call getQ(Lda, nca, a, u, q)
 ! Функция MATMUL(q, r) вернет исходную матрицу A
 do i = 1, nca; r(i, i:) = a(i, i:); end do  ! Матрица R введена для проверки
 call wrrrn('QR', Lda, nca, matmul(q, r), Lda, 0)
end program qrmat
subroutine QRhous(m, n, a)                   ! Вычисление QR-разложения
 integer(4) :: m, n, k
 real(4) :: a(m, n), u(m)
 do k = 1, n
  call hvec(m - k + 1, a(k:m, k), u(k:m))    ! Вычисляем нормированный вектор Хаусхолдера
  call rowHouse(m, n, k, a, u)               ! Обнуляем столбец k матрицы A
  a(k + 1:m, k) = u(k + 1:m)                 ! Запоминаем вектор Хаусхолдера в обнуляемой
 end do                                      ! части матрицы A
end subroutine Qrhous                        ! Подпрограммы hvec и rowHouse приведены выше
subroutine getQ(m, n, a, u, q)               ! Подпрограмма формирует матрицу Q
 integer(4) :: m, n, i, k
 real(4) :: a(m, n), u(m), q(m, m)
 do i = 1, m; q(i, i) = 1.0; end do          ! Инициализация матрицы Q
 do k = n, 1, -1
   u(k) = 1; u(k + 1:) = a(k + 1:, k)
```

```
  call rowHouse(m, n, k, q, u)          ! Q = H1...HnIm
 end do
end subroutine getQ
```

Результат:

	R		
	1	2	3
1	-2.00	-10.00	-60.00
2		-4.47	-44.72
3			8.00

Приведем для сравнения неоптимизированный код вычисления *QR*-разложения, в котором явно формируются матрицы Хаусхолдера и ортогональная матрица *Q*.

```
subroutine QRhous(m, n, a, q)          ! Вычисление QR-разложения
 use msimsl                            ! Для вызова SSET
 real(4) :: a(m, n), q(m, m), u(m, 1), p(m, m), e(m, m)
 real(4) :: s, rk
 integer(4) :: k
 e = 0.0                               ! Инициализация массива e
 ! Присваиваем диагональным элементам массива e единицы
 call sset(m, 1.0, e, m + 1)           ! Массив e представляет единичную матрицу
 q = e                                 ! Инициализация массива q
 do k = 1, n
  s = sqrt(sum(a(k:, k)**2))           ! 2-норма
  s = sign(s, a(k, k))                 ! Ее знак
  u(:k-1, 1) = 0                       ! Формируем вектор u, применяемый
  u(k, 1) = a(k, k) + s                ! для построения матрицы преобразования
  u(k + 1:, 1) = a(k + 1:, k)          ! Хаусхолдера
  rk = dot_product(u(k:, 1), u(k:, 1))
  p = e - 2.0 * matmul(u, transpose(u)) / rk
  a = matmul(p, a)
  q = matmul(p, q)                     ! Формируем матрицу Q^T
 end do
 q = transpose(q)                      ! Возвращаем матрицу Q
end subroutine QRhous
```

7.3.2. *QR*-разложение симметрической матрицы

Учет симметричности матрицы позволяет существенно ускорить ее *QR*-разложение. Разложение выполняется в 2 этапа. На первом симметрическая матрица в результате преобразований Хаусхолдера приводится к трехдиагональному виду. На втором используются вращения Гивенса, под действием которых трехдиагональная матрица представляется в виде произведения ортогональной матрицы *Q* и верхней треугольной матрицы *R*.

7.3.2.1. Метод Хаусхолдера приведения симметрической матрицы к трехдиагональному виду

В *методе Хаусхолдера* вещественная симметрическая $n \times n$-матрица A приводится к трехдиагональному виду за n - 2 шагов. На каждом из них выполняются преобразования

$$A_k = H_k A_{k-1} H_k^T, \tag{7.2}$$

где $A_0 = A$. Каждая преобразовывающая матрица H_k является матрицей Хаусхолдера

$$H = I - 2\frac{uu^T}{u^T u}, u \in \mathbf{R}^{n-k}, u \neq 0.$$

Преобразование (7.2) позволяет на шаге k обратить в нуль все стоящие вне трех диагоналей элементы k-го столбца и k-й строки. Поэтому для приведения симметрической матрицы к трехдиагональному виду требуется n - 2 преобразований.

Рассмотрим в качестве примера вещественную симметрическую матрицу

$$A = \begin{pmatrix} 5 & -4 & 1 & 1 & 1 & 1 \\ -4 & 6 & -4 & 1 & 1 & 1 \\ 1 & -4 & 6 & -4 & 1 & 1 \\ 1 & 1 & -4 & 6 & -4 & 1 \\ 1 & 1 & 1 & -4 & 6 & -4 \\ 1 & 1 & 1 & 1 & -4 & 5 \end{pmatrix}, \tag{7.3}$$

и найдем, применив метод отражений, ее трехдиагональное представление. Выполним в подпрограмме *hous* в первом варианте решения задачи явное вычисление матрицы Хаусхоллдера.

```
program tred
 use msimsl                                  ! Для вызова WRRRN
 integer(4), parameter :: n = 6
 real(4) :: a(n, n)
 a = reshape((/      5.0,  -4.0, -10.0,   1.0,   1.0,  -2.0,          &
                    -4.0,   6.0,  -4.0,   1.0,   1.0,   1.0,          &
                   -10.0,  -4.0,   6.0,  -4.0,   1.0,   1.0,          &
                     1.0,   1.0,  -4.0,   6.0,  -4.0,  -3.0,          &
                     1.0,   1.0,   1.0,  -4.0,   6.0,  -4.0,          &
                    -2.0,   1.0,   1.0,  -3.0,  -4.0,   5.0 /), shape = (/ n, n /), order = (/ 2, 1 /))
 ! Приводим исходную симметрическую матрицу к трехдиагональному виду
 call hous(n, a)
 call wrrrn('A', n, n, a, n, 0)              ! Вывод результата
end program tred
subroutine hous(n, a)
```

```
! Подпрограмма реализует метод отражений и преобразовывает
! вещественную симметрическую матрицу втрехдиагональный вид
use msimsl                                  ! Для вызова SSET
integer(4) :: n, k
real(4) :: a(n, n), u(n, 1), h(n, n), e(n, n)   ! u, h, e – динамические массивы
real(4) :: dp
e = 0.0                                     ! Инициализация массива e
! Присваиваем диагональным элементам массива e единицы
call sset(n, 1.0, e, n + 1)                 ! Массив e представляет единичную матрицу
do k = 1, n - 2
  u(:k, 1) = 0                              ! Вычисляем нормированный вектор Хаусхолдера
  call hvec(n - k, a(k+1:, k), u(k+1:, 1))
  dp = dot_product(u(k + 1:, 1), u(k + 1:, 1))
  h = e - 2.0 * matmul(u, transpose(u)) / dp
  a = matmul(matmul(h, a), transpose(h))
end do
end subroutine hous
```

Результат:

A

	1	2	3	4	5	6
1	5.00	11.05	0.00	0.00	0.00	0.00
2	11.05	4.33	5.19	0.00	0.00	0.00
3	0.00	5.19	9.12	1.92	0.00	0.00
4	0.00	0.00	1.92	8.75	2.35	0.00
5	0.00	0.00	0.00	2.35	0.89	4.39
6	0.00	0.00	0.00	0.00	4.39	5.90

Заметим, что такой код приводит к неоправданным издержкам памяти и времени. Пример кода, лишенного этих недостатков, можно найти в [16].

7.3.2.2. QR-разложение симметрической трехдиагональной матрицы

При решении проблемы собственных значений и собственных векторов симметрической матрицы A на одном из этапов приходится выполнять ее QR-разложение, т. е. представлять ее в виде произведения ортогональной матрицы Q и верхней треугольной матрицы R ($A = QR$). Разложение получается достаточно просто. Для этого потребуется n - 1 вращений. Пусть дана трехдиагональная матрица

$$A=\begin{pmatrix} a_{11} & a_{12} & 0 & \dots & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & \dots & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n-2,n-2} & a_{n-2,n-1} & 0 \\ 0 & 0 & 0 & \dots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \dots & 0 & a_{n,n-1} & a_{nn} \end{pmatrix}.$$

Для получения верхней треугольной матрицы R необходимо исключить (сделать равным нулю) из каждой i-й строки A ($i = 2, \dots, n$) элемент $a_{i,i-1}$. Используем для этого метод вращения.

Исключение a_{21} выполняется в результате умножения слева матрицы $A_0 = A$ на матрицу

$$G_{12}=\begin{pmatrix} c_{12} & s_{12} & 0 & 0 & \dots & 0 \\ -s_{12} & c_{12} & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

где

$$c_{12}=\frac{a_{11}}{\sqrt{a_{11}^2+a_{21}^2}},\ s_{12}=\frac{a_{21}}{\sqrt{a_{11}^2+a_{21}^2}}.$$

Легко проверить, что $c_{12}^2+s_{12}^2=1$ и $-s_{12}a_{11}+c_{12}a_{21}=0$.

После выполнения умножения $G_{12}A_0$ будет получена матрица

$$A_1=G_{12}A_0=\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & 0 & \dots & 0 & 0 & 0 \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n-2,n-2} & a_{n-2,n-1} & 0 \\ 0 & 0 & 0 & \dots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \dots & 0 & a_{n,n-1} & a_{nn} \end{pmatrix},$$

в которой изменены две первые строки A_0 и $a_{21} = 0$.

Матрицы плоского вращения являются ортогональными, поэтому $G_{12}^T=G_{12}^{-1}$.

Второе вращение, приводящее к исключению a_{32}, выполняется в плоскости e_2e_3 в результате применения матрицы

$$G_{23} = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & c_{23} & s_{23} & 0 & \dots & 0 \\ 0 & -s_{23} & c_{23} & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

где

$$c_{23} = \frac{a_{22}}{\sqrt{a_{22}^2 + a_{32}^2}}, \; s_{12} = \frac{a_{32}}{\sqrt{a_{22}^2 + a_{32}^2}}.$$

Выполнив аналогичным образом оставшиеся n - 4 вращений, получим верхнюю треугольную матрицу

$$R = Q^T A,$$

где $Q^T = G_{n-1,\,n} G_{n-2,\,n-1} \dots G_{12}$ – матрица результирующего вращения. Она ортогональна, так как является произведением ортогональных матриц. Вычислив $Q = Q^{-T}$, получим QR-разложение матрицы $A = QR$. Общая трудоемкость равна около $3n^2$ флопов.

Пример. Вычисляется QR-разложение матрицы (7.3) и восстанавливается ортогональная матрица Q. Предварительно матрица (7.3) приводится процедурой *hous* к трехдиагональному виду. Затем выполняются вращения Гивенса. Поскольку матрица, подвергаемая преобразованиям Гивенса, является трехдиагональной и симметрической, то подпрограмму *row_rot* умножения на матрицу Гивенса можно упростить, заменив ее на *row_rot*2. Матрица Q восстанавливается подпрограммой *qrest*2 по значениям, хранящимся в массиве g, содержащем элементы матриц Гивенса. Заполнение массива g осуществляется в подпрограмме *qrfac*. Для проверки выводится произведение QR, которое равно исходной матрице A.

```
program giv2
 use msimsl
 integer(4), parameter :: n = 6
 real(4) :: a(n, n), g(2, n) = 0.0, q(n, n)
 a = reshape((/      5.0,  -4.0, -10.0,   1.0,   1.0,  -2.0,                  &
                    -4.0,   6.0,  -4.0,   1.0,   1.0,   1.0,                  &
                   -10.0,  -4.0,   6.0,  -4.0,   1.0,   1.0,                  &
                     1.0,   1.0,  -4.0,   6.0,  -4.0,  -3.0,                  &
                     1.0,   1.0,   1.0,  -4.0,   6.0,  -4.0,                  &
                    -2.0,   1.0,   1.0,  -3.0,  -4.0,   5.0 /), shape = (/ n, n /), order = (/ 2, 1 /))
 ! Приводим исходную симметрическую матрицу к трехдиагональному виду
 call hous(n, a)
 ! Выполняем QR-разложение трехдиагональной матрицы A
```

```
 call qrfac(n, a, g)
 ! Восстанавливаем ортогональную матрицу Q = G1...G5
 call qrest2(n, g, q)
 call wrrrn('R', n, n, a, n, 1)              ! Вывод верхней треугольной матрицы
 call wrrrn('Q', n, n, q, n, 0)              ! Вывод матрицы Q = G1...G5
 call wrrrn('QR', n, n, matmul(q, a), n, 0)  ! Вывод матрицы QR = A
end program giv2
subroutine qrfac(n, a, g)
 integer(4) :: n, i
 real(4) :: a(n, n), g(2, n), c, s
 do i = 1, n - 1                             ! Текст cs приведен в разд. 7.2.2.1
  call cs(a(i, i), a(i + 1, i), c, s)        ! Вычисляем элементы матрицы Гивенса
  call row_rot2(n, a, i, c, s)               ! Обнуление элемента a(i + 1, i)
   g(1, i) = c; g(2, i) = s                  ! Запоминаем элементы матрицы Гивенса
end do
end subroutine qrfac
subroutine row_rot2(n, a, i, c, s)           ! Подпрограмма возвращает GA на месте матрицы A
 integer(4) :: n, i, j                       ! Меняются строки i и i + 1
 real(4) :: a(n, n), c, s, t1, t2            ! c = cosθ, s = sinθ
 do j = i, min(i + 2, n)                     ! В каждой строке изменяются по 3 элемента
  t1 = a(i, j); t2 = a(i + 1, j)
  a(i, j) = c * t1 - s * t2
  a(i + 1, j) = s * t1 + c * t2
 end do
end subroutine row_rot2
```

Результат:

R

	1	2	3	4	5	6
1	12.12	8.50	4.73	0.00	0.00	0.00
2		9.77	3.03	1.02	0.00	0.00
3			9.07	3.44	0.50	0.00
4				8.54	2.45	1.21
5					4.40	6.11
6						3.91

Q

	1	2	3	4	5	6
1	-0.4124	0.7719	-0.4729	0.0984	0.0015	-0.0281
2	-0.9110	-0.3494	0.2141	-0.0445	-0.0007	0.0127
3	0.0000	0.5311	0.8281	-0.1723	-0.0026	0.0492
4	0.0000	0.0000	0.2115	0.9397	0.0140	-0.2686
5	0.0000	0.0000	0.0000	0.2751	-0.0500	0.9601
6	0.0000	0.0000	0.0000	0.0000	-0.9986	-0.0520

7.4. *QR*-РАЗЛОЖЕНИЕ ПРОЦЕДУРАМИ IMSL

7.4.1. *QR*-разложение прямоугольной матрицы

QR-разложение матрицы $AP = QR$, где A – $m \times n$-матрица, а P – матрица перестановок, с использованием преобразований Хаусхолдера выполняет подпрограмма LQRRR (DLQRRR):

CALL LQRRR(*nra*, *nca*, *a*, *Lda*, *pivot*, *ipvt*, *qr*, *Ldqr*, *qraux*, *conorm*)

Входные данные: *nra*, *nca*, *a*, *Lda*, *pivot*, *Ldqr*.

Входные/выходные данные: *ipvt*.

Выходные данные: *qr*, *qraux*, *conorm*.

nra и *nca* – соответственно число рядов и столбцов в матрице A.

a – вещественный массив формы (*Lda*, *nca*), содержащий *nra*×*nca*-матрицу A.

Lda – ведущий размер массива *a*.

pivot – скаляр типа LOGICAL(4); если *pivot* = .TRUE., то допускаются перестановки столбцов; если *pivot* = .FALSE., то перестановки не выполняются.

ipvt – целочисленный вектор размера *nca*, содержащий информацию о выполненных перестановках столбцов во время *QR*-разложения матрицы A. На входе если $ipvt(k) > 0$, то k-й столбец массива *a* является начальным; если $ipvt(k) = 0$, то k-й столбец массива *a* является свободным (может быть переставлен); если $ipvt(k) < 0$, то k-й столбец *a* является конечным. На выходе $ipvt(k)$ содержит индекс столбца *a*, который был переставлен на место k-го столбца. Вектор *ipvt* определяет матрицу перестановок P; *ipvt* адресуется, если *pivot* = .TRUE..

qr – вещественный массив формы (*Ldqr*, *nca*), содержащий необходимую для восстановления *QR*-разложения информацию. Верхняя трапециевидная часть *qr* содержит верхнюю трапециевидная часть матрицы R с диагональными элементами, отсортированными в возрастающем порядке. Строго нижняя трапециевидная часть *qr* содержит информацию, необходимую для восстановления ортогональной матрицы Q из *QR*-разложения. Если после работы LQRRR данные массива *a* не нужны, то *a* и *qr* могут быть одним и тем же массивом.

Ldqr – ведущий размер массива *qr*.

qraux – вещественный вектор размера *nca*, содержащий информацию об ортогональной части *QR*-разложения в первых min(*nra*, *nca*) столбцах; нужен для восстановления матрицы Q.

conorm – вещественный вектор размера *nca*, содержащий нормы столбцов входной матрицы. Если эти данные после работы LQRRR не нужны, то *conorm* и *qraux* могут быть одним и тем же вектором.

Описание:

Подпрограмма LQRRR основана на процедуре SQRDC пакета LINPACK, [34]; LQRRR определяет ортогональную матрицу Q, матрицу перестановок P и верхнюю трапециевидную матрицу R, диагональные элементы которой расположены в неубывающем порядке. При этом $AP = QR$. Преобразования Хаусхолдера, применяемые для получения QR-разложения, описаны в разд. 7.3.1. Матрица Q не возвращается LQRRR непосредственно, но информация, необходимая для восстановления Q, сохраняется. Если матрица Q нужна явно, то после LQRRR можно вызвать LQERR, которая извлекает матрицу Q из QR-разложения.

До выполнения разложения начальные столбцы перемещаются в начало массива a, а конечные – в его конец. Как начальные, так и конечные столбцы во время разложения остаются неподвижными. Могут переставляться только свободные столбцы. Перестановке, если она необходима, подвергаются столбцы с наибольшими нормами.

Когда вектором *ipvt* заданы перестановки, оценка числа обусловленности матрицы A может быть получена по выходным данным как отношение $qr(1, 1)/qr(k, k)$, где $k = \min(nra, nca)$. Эта оценка может быть использована при работе с подпрограммой LQRSL для выбора числа столбцов (*kbasis*) матрицы A_k.

Пример. Вычисляется QR-разложение 4×3-матрицы A и формируется матрица R этого разложения.

```
program QRfact
 use msimsl
 integer(4), parameter :: nca = 3, nra = 4, Lda = nra, Ldqr = nra
 integer(4) :: ipvt(nca), i
 real(4) :: a(Lda, nca), conorm(nca), qr(Ldqr, nca), qraux(nca), r(nra, nca)
 logical(4) :: pivot
 a = reshape((/      1.0,    2.0,    4.0,                &
                     1.0,    4.0,   16.0,                &
                     1.0,    6.0,   36.0,                &
                     1.0,    8.0,   64.0  /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
 pivot = .true.; ipvt = 0                    ! Все столбцы являются свободными
 ! Находим QR-разложение матрицы A
 call Lqrrr(nra, nca, a, Lda, pivot, ipvt, qr, Ldqr, qraux, conorm)
 r= 0.0                                      ! r – верхняя трапециевидная часть qr
 do i =1, nra                                ! Формируем матрицу R
   r(i, i:nca) = qr(i, i:nca)
 end do
 call wrrrn('conorm', 1, nca, conorm, 1, 0)  ! Вывод результата
 call wrrrn('R', nra, nca, r, nra, 0)        ! Выводим матрицу R
end program QRfact
```

Результат:

```
              conorm
   2.00        10.95          75.26

                R
              1          2          3
  1        -75.26     -10.63     -1.59
  2          0.00      -2.65     -1.15
  3          0.00       0.00      0.36
  4          0.00       0.00      0.00
```

7.4.2. Восстановление ортогональной матрицы *Q* из *QR*-разложения прямоугольной матрицы

Выполняет подпрограмма LQERR (DLQERR):

CALL LQERR(*nrqr*, *ncqr*, *qr*, *Ldqr*, *qraux*, *q*, *Ldq*)

Входные данные: *nrqr*, *ncqr*, *qr*, *Ldqr*, *qraux*, *Ldq*.

Выходные данные: *q*.

nrqr и *ncqr* – соответственно число строк и столбцов в матрице *QR*-разложения.

qr – вещественный массив формы (*Ldqr*, *ncqr*), содержащий в первых min(*nrqr*, *ncqr*) столбцах своей строго нижней трапециевидной части матрицу *Q*; *qr* возвращается подпрограммой LQRRR (DLQRRR).

Ldqr – ведущий размер массива *qr*.

qraux – вещественный вектор размера *nca*, содержащий информацию об ортогональной части *QR*-разложения в первых min(*nrqr*, *ncqr*) позициях; возвращается подпрограммой LQRRR (DLQRRR).

q – вещественный массив формы (*Ldq*, *nrqr*), содержащий ортогональную *nrqr*×*nrqr*-матрицу *Q*; массивы *q* и *qr* могут занимать одну и ту же область памяти.

Ldq – ведущий размер массива *q*.

Пример. Подпрограмма LQERR возвращает ортогональную матрицу *Q* из *QR*-разложения прямоугольной матрицы *A*. Для проверки вычисляется матрица *A*2 = *QR*.

```
program recall
 use msimsl
 integer(4), parameter :: nca = 3, nra = 4, Lda = nra, Ldqr = nra, Ldq = nra
 integer(4) :: ipvt(nca), i
 real(4) :: a(Lda, nca), conorm(nca), q(Ldqr, nra), qr(Ldqr, nca), qraux(nca), r(nra, nca), a2(nra, nca)
 logical(4) :: pivot
 a = reshape((/      1.0,    2.0,    4.0,                &
                     1.0,    4.0,   16.0,                &
```

```
                    1.0,    6.0,   36.0,                &
                    1.0,    8.0,   64.0  /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
pivot = .true.; ipvt = 0                    ! Все столбцы являются свободными
! Находим QR-разложение матрицы A
call Lqrrr(nra, nca, a, Lda, pivot, ipvt, qr, Ldqr, qraux, conorm)
call Lqerr(nra, nca, qr, Ldqr, qraux, q, Ldq)     ! Восстанавливаем матрицу Q
r= 0.0                                      ! r содержит верхнюю трапециевидную часть qr
do i =1, nra                                ! Формируем матрицу R
  r(i, i:nca) = qr(i, i:nca)
end do                                      ! Вычисляем для проверки a2 = q * r
call mrrrr(nra, nra, q, Ldq, nra, nca, r, nra, nra, nca, a2, Lda)
call wrirn('ipvt', 1, nca, ipvt, 1, 0)      ! Вывод результата
call wrrrn('Q', nra, nra, q, Ldq, 0)        ! Матрица Q
call wrrrn('A2 = QR', nra, nca, a2, Lda, 0) ! Матрица A2 отличается от исходной матрицы A
end program recall                          ! порядком следования столбцов, заданным ipvt
```

Результат:

```
        ipvt
  3       2       1

                  Q
              1        2        3        4
  1     -0.0531  -0.5422   0.8082  -0.2236
  2     -0.2126  -0.6574  -0.2694   0.6708
  3     -0.4783  -0.3458  -0.4490  -0.6708
  4     -0.8504   0.3928   0.2694   0.2236

              A2 = QR
              1        2        3
  1        4.00     2.00     1.00
  2       16.00     4.00     1.00
  3       36.00     6.00     1.00
  4       64.00     8.00     1.00
```

7.5. ПЕРЕСЧЕТ МАТРИЧНЫХ РАЗЛОЖЕНИЙ

7.5.1. Алгоритм пересчета *QR*-разложения

В некоторых задачах, например оптимизации или решения системы нелинейных уравнений, неоднократно приходится находить разложение матриц. Если, однако, $m \times n$-матрицы $\tilde{A}$ и A, для которых вычисляются разложения, связаны соотношением

$$\tilde{A} = A + xy^T,$$

где x и y – векторы соответственно размеров m и n, и для матрицы A уже известно ее разложение, например QR, то $\tilde{Q}\tilde{R}$-разложение матрицы $\tilde{A}$ целесообразнее найти, пересчитав QR-разложение матрицы A. На такой пересчет требуется $O(n^2)$ операций вместо $4n^3/3$, необходимых для определения QR-разложения матрицы $\tilde{A}$.

Замечание. Ранг xy^T равен единице, поэтому $A + xy^T$ является одноранговой модификацией матрицы A.

Проиллюстрируем порядок пересчета QR-разложения, в котором Q и R – соответственно ортогональная и верхняя треугольная матрицы. Порядок пересчета следует из равенств

$$\tilde{A} = A + xy^T = QR + xy^T = Q(R + wy^T) = QG^T(GR + Gwy^T) = \tilde{Q}\tilde{R}, \tag{7.4}$$

где $w = Q^Tx$ (для ортогональных матриц справедливо $Q^{-1} = Q^T$); G – ортогональная матрица, равная произведению матриц вращения Гивенса (разд. 7.2.2.1).

Из (7.4) видно, что необходимо применить такую последовательность преобразований Гивенса, которая приведет матрицу $R + wy^T$ к верхней треугольной матрице $\tilde{R}$.

Такое преобразование осуществляется в 2 этапа. На первом матрица $R + wy^T$ подвергается n - 1 вращению и, как результат, преобразовывается в верхнюю матрицу Хессенберга с одной нижней кодиагональю. Затем, на втором этапе, полученная матрица Хессенберга в результате новых n - 1 вращений приводится к верхнему треугольному виду.

Матрицу преобразований первого этапа обозначим буквой J, а второго – G. Тогда $A + xy^T = \tilde{Q}\tilde{R}$, где матрица $\tilde{Q} = QJ^TG^T$ является ортогональной и матрица $\tilde{R} = G(JR + Jwy^T)$ является верхней треугольной.

Первые n - 1 вращений выполняются раздельно для матриц R и wv^T. Причем они таковы, что верхняя треугольная матрица R преобразовывается в форму Хессенберга, т. е. ее форма несколько "ухудшается" по сравнению с исходной, а wv^T преобразовывается в матрицу $\|w\|_2 e_1 v^T$, где $e_1 = (1, 0, ..., 0)^T$. Чтобы обеспечить такое преобразование wv^T, а фактически w, элементы матрицы Гивенса каждого из n - 1 вращений рассчитываются по формулам

$$c = \frac{\alpha}{\sqrt{\alpha^2 + \beta^2}}; \quad s = \frac{\beta}{\sqrt{\alpha^2 + \beta^2}}, \tag{7.5}$$

в которых $\alpha = w_i$, а $\beta = w_{i+1}$, где i – номер вращения. После выполнения вращения i элемент w_{i+1} становится равным нулю. Число поворотов будет меньше, если последние k элементов исходного вектора w равны нулю (тогда вращения "стартуют" с элемента n - k - 1). В матрицах R и Q, также подвергаемых поворотам, комбинируются строки i и i + 1. Матрица Q остается

ортогональной, поскольку умножается на каждом шаге на ортогональную матрицу J^T.

Вторые n - 1 вращений производятся с целью вернуть R к верхнему треугольному виду. При этом на каждом шаге при расчетах по формулам (7.5) $\alpha = R_{ii}$, а $\beta = R_{i+1,i}$. Для пересчета Q^T применяется матрица G.

Пример. Вычисляется QR-разложение матрицы A, заданной в программе *upQR* массивом a, и выполняется пересчет разложения для матрицы $\tilde{A} = QR + xy^T$.

```
program upQR
 use msimsl
 integer(4), parameter :: nrow = 4, ncol = 3
 integer(4) :: ipvt(ncol), j, k
 real(4) :: a(nrow, ncol), conorm(ncol), q(nrow, nrow), qr(nrow, ncol), &
        qraux(ncol), r(nrow, ncol), x(nrow), y(ncol), anew(nrow, ncol)
 logical(4) :: pivot
 a = reshape((/      1.0,    2.0,    4.0,                &
                     1.0,    4.0,   16.0,                &
                     1.0,    6.0,   36.0,                &
                     1.0,    8.0,    64.0  /), shape = (/ nrow, ncol /), order = (/ 2, 1 /))
 x = (/ 1.0, 2.0, 3.0, 4.0 /)
 y = (/ 3.0, 2.0, 1.0 /)
 anew = a + matmul(reshape(x, shape = (/ nrow, 1 /)), reshape(y, shape = (/ 1, ncol /)))
 ! Разрешаем перестановку всех столбцов при разложении
 pivot = .true.; ipvt = 0
 ! QR-разложение прямоугольной матрицы A
 call Lqrrr(nrow, ncol, a, nrow, pivot, ipvt, qr, nrow, qraux, conorm)
 ! Получаем матрицу Q из разложения матрицы A
 call Lqerr(nrow, ncol, qr, nrow, qraux, q, nrow)
 call permu(ncol, y, ipvt, 1, y)          ! Перестановка y
 r = 0.0                                  ! r – верхняя трапециевидная часть QR-разложения
 do j = 1, ncol                           ! Формируем матрицу R
  k = min(j, nrow)
  r(:k, j) = qr(:k, j)
 end do
 call qrUpdate(nrow, ncol, q, r, x, y)    ! Пересчет QR-разложения
 call wrrrn('R', nrow, ncol, r, nrow, 0)  ! Вывод верхней треугольной матрицы R
 call wrrrn('Q = q*r', nrow, ncol, matmul(q, r), nrow, 0)
 ! Контрольный вывод
 call wrrrn('A + xyT', nrow, ncol, anew, nrow, 0)
end program upQR

subroutine qrUpdate(nrow, ncol, q, r, x, y) ! Пересчет QR-разложения
 integer(4):: nrow, ncol, i
 real(4) :: q(nrow, nrow), r(nrow, ncol), x(nrow), y(ncol), w(nrow)
```

```
! Получаем вектор w = Q^T x
w = matmul(transpose(q), x)
! Находим первый от конца ненулевой элемент вектора w
do n = nrow, 1, -1
 if(w(n) /= 0.0) exit
end do
! Первый этап пересчета. Приводим вектор w к виду ||w||2e1
! Матрица R будет преобразована в верхнюю форму Хессенберга
do i = n - 1, 1, -1
  ! Выполняет очередной поворот под действием матрицы Гивенса
  ! Подпрограмма GivRotate преобразовывает матрицы R и Q
  call GivRotate(nrow, ncol, i, w(i), w(i + 1), r, q)
  ! Отдельно выполняем поворот вектора w
  call wRotate(nrow, i, w)
end do
! Формируем верхнюю матрицу Хессенберга R + wy^T из преобразованных R и w
r(1, 1:ncol) = r(1, 1:ncol) + w(1) * y(1:ncol)
call wrrrn('H', nrow, ncol, r, nrow, 0)        ! Вывод верхней матрицы Хессенберга J(R + wy^T)
! Второй этап пересчета
! Преобразовываем верхнюю матрицу Хессенберга H = J(R + wy^T) в верхнюю треугольную
do i = 1, nrow - 1
  call GivRotate(nrow, ncol, i, r(i, i), r(i + 1, i), r, q)
end do

contains

 subroutine GivRotate(nrow, ncol, i, a, b, r, q)
 integer(4) :: nrow, ncol, i                   ! Поворот матриц R и Q
 real(4) :: r(nrow, ncol), q(nrow, nrow), g(nrow, nrow)
 real(4) :: a, b, c, s, d
  d = sqrt(a**2 + b**2)                        ! Строим плоское вращение Гивенса
  c = a / d; s = b / d
  g = 0.0
  ! Присваиваем диагональным элементам матрицы G единицы
  call sset(nrow, 1.0, g, nrow + 1)
  ! Выполняем плоское вращение Гивенса
  g(i, i) = c; g(i, i + 1) = s; g(i + 1, i) = -s; g(i + 1, i + 1) = c
  r = matmul(g, r)                             ! Умножаем слева матрицу R на матрицу Гивенса
  q = matmul(q, transpose(g))                  ! Умножаем справа матрицу Q
 end subroutine GivRotate                      ! на транспонированную матрицу Гивенса

 subroutine wRotate(nrow, i, w)                ! Поворот вектора w
  integer(4) :: nrow, i
  real(4) :: w(nrow), g(nrow, nrow), d, c, s
  d = sqrt(w(i)**2 + w(i + 1)**2)              ! Строим плоское вращение Гивенса
  c = w(i) / d; s = w(i + 1) / d
  g = 0.0
```

```
 ! Присваиваем диагональным элементам матрицы G единицы
 call sset(nrow, 1.0, g, nrow + 1)
 g(i, i) = c; g(i, i + 1) = s; g(i + 1, i) = -s; g(i + 1, i + 1) = c
 w = matmul(g, w)                          ! Поворот вектора w
end subroutine wRotate
end subroutine qrUpdate
```

Результат:

$H = J(R + wy^T)$ – верхняя матрица Хессенберга после первого этапа пересчета

	1	2	3
1	78.51	21.91	18.26
2	18.18	0.00	-0.73
3	0.00	0.00	-0.26
4	0.00	0.00	0.24

$\tilde{R}$ – верхняя треугольная матрица после второго пересчета

	1	2	3
1	80.59	21.34	17.62
2	0.00	4.94	4.83
3	0.00	0.00	0.36
4	0.00	0.00	0.00

Произведение пересчитанных матриц $\tilde{Q}$ и $\tilde{R}$ матриц

	1	2	3
1	5.00	4.00	4.00
2	18.00	8.00	7.00
3	39.00	12.00	10.00
4	68.00	16.00	13.00

Контрольная сумма $A + xy^T$, поскольку разрешены перестановки столбцов, отличается от $\tilde{Q}\tilde{R}$ порядком следования столбцов:

	1	2	3
1	4.00	4.00	5.00
2	7.00	8.00	18.00
3	10.00	12.00	39.00
4	13.00	16.00	68.00

Некоторые фрагменты приведенной программы можно оптимизировать по быстродействию. Так, совсем нет необходимости вызывать подпрограмму *wRotate*, осуществляющую поворот вектора w, поскольку для выполнения вычислений достаточно определить итоговое значение $w(1)$. Легко показать, что оно формируется циклом

```
do i = n - 1, 1, -1
```

```
  w(i) = sqrt(w(i)**2 + w(i + 1)**2)
end do
```

т. е. вызов *wRotate* заменяется оператором SQRT($w(i)$**2 + $w(i + 1)$**2). Извлечение корня можно сделать более безопасным, снизив вероятность переполнения и исчезновения порядка, применив вместо $w(i)$ = SQRT($w(i)$**2 + + $w(i + 1)$**2) код

```
if(w(i) == 0.0) then
  w(i) = abs(w(i + 1))
else if(abs(w(i)) > abs(w(i + 1))) then
  w(i) = abs(w(i)) * sqrt(1.0 + (w(i + 1) / w(i))**2)
else
  w(i) = abs(w(i + 1)) * sqrt(1.0 + (w(i) / w(i + 1))**2)
end if
```

Таким же образом можно модифицировать вычисление корня

```
d = sqrt(a**2 + b**2)                    ! Строим плоское вращение Гивенса
```

в подпрограмме *GivRotate*.

Замечание. Для тех же целей можно применить и функцию HYPOT библиотеки IMSL, вычисляющую корень без переполнения и исчезновения порядка:

```
w(i) = hypot(w(i), w(i + 1))
d = hypot(a, b)
```

Еще большую экономию времени вычислений и вдобавок снижение требуемой памяти даст замена произведения матриц, например

```
r = matmul(g, r)                         ! Умножаем слева матрицу R на матрицу Гивенса
```

на цикл, в котором, во-первых, не используется массив g, а, во-вторых, модифицируются только строки i и $i + 1$ матрицы R:

```
! Цикл, равнозначный произведению r = MATMUL(g, r), где g – массив, представляющий
! матрицу Гивенса
do j = i, ncol
  h1 = r(i, j); h2 = r(i + 1, j)
  r(i, j) = c * h1 + s * h2
  r(i + 1, j) = -s * h1 + c * h2
end do
```

Аналогичную замену можно предусмотреть и для произведения

```
q = matmul(q, transpose(g))              ! Умножаем справа матрицу Q на матрицу Гивенса
```

7.5.2. Перечень процедур

IMSL содержит приведенные в табл. 7.2 процедуры пересчета R^TR-разложения Холецкого и QR-разложения. Синтаксис вызова этих процедур

дан в табл. 7.3; возможные информационные ошибки – в табл. 7.4. Приводимые в таблицах имена процедур соответствуют одинарной точности.

Таблица 7.2. Подпрограммы пересчета разложений

Подпрограмма	*Назначение*
LDNCH	Получение $\tilde{R}^T\tilde{R}$-разложения Холецкого вещественной симметрической положительно определенной матрицы $\tilde{A} = A - xx^T$ по известному R^TR-разложению матрицы A
LUPCH	То же, но матрица $\tilde{A} = A + xx^T$
LUPQR	Получение $\tilde{Q}\tilde{R}$-разложения вещественной матрицы $\tilde{A} = A + \alpha xy^T$ по известному QR-разложению матрицы A

Таблица 7.3. Синтаксис вызова подпрограмм из табл. 7.2

Синтаксис вызова	*Автоматически выделяемая память*
CALL LDNCH(*n*, *r*, *Ldr*, *x*, *rnew*, *Ldrnew*, *cs*, *sn*)	-
CALL LUPCH(*n*, *r*, *Ldr*, *x*, *rnew*, *Ldrnew*, *cs*, *sn*)	-
CALL LUPQR(*nrow*, *ncol*, *alpha*, *w*, *y*, *q*, *Ldq*, *r*, *Ldr*, *ipath*, *qnew*, *Ldqnew*, *rnew*, *Ldrnew*)	*nrow* + min(*nrow* - 1, *ncol*)

Таблица 7.4. Возможная ошибка подпрограмм из табл. 7.2

Тип	*Код*	*Причина ошибки*	*Где возникает*
4	1	Матрица $R^TR - xx^T$ не является положительно определенной; пересчет R^TR-разложения невозможен	LDNCH, LUPCH

7.5.3. Пересчет разложения Холецкого

7.5.3.1. Пересчет разложения матрицы $A - xx^T$

Пересчет разложения R^TR-разложения Холецкого матрицы $A - xx^T$ выполняется подпрограммой LDNCH (DLDNCH):

CALL LDNCH(*n*, *r*, *Ldr*, *x*, *rnew*, *Ldrnew*, *cs*, *sn*)

Входные данные: *n*, *r*, *Ldr*, *x*, *Ldrnew*.

Выходные данные: *rnew*, *cs*, *sn*.

n – порядок матрицы *R*.

r – вещественный массив формы (*Ldr*, *n*), содержащий верхнюю треугольную часть обновляемого R^TR-разложения Холецкого матрицы $A = R^TR$. Адресуется только верхняя треугольная часть *r*.

Ldr – ведущий размер массива *r*.

x – вещественный вектор размера n, определяющий одноранговую матрицу, вычитаемую из матрицы A.

rnew – массив формы (*Ldrnew*, *n*), содержащий верхнюю треугольную часть пересчитанного $\tilde{R}^T\tilde{R}$-разложения матрицы $A - xx^T$. Адресуется только верхняя треугольная часть *rnew*. Если *r* после вызова LDNCH не используется, то *r* и *rnew* могут занимать одну и ту же область памяти – при вызове LDNCH имена *r* и *rnew* совпадают.

Ldrnew – ведущий размер массива *rnew*.

cs и *sn* – вещественные векторы размера n, содержащие соответственно косинусы и синусы матрицы вращения.

Описание:

Подпрограмма LDNCH основана на процедуре SCHDD пакета LINPACK [34], вычисляющей $\tilde{R}^T\tilde{R}$-разложение матрицы $A - xx^T$ по известному R^TR-разложению матрицы A. (В программе $\tilde{R}$ представляется массивом *rnew*.) Подпрограмма LDNCH находит $\tilde{R}^T\tilde{R}$-разложение лишь в том случае, когда матрица $A - xx^T$ является положительно определенной.

Подпрограмма LDNCH определяет ортогональную матрицу U, равную произведению матриц вращения Гивенса $G_n \ldots G_1$, таких, что

$$U\begin{pmatrix} R \\ 0 \end{pmatrix} = \begin{pmatrix} \tilde{R} \\ x^T \end{pmatrix}.$$

Это соотношение эквивалентно равенству $R^TR - xx^T = \tilde{R}^T\tilde{R}$. Действительно, умножим левую и правую части равенства на U^T:

$$U^TU\begin{pmatrix} R \\ 0 \end{pmatrix} = U^T\begin{pmatrix} \tilde{R} \\ x^T \end{pmatrix},$$

учитывая, что $U^TU = I$, получим

$$\begin{pmatrix} R \\ 0 \end{pmatrix} = U^T\begin{pmatrix} \tilde{R} \\ x^T \end{pmatrix} = \begin{pmatrix} R \\ 0 \end{pmatrix}.$$

Вращения Гивенса выбираются таким образом, что

$$G_1 \ldots G_n\begin{pmatrix} a \\ \alpha \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

где a – решение системы линейных уравнений $R^Ta = x$ и $\alpha = \sqrt{1 - \|a\|_2^2}$; G_i - $n+1 \times n+1$-матрица вида

$$G_i = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & c_i & 0 & -s_i \\ 0 & 0 & I_{n-i} & 0 \\ 0 & s_i & 0 & c_i \end{pmatrix},$$

где I_k – единичная $k \times k$-матрица; $c_i = \cos\theta_i = cs(i)$, $s_i = \sin\theta_i = sn(i)$ для некоторого θ_i. Далее найденные матрицы используются для вычисления $\tilde{R}$:

$$G_1 \ldots G_n \begin{pmatrix} R \\ 0 \end{pmatrix} = \begin{pmatrix} \tilde{R}^T \\ \tilde{x}^T \end{pmatrix}.$$

Матрица $\tilde{R}$ является верхней треугольной и $\tilde{x} = x$, поскольку

$$x = \begin{pmatrix} R^T & 0 \end{pmatrix} \begin{pmatrix} a \\ \alpha \end{pmatrix} = \begin{pmatrix} R^T & 0 \end{pmatrix} U^T U \begin{pmatrix} a \\ \alpha \end{pmatrix} = \begin{pmatrix} \tilde{R}^T & \tilde{x} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \tilde{x}.$$

Пример. Решается система линейных уравнений $Az = b$ с использованием R^TR-разложения Холецкого вещественной симметрической матрицы A. Затем выполняется пересчет R^TR-разложения и новое разложение берется для решения системы $(A - xx^T)z = b$.

```
program down
 use msimsl, nouse => fac
 integer(4), parameter  :: Lda=3, Ldfac=3, n=3
 real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), facnew(Ldfac, Ldfac), x(n), b(n), cs(n), sn(n), z(n)
 a = reshape((/      10.0,     3.0,     5.0,                    &
                      3.0,    14.0,    -3.0,                    &
                      5.0,    -3.0,     7.0  /), shape = (/ Lda, n /))
 x = (/ 3.0, 2.0, 1.0 /); b = (/ 53.0, 20.0, 31.0 /)
 call Lftds(n, a, Lda, fac, Ldfac)              ! Находим R^TR-разложение матрицы A
 call Lfsds(n, fac, Ldfac, b, z)                ! Решаем исходную систему линейных уравнений
 call wrrrn('fac', n, n, fac, Ldfac, 1)         ! Выводим R^TR-разложение
 call wrrrn('z', 1, n, z, 1, 0)                 ! и решение системы Ax = b
 ! Пересчет R^TR-разложения матрицы A
 call Ldnch(n, fac, Ldfac, x, facnew, Ldfac, cs, sn)
 ! Решаем новую систему линейных уравнений, используя пересчитанное разложение facnew
 call Lfsds(n, facnew, Ldfac, b, z)
 call wrrrn('facnew', n, n, facnew, Ldfac, 1) ! Вывод нового разложения
 call wrrrn('z', 1, n, z, 1, 0)                 ! и нового решения
end program down
```

Результат:

fac

	1	2	3
1	3.162	0.949	1.581
2	0	3.619	-1.243

```
3          0          0          1.719

                  z
  4.000        1.000        2.000

                 facnew
             1          2          3
1        1.000     -3.000      2.000
2            0      1.000      1.000
3            0          0      1.000

                  z
1859.9       433.0      -254.0
```

7.5.3.2. Пересчет разложения матрицы $A + xx^T$

Пересчет разложения R^TR-разложения Холецкого матрицы $A + xx^T$ выполняется подпрограммой LUPCH (DLUPCH):

CALL LUPCH(*n*, *r*, *Ldr*, *x*, *rnew*, *Ldrnew*, *cs*, *sn*)

Описание параметров, кроме *rnew*, такое же, как и для LDNCH.

rnew – массив формы (*Ldrnew*, *n*), содержащий верхнюю треугольную часть пересчитанного $\tilde{R}^T\tilde{R}$-разложения матрицы $A + xx^T$. Адресуется только верхняя треугольная часть *rnew*. Если *r* после вызова LUPCH не используется, то *r* и *rnew* могут занимать одну и ту же область памяти.

Описание:

Подпрограмма LUPCH основана на процедуре SCHUD пакета LINPACK [34], вычисляющей $\tilde{R}^T\tilde{R}$-разложение матрицы $A + xx^T$ по известному R^TR-разложению матрицы A. (В программе $\tilde{R}$ представляется массивом *rnew*.) Подпрограмма LUPCH находит $\tilde{R}^T\tilde{R}$-разложение лишь в том случае, когда матрица $A + xx^T$ является положительно определенной.

Подпрограмма LUPCH определяет ортогональную матрицу U, равную произведению матриц вращения Гивенса $G_n \ldots G_1$, таких, что

$$U\begin{pmatrix} R \\ x^T \end{pmatrix} = \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}.$$

Это соотношение эквивалентно равенству $R^TR + xx^T = \tilde{R}^T\tilde{R}$.

Каждое вращение Гивенса G_i создает равный нулю элемент в x^T; G_i является $n+1 \times n+1$-матрицей вида

$$G_i = \begin{pmatrix} I_{i-1} & 0 & 0 & 0 \\ 0 & c_i & 0 & s_i \\ 0 & 0 & I_{n-i} & 0 \\ 0 & -s_i & 0 & c_i \end{pmatrix},$$

где I_k – единичная $k\times k$-матрица; $c_i = \cos\theta_i = cs(i)$, $s_i = \sin\theta_i = sn(i)$ для некоторого θ_i.

Пример. Решается система линейных уравнений $Az = b$ с использованием R^TR-разложения Холецкого вещественной симметрической матрицы A. Затем выполняется пересчет R^TR-разложения и новое разложение берется для решения системы $(A + xx^T)z = b$.

```
program up
 use msimsl, nouse => fac
 integer(4), parameter  :: Lda=3, Ldfac=3, n=3
 real(4) :: a(Lda, Lda), fac(Ldfac, Ldfac), facnew(Ldfac, Ldfac), x(n), b(n), cs(n), sn(n), z(n)
 a = reshape((/      1.0,   -3.0,    2.0,                  &
                    -3.0,   10.0,   -5.0,                  &
                     2.0,   -5.0,    6.0  /), shape = (/ Lda, n /))
 x = (/ 3.0, 2.0, 1.0 /); b = (/ 53.0, 20.0, 31.0 /)
 call Lftds(n, a, Lda, fac, Ldfac)          ! Находим RTR-разложение матрицы A
 call Lfsds(n, fac, Ldfac, b, z)            ! Решаем исходную систему линейных уравнений
 call wrrrn('fac', n, n, fac, Ldfac, 1)     ! Выводим RTR-разложение
 call wrrrn('z', 1, n, z, 1, 0)             ! и решение системы Ax = b
 ! Пересчет RTR-разложения матрицы A
 call Lupch(n, fac, Ldfac, x, facnew, Ldfac, cs, sn)
 ! Решаем новую систему линейных уравнений, используя пересчитанное разложение facnew
 call Lfsds(n, facnew, Ldfac, b, z)
 call wrrrn('facnew', n, n, facnew, Ldfac, 1) ! Вывод нового разложения
 call wrrrn('z', 1, n, z, 1, 0)             ! и нового решения
end program up
```

Результат:

```
                    fac
              1          2          3
   1      1.000     -3.000      2.000
   2          0      1.000      1.000
   3          0          0      1.000

                    z
   1859.9         433.0       -254.0

                  facnew
              1          2          3
   1      3.162      0.949      1.581
   2          0      3.619     -1.243
   3          0          0      1.719

                    z
    4.000         1.000       2.000
```

7.5.4. Пересчет *QR*-разложения

Выполняется подпрограммой LUPQR (DLUPQR):

CALL LUPQR(*nrow*, *ncol*, *alpha*, *w*, *y*, *q*, *Ldq*, *r*, *Ldr*, *ipath*,
qnew, *Ldqnew*, *rnew*, *Ldrnew*)

Входные данные: *nrow*, *ncol*, *alpha*, *w*, *y*, *q*, *Ldq*, *r*, *Ldr*, *ipath*, *Ldqnew*, *Ldrnew*.

Выходные данные: *qnew*, *rnew*.

nrow и *ncol* – соответственно число строк и столбцов в матрице $A = QR$.

alpha – вещественный скаляр α в выражении $A + \alpha xy^T$.

w – вещественный вектор размера *nrow* в добавляемой одноранговой матрице αxy^T. Если пересчитывается разложение матрицы $A + \alpha xy^T$ и $i = 0$ (см. параметр *ipath*), то *w* содержит *x*; если $i = 1$, то *w* содержит вектор $Q^T x$.

y – вещественный вектор размера *ncol* в добавляемой ранга 1 матрице αxy^T.

q – вещественный массив формы (*Ldq*, *nrow*), содержащий *nrow*×*nrow*-матрицу *Q* из *QR*-разложения матрицы *A*. Игнорируется, если *ipath* = 0.

Ldq – ведущий размер массива *q*. Игнорируется, если *ipath* = 0.

r – массив формы (*Ldr*, *ncol*), представляющий *nrow*×*ncol*-матрицу *R* из *QR*-разложения матрицы *A*. Адресуется только верхняя трапециевидная часть *r*.

Ldr – ведущий размер массива *r*.

ipath – флаг, управляющий пересчетом *QR*-разложения; задается в виде *ij*, где $i = 0$ означает, что *w* содержит вектор *x*; $i = 1$ означает, что *w* содержит вектор $Q^T x$; $j = 0$ означает, что матрица *Q* не пересчитывается; $j = 1$ задает пересчет матрицы *Q*. Например, если *ipath* = 10, то $i = 1$ и $j = 0$.

qnew – вещественный массив формы (*Ldqnew*, *nrow*), представляющий пересчитанную *nrow*×*nrow*-матрицу *Q* из *QR*-разложения матрицы *A*. Игнорируется, если *ipath* = 0.

Ldqnew – ведущий размер массива *qnew*. Игнорируется, если *ipath* = 0.

rnew – вещественный массив формы (*Ldrnew*, *ncol*), содержащий обновленную *nrow*×*ncol*-матрицу *R* из *QR*-разложения матрицы *A*. Обновляется только верхняя трапециевидная часть *rnew*. Массивы *r* и *rnew* могут совпадать.

Ldrnew – ведущий размер массива *rnew*.

Алгоритм пересчета разложения приведен в разд. 7.5.1 и подробно изложен в [8] и в [45, p. 437-439].

Пример. Вычисляется *QR*-разложение прямоугольной матрицы *A*, которое затем используется для поиска *QR*-разложения матрицы $A + xy^T$. Поскольку применяются перестановки, то подпрограмма находит разложение $AP = QR$, где *P* – матрица перестановок, хранящаяся в векторе *ipvt*.

Порядок вычислений: $AP+\alpha xy^T=(A+\alpha x(Py)^T)P=\tilde{Q}\tilde{R}$. Процедура PERMU возвращает Py. Для проверки находится и выводится произведение $\tilde{Q}\tilde{R}$. Также оно может быть получено из матрицы $A+xy^T$ путем перестановки ее столбцов в соответствии с заданным *ipvt* порядком.

```
program upQR2
 use msimsl
 integer(4), parameter :: ncol = 3, nrow = 4, Lda = nrow, Ldaqr = nrow,                    &
                          Ldq = nrow, Ldqnew = nrow, Ldqr = nrow, Ldr = nrow, Ldrnew = nrow
 integer(4) :: ipath, ipvt(ncol), j, k
 real(4) :: a(Lda, ncol), alpha, conorm(ncol), q(Ldq, nrow),                               &
            qnew(Ldqnew, nrow), qr(Ldqr, ncol), qraux(ncol),                               &
            r(Ldr, ncol), rnew(Ldrnew, ncol), x(nrow), y(ncol)
 logical(4) :: pivot
 a = reshape((/      1.0,    2.0,    4.0,                  &
                     1.0,    4.0,   16.0,                  &
                     1.0,    6.0,   36.0,                  &
                     1.0,    8.0,   64.0  /), shape = (/ Lda, ncol /), order = (/ 2, 1 /))
 x = (/ 1.0, 2.0, 3.0, 4.0 /); y = (/ 3.0, 2.0, 1.0 /)
 ! Разрешаем перестановку всех столбцов при разложении
 pivot = .true.; ipvt = 0
 ! QR-разложение прямоугольной матрицы A
 call Lqrrr(nrow, ncol, a, Lda, pivot, ipvt, qr, Ldqr, qraux, conorm)
 ! Получаем матрицу Q из разложения матрицы A
 call Lqerr(nrow, ncol, qr, Ldqr, qraux, q, Ldq)
 call permu(ncol, y, ipvt, 1, y)              ! Перестановка y
 r = 0.0                                      ! r – верхняя трапециевидная часть QR-разложения
 do j = 1, ncol
  k = min(j, nrow); r(:k, j) = qr(:k, j)
 end do
 alpha = 1.0; ipath = 01                      ! Пересчет q и r
 call Lupqr(nrow, ncol, alpha, x, y, q, Ldq, r, Ldr, ipath, qnew, Ldqnew, rnew, Ldrnew)
 call wrim('ipvt', 1, ncol, ipvt, 1, 0)       ! Вывод результата
 call wrrm('qnew', nrow, nrow, qnew, Ldqnew, 0)
 call wrrm('rnew', nrow, ncol, rnew, Ldrnew, 0)
 call wrrm('qnew*rnew', nrow, ncol, matmul(qnew, rnew), Ldaqr, 0)
end program upQR2
```

Результат:

```
          ipvt
   3        2        1

                   qnew
                 1        2        3        4
 1        -0.0620  -0.5412   0.8082  -0.2236
```

2	-0.2234	-0.6539	-0.2694	0.6708
3	-0.4840	-0.3379	-0.4490	-0.6708
4	-0.8438	0.4067	0.2694	0.2236

rnew

	1	2	3
1	-80.59	-21.34	-17.62
2	0.00	-4.94	-4.83
3	0.00	0.00	0.36
4	0.00	0.00	0.00

qnew*rnew

	1	2	3
1	5.00	4.00	4.00
2	18.00	8.00	7.00
3	39.00	12.00	10.00
4	68.00	16.00	13.00

7.6. ВЫЧИСЛЕНИЕ СИНГУЛЯРНОГО РАЗЛОЖЕНИЯ

В разделе рассмотрены процедуры вычисления сингулярного разложения, а также его применение для вычисления $A^{\dagger}$ – обобщенной обратной матрицы в общем случае неполного ранга $m\times n$-матрицы A.

7.6.1. Понятие сингулярного разложения

Сингулярное разложение – это представление $m\times n$-матрицы A в виде

$$A = U\Sigma V^T, \tag{7.6}$$

где U и V^T – ортогональные соответственно $m\times m$- и $n\times n$-матрицы, Σ – прямоугольная диагональная $m\times n$-матрица; $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, ..., \sigma_p)$, $p = \min(m, n)$.

Диагональные элементы σ_i матрицы Σ называются *сингулярными значениями* матрицы A. Причем разложение формируется таким образом, что $\sigma_1 \geq \sigma_2 \geq ... \sigma_n \geq 0$. Столбцы матриц $U = (u^{(1)}, ..., u^{(m)})$ и $V = (v^{(1)}, ..., v^{(n)})$ называются соответственно *левыми* и *правыми сингулярными векторами* матрицы A. Причем из (7.6) следует, что $AV = \Sigma U$ и $A^TU = \Sigma^T A$, поэтому

$$\left.\begin{aligned} Av^{(i)} &= \sigma_i u^{(i)} \\ A^T u^{(i)} &= \sigma_i v^{(i)} \end{aligned}\right\} i = 1, \min(m,n) .$$

<u>*Замечание*</u>. Сингулярное разложение сокращенно обозначается аббревиатурой SVD – singular value decomposition – и называется также *SVD-разложением*.

Можно показать, что для $m \times n$-матрицы A существует сингулярное разложение, в котором диагональные элементы σ_i матрицы Σ есть неотрицательные квадратные корни из собственных значений матрицы AA^T, если $m \le n$, или матрицы A^TA, если $m \ge n$, а столбцами матриц U и V являются собственные векторы симметрических матриц AA^T и A^TA соответственно.

Количество ненулевых чисел σ_i равно рангу матрицы A. Наличие ошибок округления, а также неточность исходных данных заставляют изменить подход к определению ранга матрицы. На практике ранг определяется по количеству σ_i, больших некоторого положительного допуска.

Сингулярное разложение позволяет вычислить норму Фробениуса и 2-норму:

$$\|A\|_F = \sigma_1^2 + \sigma_2^2 + \ldots + \sigma_p^2, \quad p = \min(m, n),$$

$$\|A\|_2 = \sigma_1.$$

Отношение σ_1/σ_n ($\sigma_1 \ge \sigma_2 \ge \ldots \sigma_n > 0$) равно числу обусловленности матрицы A, вычисленному в 2-норме:

$$cond_2(A) = \|A\|_2\|A^{-1}\|_2 = \sigma_1/\sigma_n.$$

Оно имеет обычно тот же порядок, что и рассмотренное в разд. 5.2.6 число обусловленности $cond_1(A) = \|A\|_1\|A^{-1}\|_1$. Число обусловленности является мерой относительного расстояния от A до множества вырожденных матриц, поэтому A близка к вырожденной, если одно (σ_n) или несколько сингулярных значений близки к нулю. Более того, наименьшее сингулярное значение матрицы A равняется измеренному в 2-норме расстоянию от этой матрицы до множества всех матриц неполного ранга [6].

Сингулярное разложение весьма трудоемко; оно требует существенно большего числа арифметических операций, чем QR-разложение (см. табл. 7.1). Есть еще один недостаток, ограничивающий применение сингулярного разложения, – оно, в отличие, например, от QR-разложения, не поддается пересчету. С областями применения сингулярного разложения можно познакомиться в [6].

Пример. Ищется сингулярное разложение $n \times n$-матрицы A и для сравнения с результатом вычисляются собственные значения и собственные векторы матрицы AA^T.

```
program sing0
 use msimsl
 integer(4), parameter :: n = 3
 real(4) :: a(n, n), u(n, n), v(n, n), s(n), tol
 real(4) :: aat(n, n), eval(n), evec(n, n)
 a = reshape((/      8.0,    -1.0,    -5.0,                    &
                    -4.0,     4.0,    -2.0,                    &
                    18.0,    -5.0,     -7.0  /), shape = (/ n, n /), order = (/ 2, 1 /))
```

```
 tol = 10.0 * amach(4)                    ! Допуск
 ! Вычисляем сингулярные значения и матрицы U и V
 call Lsvrr(n, n, a, n, 11, tol, irank, s, u, n, v, n)
 aat = matmul(a, transpose(a))            ! Вычисляем AA^T
 call evcsf(n, aat, n, eval, evec, n)     ! Находим собственные значения и векторы
 print *, 'irank = ', irank               ! Вывод результата
 call wrrrn('s', 1, n, s, 1, 0)           ! Сингулярные значения
 eval = sqrt(eval)                        ! Ожидается, что s = SQRT(eval)
 call wrrrn(' sqrt_eval', 1, n, eval, 1, 0)  ! Корень из собственных значений матрицы AA^T
 call wrrrn('u', n, n, u, n, 0)           ! Матрица U (левые сингулярные векторы)
 call wrrrn('evec', n, n, evec, n, 0)     ! и собственные векторы матрицы AA^T
end program sing0
```

Результат:

```
irank =  3

                    s
    22.32          5.06          0.18

              sqrt_eval
    22.32          5.06          0.18

                     u
                 1             2             3
  1        -0.4137       -0.4275       -0.8038
  2         0.1740       -0.9037        0.3911
  3        -0.8936        0.0219        0.4483

                   evec
                 1             2             3
  1         0.4137        0.4275        0.8038
  2        -0.1740        0.9037       -0.3911
  3         0.8936       -0.0219       -0.4483
```

Аналогично можно обнаружить совпадение (с точностью до знака) собственных векторов матрицы A^TA и правых сингулярных векторов матрицы V.

Из приведенного примера видно, что процесс отыскания сингулярного разложения связан с алгоритмом вычисления собственных значений и собственных векторов симметрической матрицы.

7.6.2. Сингулярное разложение комплексной матрицы

Выполняет подпрограмма LSVCR (DLSVCR):

CALL LSVCR(*nra*, *nca*, *a*, *Lda*, *ipath*, *tol*, *irank*, *s*, *u*, *Ldu*, *v*, *Ldv*)

Входные данные: *nra*, *nca*, *a*, *Lda*, *ipath*, *tol*, *Ldu*, *Ldv*.

Выходные данные: *irank*, *s*, *u*, *v*.

nra и *nca* – соответственно число строк и столбцов в матрице *A*.

a – комплексный массив формы (*Lda*, *nca*), представляющий комплексную *nra*×*nca*-матрицу *A*.

Lda – ведущий размер массива *a*.

ipath – целочисленный флаг, управляющий вычислением сингулярных векторов; имеет вид *ij*, где:

i = 0 означает, что не вычисляются левые сингулярные векторы;

i = 1 означает, что в *u* возвращаются *nca* левых сингулярных векторов;

i = 2 означает, что в *u* возвращаются min(*nra*, *nca*) левых сингулярных векторов;

j = 0 означает, что не вычисляются правые сингулярные векторы;

j = 1 означает, что в *v* возвращаются правые сингулярные векторы.

Например, задание *ipath* = 20 означает, что *i* = 2 и *j* = 0;

tol – вещественный скаляр, содержащий допуск, используемый для определения ситуации, когда сингулярное значение пренебрежительно мало. Если *tol* > 0, то сингулярное значение σ_i считается пренебрежительно малым, когда $\sigma_i \le tol$; если *tol* < 0, то сингулярное значение σ_i считается пренебрежительно малым, когда $\sigma_i \le |tol|*(\|A\|_\infty)$. В последнем случае $|tol|$ должен содержать оценку относительной ошибки входных данных.

irank – целочисленный скаляр, содержащий оценку ранга матрицы *A*.

s – комплексный вектор размера min(*nra* + 1, *nca*), содержащий в первых min(*nra*, *nca*) позициях сингулярные значения матрицы *A*, упорядоченные по убыванию их величин.

u – комплексный массив формы (*Ldu*, *nra*), если *i* = 1, или формы (*Ldu*, min(*nra*, *nca*)), если *i* = 2, содержащий левые сингулярные векторы матрицы *A*. Массив *u* не адресуется, если *i* = 0. Если *nra* ≤ *nca* или *ipath* = 2, то *u* и *a* могут быть одним и тем же массивом.

Ldu – ведущий размер массива *u*.

v – комплексный массив формы (*Ldv*, *nca*), содержащий правые сингулярные векторы матрицы *A*. Массив *v* не адресуется, если *j* = 0. Если *nca* ≤ *nra*, то *v* и *a* могут быть одним и тем же массивом, но *u* и *v* не могут совпадать с *a* одновременно.

Ldv – ведущий размер массива *v*.

Возможная информационная ошибка, имеющая тип 4 и код 1, означает что сходимость не может быть достигнута для всех сингулярных значений и им отвечающих сингулярных векторов.

Когда *nra* >> *nca*, нецелесообразно хранить весь массив *u*. В этом случае значение *ipath* с *i* = 2 позволяет, вычисляя сингулярное разложение *A*, находить только первые *nca* столбцов *u*, что вполне достаточно для многих приложений.

Целочисленные рабочие параметры LSVCR можно регулировать подпрограммой IUMAG (разд. 3.6). Так, задание в массиве *iopts* подпрограммы IUMAG целочисленного значения 16 используется для предотвращения конфликтов памяти: в подпрограмме LSVCR, когда *n* кратно *ival*(4), выделяется дополнительная память под разложение. По умолчанию *ival*(1:4) = (/ 1, 16, 0, 1 /).

Целочисленное значение 17, заданное в массиве *iopts*, позволяет включать или отключать режим расчета оценки числа обусловленности. Число обусловленности вычисляется, если *ival*(2) = 2, и не вычисляется в противном случае. По умолчанию *ival*(1:2) = (/ 1, 2 /).

Описание:

Подпрограмма LSVCR основана на процедуре CSVDC пакета LINPACK, [34].

Пусть $n = nra$ и $p = nca$. Для любой $n \times p$-матрицы A существуют ортогональные $n \times n$-матрица U и $p \times p$-матрица V, такие, что

$$U^T A V = \begin{cases} \begin{pmatrix} \Sigma \\ 0 \end{pmatrix}, & n \ge p, \\ \begin{pmatrix} \Sigma & 0 \end{pmatrix}, & n \le p, \end{cases}$$

где $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_m)$ и $m = \min(n, p)$.

Оценкой ранга матрицы A является количество значений σ_k, больших допуска η. Причем

$$\eta = \begin{cases} \tau, & \tau > 0, \\ |\tau| \|A\|_\infty, & \tau < 0, \end{cases}$$

где τ – задаваемый в программе параметр *tol*.

Пример. Вычисляется сингулярное разложение 6×3-матрицы A. Выводятся массивы u и v, содержащие левые и правые сингулярные векторы, и вектор s с сингулярными значениями.

```
program sing
 use msimsl
 integer(4), parameter :: nra = 6, nca = 3, Lda = nra, Ldu = nra, Ldv = nca
 complex(4) :: a(Lda, nca), u(Ldu, nra), v(Ldv, nca), s(nca)
 a = reshape((/     (1.0, 2.0),  (3.0, 2.0),  (1.0, -4.0),                &
                   (3.0, -2.0),  (2.0, -4.0),  (1.0, 3.0),                &
                    (4.0, 3.0),  (-2.0, 1.0),  (1.0, 4.0),                &
                   (2.0, -1.0),   (3.0, 0.0),  (3.0, -1.0),               &
                   (1.0, -5.0),  (2.0, -5.0),   (2.0, 2.0),               &
                    (1.0, 2.0),  (4.0, -2.0),   (2.0, -3.0) /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
 ipath = 11                                  ! Вычисляем все сингулярные векторы
 tol = 10.0 * amach(4)                       ! Допуск
 call Lsvcr(nra, nca, a, Lda, ipath, tol, irank, s, u, Ldu, v, Ldv)
 print *, 'irank = ', irank                  ! Вывод результата
```

```
 call wrcrn('u', nra, nra, u, Ldu, 0); call wrcrn('s', 1, nca, s, 1, 0); call wrcrn('v', nca, nca, v, Ldv, 0)
end program sing
```

Результат:

```
irank =   3

                                    u
                1                   2                   3                   4
1   ( 0.1968, 0.2186)   ( 0.5011, 0.0217)   (-0.2007,-0.1003)   (-0.2036, 0.0405)
2   ( 0.3443,-0.3542)   (-0.2933, 0.0248)   ( 0.1155,-0.2338)   (-0.2316, 0.0287)
3   ( 0.1457, 0.2307)   (-0.5424, 0.1381)   (-0.4361,-0.4407)   ( 0.0281,-0.3088)
4   ( 0.3016,-0.0844)   ( 0.2157, 0.2659)   (-0.0523,-0.0894)   ( 0.8617, 0.0223)
5   ( 0.2283,-0.6008)   (-0.1325, 0.1433)   ( 0.3152,-0.0090)   (-0.0392,-0.0145)
6   ( 0.2876,-0.0350)   ( 0.4377,-0.0400)   ( 0.0458,-0.6205)   (-0.2303, 0.0924)

                5                   6
1   ( 0.4132,-0.0985)   (-0.6017, 0.1612)
2   (-0.5061, 0.0198)   (-0.5380,-0.0317)
3   ( 0.2043,-0.1853)   ( 0.1012, 0.2132)
4   (-0.1272,-0.0866)   (-0.0808,-0.0266)
5   ( 0.6482,-0.1033)   ( 0.0995,-0.0837)
6   (-0.1412, 0.1121)   ( 0.4897,-0.0436)

                    s
( 11.77,  0.00)     ( 9.30,  0.00)     ( 4.99,  0.00)

                    v
                1                   2                   3
1   ( 0.6616, 0.0000)   (-0.2651, 0.0000)   (-0.7014, 0.0000)
2   ( 0.7355, 0.0379)   ( 0.3850,-0.0707)   ( 0.5482, 0.0624)
3   ( 0.0507,-0.1317)   ( 0.1724, 0.8642)   (-0.0173,-0.4509)
```

7.6.3. Сингулярное разложение вещественной матрицы

Выполняет подпрограмма LSVRR (LSVRR):

CALL LSVRR(*nra*, *nca*, *a*, *Lda*, *ipath*, *tol*, *irank*, *s*, *u*, *Ldu*, *v*, *Ldv*)

Описание параметров такое же, как и для LSVCR (см. разд. 7.6.2), за тем исключением, что массивы *a*, *u*, *v* и *s* являются вещественными.

Подпрограмма LSVRR основана на процедуре SSVDC пакета LINPACK, [34]. Описания информационной ошибки и возможных настроек подпрограммы и алгоритма такие же, как и для LSVCR.

Пример. Вычисляется сингулярное разложение 6×4-матрицы *A*. Выводятся массивы *u* и *v*, содержащие левые и правые сингулярные векторы, и вектор *s* с сингулярными значениями.

```
program sing2
 use msimsl
```

```
integer(4), parameter :: nra = 6, nca = 4, Lda = nra, Ldu = nra, Ldv = nca
integer(4) :: i
real(4) :: a(Lda, nca), u(Ldu, nra), v(Ldv, nca), s(nca)
a = reshape((/      1.0,    2.0,    1.0,    4.0,                &
                    3.0,    2.0,    1.0,    3.0,                &
                    4.0,    3.0,    1.0,    4.0,                &
                    2.0,    1.0,    3.0,    1.0,                &
                    1.0,    5.0,    2.0,    2.0,                &
                    1.0,    2.0,    2.0,     3.0  /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
ipath = 11                                  ! Вычисляем все сингулярные векторы
tol = 10.0 * amach(4)                       ! Допуск
call Lsvrr(nra, nca, a, Lda, ipath, tol, irank, s, u, Ldu, v, Ldv)
print *, 'irank = ', irank                  ! Вывод результата
call wrrrn('u', nra, nra, u, Ldu, 0); call wrrrn('s', 1, nca, s, 1, 0); call wrrrn('v', nca, nca, v, Ldv, 0)
! Проверка разложения
! Исходную матрицу A вернет произведение матриц, входящих в разложение
a = 0.0; do i = 1, nca; a(i, i) = s(i); end do  ! Формируем диагональную матрицу
a = matmul(u, matmul(a, transpose(v)));
call wrrrn('a', Lda, nca, a, Lda, 0)
end program sing2
```

Результат:

irank = 4

u

	1	2	3	4	5	6
1	-0.3805	0.1197	0.4391	-0.5654	0.0243	-0.5726
2	-0.4038	0.3451	-0.0566	0.2148	0.8089	0.1193
3	-0.5451	0.4293	0.0514	0.4321	-0.5723	0.0403
4	-0.2648	-0.0683	-0.8839	-0.2153	-0.0625	-0.3062
5	-0.4463	-0.8168	0.1419	0.3213	0.0621	-0.0799
6	-0.3546	-0.1021	-0.0043	-0.5458	-0.0988	0.7457

s

11.49 3.27 2.65 2.09

v

	1	2	3	4
1	-0.4443	0.5555	-0.4354	0.5518
2	-0.5581	-0.6543	0.2775	0.4283
3	-0.3244	-0.3514	-0.7321	-0.4851
4	-0.6212	0.3739	0.4444	-0.5261

7.6.4. Вычисление обобщенной обратной матрицы

Выполняется подпрограммой LSGRR (DLSGRR):

CALL LSGRR(*nra*, *nca*, *a*, *Lda*, *tol*, *irank*, *ginva*, *Ldginv*)

Входные данные: *nra*, *nca*, *a*, *Lda*, *tol*, *Ldginv*.

Выходные данные: *irank*, *ginva*.

nra и *nca* – соответственно число строк и столбцов в матрице A.

a – вещественный массив формы (*Lda*, *nca*), содержащий *nra*×*nca*-матрицу A.

Lda – ведущий размер массива *a* по первому измерению.

tol – вещественный скаляр, содержащий допуск, используемый для определения ситуации, когда сингулярное значение матрицы A пренебрежительно мало. Если $tol > 0$, то сингулярное значение σ_i считается пренебрежительно малым, если $\sigma_i \le tol$. Если $tol < 0$, то сингулярное значение σ_i считается пренебрежительно малым, если $\sigma_i \le |tol|*\|A\|_\infty$. В этом случае $|tol|$ должен содержать оценку относительной ошибки в данных.

irank – целочисленный скаляр, содержащий оценку ранга матрицы A.

ginva – вещественный массив формы (*Ldginv*, *nra*), содержащий обобщенную обратную матрицу $A^\dagger$.

Ldginv – ведущий размер массива *ginva*.

Если сходимость не может быть достигнута для всех сингулярных значений и им отвечающих сингулярных векторов, то генерируется *информационная ошибка* типа 4 с кодом 1.

Описание:

Пусть $k = irank$, $n = nra$, $p = nca$ и $A^\dagger = ginva$. Для вычисления Мура – Пенроуза обобщенной обратной матрицы A подпрограмма LSVRR прежде выполняет сингулярное разложение A, состоящее из ортогональных n×n-матрицы U и p×p-матрицы V и диагональной матрицы $\Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_m)$, $m = \min(n, p)$, таких, что $U^TAV = (\Sigma, 0)$, если $n \le p$, и $U^TAV = (\Sigma, 0)^T$, если $n > p$. Определяются только первые p столбцов U. Ранг k матрицы A оценивается путем вычисления числа пренебрежительно малых значений σ_i.

Матрицы U и V могут быть представлены как $U = (U_1, U_2)$ и $V = (V_1, V_2)$, где U_1 и V_1 – k×k-матрицы.

Пусть $\Sigma_1 = \mathrm{diag}(\sigma_1, ..., \sigma_k)$. Тогда Мура – Пенроуза обобщенное обращение матрицы A – это матрица

$$A^\dagger = V_1\Sigma_1^{-1}U_1^T.$$

Пример. Вычисляется и выводится $A^\dagger$ – обобщенная обратная матрица заданной 3×2-матрицы A. Также выводится ранг $k = irank$ найденной матрицы $A^\dagger$.

```
program gin
 use msimsl
 integer(4), parameter :: nra = 3, nca = 2, Lda = nra, Ldginv = nca
 real(4) :: a(Lda, nca), ginv(Ldginv, nra), tol
```

```
a = reshape((/        1.0,      0.0,          &
                      1.0,      1.0,          &
                    100.0,     50.0  /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
tol = 10.0 * amach(4)                    ! Вычисляем обобщенную обратную матрицу
call Lsgrr(nra, nca, a, Lda, tol, irank, ginv, Ldginv)
print *, 'irank = ', irank               ! Вывод результатов
call wrrrn('ginv', nca, nra, ginv, Ldginv, 0)
end program gin
```

Результат:

```
irank =   2

                ginv
                  1          2          3
    1         0.500     -0.500      0.010
    2        -1.000      1.000      0.000
```

8. СОБСТВЕННЫЕ ЗНАЧЕНИЯ И СОБСТВЕННЫЕ ВЕКТОРЫ

8.1. ПОДХОДЫ К РЕШЕНИЮ ПРОБЛЕМЫ СОБСТВЕННЫХ ЗНАЧЕНИЙ

8.1.1. Понятия собственного значения и собственного вектора

Пусть дана $n \times n$-матрица A. Число λ называется *собственным значением* матрицы A, если существует ненулевой вектор x, удовлетворяющий уравнению

$$Ax = \lambda x.$$

Такой вектор x называется *собственным вектором* матрицы A, соответствующим собственному значению λ.

Собственные значения и собственные векторы являются важными характеристиками линейных моделей. В частности, в электрических и механических системах собственные значения отвечают собственным частотам колебаний, а собственные векторы характеризуют их моды.

Запишем уравнение $Ax = \lambda x$ в виде

$$(A - \lambda I)x = 0, \tag{8.1}$$

где матрица $A - \lambda I$ называется *характеристической матрицей*. Система (8.1), поскольку свободные члены всех ее уравнений равны нулю, является однородной системой линейных уравнений. Такая система имеет ненулевые решения, когда ее определитель равен нулю, т. е.

$$\det(A - \lambda I) = 0. \tag{8.2}$$

Уравнение (8.2) называется *характеристическим уравнением* матрицы A, а определитель $\det(A - \lambda I)$ – *характеристическим определителем*. В развернутом виде уравнение (8.2) запишется так:

$$\begin{vmatrix} a_{11}-\lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22}-\lambda & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn}-\lambda \end{vmatrix} = 0 \tag{8.3}$$

или

$$\lambda^n + p_1\lambda^{n-1} + p_2\lambda^{n-2} + \dots + p_{n-1}\lambda + p_n = 0. \tag{8.4}$$

Таким образом, нахождение собственных значений матрицы A можно свести к вычислению $p_1, p_2, \dots, p_n$ и последующему решению характеристического уравнения (8.4), которое в общем случае имеет n корней. Причем если матрица A является вещественной симметрической или комплексной

эрмитовой, то корни многочлена (8.4) являются вещественными; в противном случае они комплексные. Также справедливо следующее: если собственные значения матрицы различны, то каждому собственному значению соответствует с точностью до коэффициента пропорциональности ровно один собственный вектор.

Пример. Найти собственные значения и собственные векторы вещественной симметрической матрицы, [7, с. 369].

$$A=\begin{pmatrix}2 & 1 & 1\\ 1 & 2 & 1\\ 1 & 1 & 2\end{pmatrix}.$$

Составим ее характеристическое уравнение:

$$\begin{vmatrix}2-\lambda & 1 & 1\\ 1 & 2-\lambda & 1\\ 1 & 1 & 2-\lambda\end{vmatrix}=0 \quad \text{или} \quad (\lambda-1)^2(4-\lambda)=0,$$

т. е. собственные значения матрицы равны

$$\lambda_1=\lambda_2=1;\ \lambda_3=4.$$

Найдем теперь собственный вектор, отвечающий λ1 = 1. Подставим λ1 = 1 в (8.1):

$$\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix}=0 \quad \text{или} \quad \begin{aligned}x_1+x_2+x_3&=0,\\ x_1+x_2+x_3&=0,\\ x_1+x_2+x_3&=0.\end{aligned}$$

Полученная однородная система имеет бесконечное число решений, которые находятся из уравнения $x_1 + x_2 + x_3 + 0$. Положив $x_1 = c_1$ и $x_2 = c_2$, найдем $x_3 = -(c_1 + c_2)$, где c_1 и c_2 – любые не равные нулю одновременно вещественные числа.

Выбирая теперь $c_1 = 1$ и $c_2 = 0$, имеем собственный вектор

$$x^{(1)}=(1,\ 0,\ -1)^T,$$

отвечающий собственному значению $\lambda_1 = 1$.

Выбрав для собственного значения $\lambda_2 = 1$, $c_1 = 0$ и $c_2 = 1$, найдем отвечающий ему собственный вектор

$$x^{(2)}=(0,\ 1,\ -1)^T.$$

Все остальные собственные векторы матрицы A, соответствующие собственным значениям $\lambda_1 = \lambda_2 = 1$, являются линейной комбинацией векторов $x^{(1)}$ и $x^{(2)}$.

Собственный вектор, отвечающий собственному значению $\lambda_3 = 4$, найдем, подставив λ_3 в (8.1). После чего имеем систему

$$\begin{aligned} -2x_1 &+ x_2 + x_3 = 0, \\ x_1 &- 2x_2 + x_3 = 0, \\ x_1 &+ x_2 - 2x_3 = 0. \end{aligned}$$

Рассматривая ее третье уравнение как следствие двух первых, получим систему

$$\begin{aligned} -2x_1 &+ x_2 + x_3 = 0, \\ x_1 &- 2x_2 + x_3 = 0. \end{aligned}$$

Нетрудно показать, что ее решения связаны равенством $x_1 = x_2 = x_3 = c$, где c – отличное от нуля вещественное число. Положив $c = 1$, найдем простейший соответствующий $\lambda_3 = 4$ собственный вектор

$$x^{(3)} = (0,\ 1,\ -1)^T.$$

Определение собственных значений и собственных векторов, удовлетворяющих системе (8.1), иногда называют *обычной проблемой собственных значений.*

Поиск всех собственных значений и собственных векторов, как это было выполнено в вышеприведенном примере, называют *полной проблемой собственных значений.*

На практике, однако, часто требуется найти не все, а лишь некоторые собственные значения, например при изучении устойчивости процессов достаточно указать границы, в которых лежат собственные значения, т. е. найти их максимальные и минимальные величины; при изучении резонанса требуется найти собственные значения, близкие к известному числу. Поэтому, поскольку в таких ситуациях решение полной проблемы собственных значений нецелесообразно, решают *частичную проблему собственных значений*, подразумевающую определение одного или небольшого числа собственных значений и собственных векторов. При этом для решения полной и частичной проблем собственных значений используются разные методы.

Помимо обычной выделяют также и *обобщенную проблему собственных значений*, состоящую в вычислении собственных значений λ и собственных векторов z системы

$$Az = \lambda Bz,$$

в которой A и B – это $n \times n$-матрицы.

Собственные значения и собственные векторы можно найти, решив характеристический многочлен (8.4). Такой подход преобладал до 40-х гг. XX в. Основные усилия при его реализации были направлены на разработку методов поиска коэффициентов характеристического многочлена (непосредственное определение его коэффициентов требует вычисления $2^n - 1$ определителей различных порядков). Поэтому были созданы специальные методы развертывания характеристических определителей, (Крылова А. Н., Данилевско-

го А. М., неопределенных коэффициентов и др.), получившие название *прямых методов*. Однако такой подход оказывается неудовлетворительным, когда определяются собственные значения матриц высокого порядка. Одна из причин состоит в том, что задачи (8.1) и (8.4) имеют разную обусловленность. Причем хуже обусловлена задача (8.4), так как нередко корни многочлена (8.4) высокой степени крайне чувствительны к погрешностям его коэффициентов, которые неизбежно возникают при их вычислении – развертывании определителя (8.3). Следовательно, точность решения проблемы собственных значений может оказаться неудовлетворительной.

К началу 60-х гг. прямые методы были практически полностью вытеснены *итерационными*, не использующими вычисления коэффициентов характеристического уравнения, но интенсивно применяющими преобразования подобия.

8.1.2. Преобразования подобия

Матрицы A и B *подобны*, если существует такая невырожденная матрица P, что справедливо

$$B = P^{-1}AP. \tag{8.5}$$

Относительно матриц A и B, связанных соотношением (8.5), говорят, что матрица B получается из матрицы A путем *преобразования подобия* с помощью матрицы P.

Подобные матрицы в проблеме собственных значений интересны тем, что их собственные значения совпадают. Причем собственные векторы x и y подобных матриц A и B связаны соотношением $x = Py$. Тогда, используя преобразования подобия, можно привести исходную матрицу к более простому виду, поиск собственных значений которой сравнительно прост. Так, если привести матрицу к треугольному виду

$$U = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1n} \\ 0 & a'_{22} & \dots & a'_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a'_{nn} \end{pmatrix},$$

то ее характеристическое уравнение (8.3) также будет иметь треугольный вид, и после его раскрытия, учитывая, что определитель треугольной матрицы равен произведению ее диагональных элементов, получим

$$(a'_{11} - \lambda_1)(a'_{22} - \lambda_2)\dots(a'_{nn} - \lambda_n) = 0,$$

т. е. собственные значения треугольной матрицы равны ее диагональным элементам.

Существуют также матрицы, называемые *матрицами простой структуры*, которые можно при помощи преобразования подобия привести к диагональному виду:

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}.$$

К ним относятся квадратные матрицы, собственные значения которых попарно различны, а также вещественные симметрические матрицы. Причем в случае последних матрица подобия P может быть выбрана ортогональной, т. е. удовлетворяющей условию $P^{-1} = P^T$.

Очевидно, что собственные значения диагональной матрицы равны ее диагональным элементам.

Приведение матриц к треугольному или диагональному виду методами подобия – весьма трудоемкое занятие. Поэтому, если решается частичная проблема собственных значений, часто применяются методы, например *степенной*, в которых преобразования подобия не используются.

8.1.3. Некоторые свойства собственных значений и собственных векторов

Свойства приводятся без доказательств:

- каждая $n \times n$-матрица имеет n собственных значений (каждое собственное значение считается столько раз, какова его кратность);
- сумма всех собственных значений матрицы равна корню из суммы ее диагональных элементов (т. е. ее следу);
- собственные значения вещественной симметрической и комплексной эрмитовой матриц являются вещественными;
- модули собственных значений симметрической матрицы совпадают с ее соответствующими сингулярными значениями;
- все собственные значения ортогональных и унитарных матриц по модулю равны единице;
- произведение собственного вектора матрицы на скаляр является так же ее собственным вектором, соответствующим тому же собственному значению. Это свойство позволяет использовать нормированные собственные векторы (что обычно и делают), получаемые в результате деления каждого элемента вектора либо на его наибольший элемент, либо на сумму квадратов всех его элементов; в последнем случае евклидова длина каждого собственного вектора будет равна единице;
- если матрицы A и B подобны, то их собственные значения совпадают;

- собственные векторы, соответствующие попарно различным собственным значениям, линейно независимы;
- если собственные значения $n \times n$-матрицы попарно различны, то ее собственные векторы ортогональны (скалярное произведение любой пары собственных векторов равно нулю); иными словами, при таких условиях собственные векторы образуют ортонормированный базис соответствующего n-мерного пространства;
- матрица является невырожденной тогда и только тогда, когда все ее собственные значения отличны от нуля;
- собственные значения и собственные векторы позволяют представить матрицу A в виде

  $$A = VDV^{-1},$$

  где D – диагональная матрица, составленная из собственных значений A; V – ортогональная матрица ($V^T = V^{-1}$), содержащая собственные векторы A.

8.1.4. Собственные значения и собственные векторы вещественной симметрической и комплексной эрмитовой матрицы

Процедуры IMSL при определении собственных значений вещественной симметрической или комплексной эрмитовой матрицы первоначально посредством преобразований подобия приводят ее к трехдиагональному виду.

В случае матрицы общего вида она приводится преобразованиями подобия к форме Хессенберга. В нижней матрице Хессенберга равны нулю все элементы, лежащие выше некоторой верхней кодиагонали; в верхней – лежащие ниже некоторой нижней кодиагонали:

$$H = \begin{pmatrix} h_{11} & h_{12} & \dots & h_{1,n-1} & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2,n-1} & h_{2n} \\ 0 & h_{32} & \dots & h_{3,n-1} & h_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & h_{n,n-1} & h_{nn} \end{pmatrix}.$$

Одним из методов, выполняющих такие преобразования, является метод Хаусхолдера.

Затем применяется QR-алгоритм, неоднократно производящий QR-разложение матрицы на произведение ортогональной (Q) и верхней треугольной (R) матриц. В результате исходная трехдиагональная матрица преобразовывается в подобную ей диагональную, а матрица Хессенберга – в верхнюю треугольную. Собственными значениями таких матриц являются их диагональные элементы.

Рассмотрим более детально метод вычисления собственных значения и собственных векторов симметрической матрицы. Он предусматривает:

1) приведение симметрической матрицы к трехдиагональному виду посредством преобразований подобия. Это означает, что собственные значения исходной и трехдиагональной матрицы совпадают;
2) приведение полученной трехдиагональной матрицы к диагональной. Оно также выполняется в результате преобразований подобия, т. е. с сохранением собственных значений исходной и диагональной матриц. Инструментом таких преобразований является QR-разложение матрицы. Искомые собственные значения равны диагональным элементам полученной диагональной матрицы;
3) вычисление для каждого собственного значения отвечающего ему собственного вектора.

Способы преобразования симметрической матрицы в трехдиагональную и вычисления ее QR-разложения рассмотрены в гл. 7. В последующих подразделах настоящего раздела приводятся алгоритмы, реализующие второй и третий этапы приведенного метода.

8.1.4.1. QR-алгоритм

QR-алгоритм позволяет привести матрицу A общего вида к верхней квазитреугольной, а вещественную симметрическую или комплексную эрмитову – к квазидиагональной форме. (В общем случае в верхней квазитреугольной матрице элементы, расположенные ниже диагонали, близки к нулю, но не равны ему точно. В квазидиагональной матрице это утверждение справедливо для всех не лежащих на ее диагонали элементов.) Исходная и результирующая матрицы подобны, поэтому их собственные значения совпадают. Алгоритм выполняется в 2 этапа.

На первом исходная матрица общего вида приводится к форме Хессенберга, а симметрическая – к трехдиагональной. Затем выполняется цикл

```
Пока алгоритм не сошелся и число итераций меньше допустимого числа, выполнять
  A = QR                          ! A – матрица, переданная с этапа 1
  A = RQ                          ! Исходная, промежуточная и результирующая
конец цикла                       ! матрицы подобны
```

Рассмотрим суть итераций. В первой вычисляют QR-разложение матрицы

$$A_0 = A = Q_1 R_1. \tag{8.6}$$

Затем строят матрицу $A_1 = R_1 Q_1$, которая подобна матрице A_0. В самом деле, из равенства (8.6) следует, что $R_1 = Q_1^{-1} A_0$, поэтому $A_1 = Q_1^{-1} A_0 Q_1$.

На второй итерации находят QR-разложение матрицы $A_1 = Q_2 R_2$ и вычисляют матрицу $A_2 = R_2 Q_2$. Она подобна A_1 и, следовательно, подобна матрице $A_0 = A$.

На итерации $k + 1$ находят матрицы $A_k = Q_{k+1} R_{k+1}$ и $A_{k+1} = R_{k+1} Q_{k+1}$.

Последовательность A_n сходится к верхней треугольной матрице, если исходной является матрица общего вида, или к диагональной, если исходная

матрица симметрическая. В качестве собственных значений матрицы A_n, а значит и ей подобной матрицы A, можно с известной степенью точности принять диагональные элементы A_n.

Пример. Применяется QR-алгоритм (реализован в подпрограмме *diag*) для нахождения всех собственных значений матрицы. Предварительно матрица преобразовывается подпрограммой *hous* в трехдиагональный вид. Поиск QR-разложения выполняется подпрограммой *qrfac*, подпрограмма *qrest2* применяется для восстановления матрицы Q очередного QR-разложения.

```
program qral
 use msimsl                                  ! Для вызова WRRRN
 integer(4), parameter :: n = 6
 real(4) :: a(n, n), eval(n)
 a = reshape((/        5.0,    -4.0,    -10.0,    1.0,    1.0,    -2.0,      &
                      -4.0,     6.0,     -4.0,    1.0,    1.0,     1.0,      &
                     -10.0,    -4.0,      6.0,   -4.0,    1.0,     1.0,      &
                       1.0,     1.0,     -4.0,    6.0,   -4.0,    -3.0,      &
                       1.0,     1.0,      1.0,   -4.0,    6.0,    -4.0,      &
                      -2.0,     1.0,      1.0,   -3.0,   -4.0,     5.0 /),   &
                 shape = (/ n, n /), order = (/ 2, 1 /))
 ! Приводим исходную симметрическую матрицу к трехдиагональному виду
 call hous(n, a)                             ! Текст hous приведен в разд. 7.3.2.1
 call diag(n, a, eval)                       ! Находим собственные значения
 call wrrrn('A', n, n, a, n, 0)              ! Выведем квазидиагональную матрицу A
 call wrrrn('eval', 1, n, eval, 1, 0)        ! Выводим собственные значения
end program qral

! Возвращает собственные значения – диагональ квазидиагональной матрицы
subroutine diag(n, a, eval)
 use text_transfer                           ! Для вывода русских сообщений в DOS-окне
 integer(4), parameter :: nmax = 100
 integer(4) :: j, k
 real(4) :: a(n, n), g(2, n), q(n, n), eval(n)
 real(4) :: eps = 1.0e-3
 logical(4) :: fl
 do k = 1, nmax                              ! Текст qrfac приведен в разд. 7.3.2.2.
  call qrfac(n, a, g)                        ! QR-разложение матрицы A
  ! Верхняя треугольная матрица R возвращается на место матрицы A
  ! Восстанавливаем ортогональную матрицу Q = G1...Gt
  call qrest2(n, g, q)                       ! Текст qrest2 см. в разд. 7.2.2.3
  a = matmul(a, q)                           ! Находим Ak+1 = Rk+1Qk+1
  fl = .true.                                ! Флаг останется равным .TRUE.,
  do j = 1, n - 1                            ! если матрица A – квазидиагональная
   fl = fl .and. abs(a(j + 1, j)) < eps
   if(.not. fl) exit
  end do
```

```
  if(fl) exit                                  ! Процесс сошелся
 end do
 print *, trim(ru_doswin("Число итераций k = ", .false.)), k
 do k = 1, n; eval(k) = a(k, k); end do        ! Формируем результат
end subroutine diag
```

Результат:

Число итераций i = 95

A

	1	2	3	4	5	6
1	17.49	0.00	0.00	0.00	0.00	0.00
2	0.00	10.75	0.00	0.00	0.00	0.00
3	0.00	0.00	8.40	0.00	0.00	0.00
4	0.00	0.00	0.00	-7.28	0.00	0.00
5	0.00	0.00	0.00	0.00	6.68	0.00
6	0.00	0.00	0.00	0.00	0.00	-2.05

eval

1	2	3	4	5	6
17.49	10.75	8.40	-7.28	6.68	-2.05

Замечание. Для работы программы *qra,l* кроме разобранной в этой главе подпрограммы *diag*, требуются подпрограммы *col_rot*, *cs*, *hous*, *hvec*, *qrest2*, *qrfac* и *row_rot2*, текст которых приведен в гл. 7.

Рассмотрим матрицу A после, например, 30-го шага алгоритма

$$A_{30} = \begin{pmatrix} 17.49 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10.75 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3.38 & 7.32 & 0 & 0 \\ 0 & 0 & 7.32 & -2.25 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 & 6.68 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2.05 \end{pmatrix}.$$

Ее начальные и конечные поддиагональные (наддиагональные) элементы близки к нулю, а диагональные содержат хорошие приближения собственных значений. Поэтому нет нужды далее преобразовывать всю матрицу, а есть смысл заняться ее центральной частью. Такое разбиение задачи называется *расщеплением*. А реализующий идею расщепления алгоритм – *QR – алгоритмом со сдвигом*. На практике расщепление осуществляется всякий раз, когда поддиагональный элемент достаточно мал. Например, в процедурах библиотеки Eispack, если

$$|a_{i+1,i}| \le c\varepsilon_{\text{м}}\left(|a_{ii}| + |a_{i+1,i+1}|\right)$$

при малой константе c, $a_{i+1,i}$ считают равным нулю.

8.1.4.2. Метод обратных итераций

После того как найдены хорошие приближения собственных значений исходной матрицы, можно, применив *метод обратных итераций*, определить и собственные векторы. В этом методе приближение к собственному вектору, отвечающему найденному собственному значению λ_j, определяют в результате последовательного решения систем уравнений

$$\left(A-\lambda_j I\right)x^{(k+1)} = evec^{(k)}$$

с последующей нормировкой решения

$$evec^{(k+1)} = \frac{x^{(k+1)}}{\left\|x^{(k+1)}\right\|_2}.$$

В качестве начального приближения берут ненулевой вектор с произвольно выбираемыми компонентами, например $evec^{(0)} = (1, 1, ..., 1)^T$. Критерием останова является незначительная разница между соответствующими компонентами векторов, полученных на двух последовательных итерациях.

Пример. Найти методом обратных итераций собственные векторы симметрической матрицы, использовав приближения ее собственных значений, вычисленные в подпрограмме *diag*.

```
program vec
 use msimsl                          ! Для вызова процедур IMSL
 use text_transfer                   ! Для вывода русских сообщений в DOS-окне
 integer(4), parameter :: n = 6, nmax = 100
 integer(4) :: i, j
 real(4) :: a(n, n), eval(n), a2(n, n), evec(n, n), x(n), sw, eps = 1.0e-4, pi
 a = reshape((/      5.0,    -4.0,   -10.0,    1.0,    1.0,   -2.0,      &
                    -4.0,     6.0,    -4.0,    1.0,    1.0,    1.0,      &
                   -10.0,    -4.0,     6.0,   -4.0,    1.0,    1.0,      &
                     1.0,     1.0,    -4.0,    6.0,   -4.0,   -3.0,      &
                     1.0,     1.0,     1.0,   -4.0,    6.0,   -4.0,      &
                    -2.0,     1.0,     1.0,   -3.0,   -4.0,    5.0 /),   &
             shape = (/ n, n /), order = (/ 2, 1 /))
 a2 = a                              ! Запоминаем исходную матрицу
 ! Приводим исходную симметрическую матрицу к трехдиагональному виду
 call hous(n, a)                     ! Текст hous приведен в разд. 7.3.2.1
 ! Находим eval – собственные значения матрицы A
 call diag(n, a, eval)               ! Текст diag приведен в разд. 8.1.4.1
 ! Ищем собственные векторы для всех собственных значений
 evec = 1.0                          ! Начальное приближение
 do j = 1, n
  a = a2                             ! Работаем с исходной матрицей
  ! Вычитая из диагонали массива a2 собственное значение eval(i), получим A - λjE
```

```
  call sadd(n, -eval(j), a, n + 1)
  do i = 1, nmax
   ! Матрица A не является положительно определенной, поэтому используем Lslsf
   call Lslsf(n, a, n, evec(:, j), x)          ! Очередное решение системы линейных уравнений
   sw = snrm2(n, x, 1)                         ! 2-норма вектора x
   x = x / sw                                  ! Нормировка вектора x
   if(all(abs(evec(:, j) - x) < eps)) exit     ! Если процесс сошелся, выходим из цикла
   evec(:, j) = x                              ! Подготовка к новой итерации
  ! Очередной собственный вектор размещается в столбце j массива evec
  end do
  print *, trim(ru_doswin('Число итераций i = ', .false.)), i, ';  eval = ', eval(j)
 end do
 pi = episf(n, n, a2, n, eval, evec, n)          ! Оценка точности решения
 call wrrrn('evec', n, n, evec, n, 0)            ! Вывод найденных собственных векторов
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program vec
```

Результат:

```
Число итераций i =     101;    eval =    17.491390
Число итераций i =       2;    eval =    10.754760
Число итераций i =     101;    eval =     8.404618
Число итераций i =       2;    eval =    -7.276278
Число итераций i =     101;    eval =     6.677792
Число итераций i =       2;    eval =    -2.052282
```

evec

	1	2	3	4	5	6
1	0.5917	-0.2811	-0.2619	0.6503	0.2812	0.0200
2	0.0391	0.3312	0.8321	0.3883	0.1375	-0.1635
3	-0.6804	-0.0820	-0.1703	0.6397	-0.2937	0.0765
4	0.3681	0.3997	-0.0055	0.0884	-0.6240	0.5546
5	-0.0572	-0.6894	0.4316	-0.0960	0.0145	0.5708
6	-0.2161	0.4118	-0.1540	0.0125	0.6528	0.5777

Оценка точности решения = 0.207

Замечания:

1. Оценку точности результата возвращает функция EPISF библиотеки IMSL. Точность считается отличной, если оценка $pi < 1$; хорошей, если $1 \le pi \le 100$, и плохой, если $pi > 100$.
2. В итерационном процессе поиска собственных векторов вместо системы $(A - \lambda_j E)x^{(k+1)} = evec^{(k)}$ с исходной $n \times n$-матрицей A можно решать систему с промежуточной трехдиагональной матрицей, возвращаемой подпрограммой *hous*. Однако собственные векторы x_i исходной матрицы не равны собственным векторам промежуточной трехдиагональной матри-

цы y_i, а вычисляются с помощью соотношения $x_i = H_1H_2 \ldots H_{n-2}y_i$, где H_k – преобразовывающая матрица Хаусхолдера.

8.2. ПРОЦЕДУРЫ IMSL ДЛЯ ВЫЧИСЛЕНИЯ СОБСТВЕННЫХ ЗНАЧЕНИЙ

8.2.1. Организация материала

Прототипами процедур IMSL, применяемых при вычислении собственных значений, являются процедуры пакета EISPACK, [69]. Ссылки на имена процедур пакета EISPACK и иные использованные при составлении процедур IMSL источники приведены в прил. 1.

Для описания процедур, решающих проблемы собственных значений, и процедур, возвращающих оценку точности решения, использованы 4 вида таблиц. Первый содержит описание имен процедур и выполняемых ими действий, второй – синтаксис вызова, третий – имена и смысл параметров, а четвертый – список возможных ошибок.

Для экономии места в таблицах, во-первых, опускаются имена процедур 2-го уровня (например, подпрограмме EVASF соответствует подпрограмма 2-го уровня E4ASF), а, во-вторых, приводятся лишь имена процедур, работающих с матрицами типа REAL(4) и COMPLEX(4). Синтаксис вызова процедур с матрицами типа REAL(8) и COMPLEX(8) аналогичен. Следует лишь добавить к имени процедуры букву D.

8.2.2. Обычная проблема собственных значений

8.2.2.1. Список, вызовы, параметры и ошибки процедур

Перечисленные в табл. 8.1 подпрограммы вычисляют собственные значения и собственные векторы вещественной или комплексной матрицы; синтаксис вызова подпрограмм приводится в табл. 8.2; параметры подпрограмм – в табл. 8.3; ошибки, которые могут возникнуть при их вызове, – в табл. 8.4.

Таблица 8.1. Подпрограммы, вычисляющие собственные значения и собственные векторы

Подпрограмма	*Назначение*
	Вещественная симметрическая матрица
EVASF	Вычисляет несколько наибольших или несколько наименьших собственных значений вещественной симметрической матрицы
EVBSF	Вычисляет собственные значения вещественной симметрической матрицы, принадлежащие заданному интервалу
EVCSF	Вычисляет все собственные значения и собственные векторы вещественной симметрической матрицы

EVESF	Вычисляет наибольшее или наименьшее собственное значение и соответствующий собственный вектор вещественной симметрической матрицы
EVFSF	Вычисляет принадлежащие заданному числовому интервалу собственные значения и соответствующие собственные векторы вещественной симметрической матрицы
EVLSF	Вычисляет все собственные значения вещественной симметрической матрицы
Вещественная симметрическая ленточная матрица	
EVASB	Вычисляет несколько наибольших или несколько наименьших собственных значений вещественной симметрической ленточной матрицы
EVBSB	Вычисляет собственные значения вещественной симметрической ленточной матрицы, принадлежащие заданному интервалу
EVCSB	Вычисляет все собственные значения и собственные векторы вещественной симметрической ленточной матрицы
EVESB	Вычисляет наибольшее или наименьшее собственное значение и соответствующий собственный вектор вещественной симметрической ленточной матрицы
EVFSB	Вычисляет принадлежащие заданному числовому интервалу собственные значения и соответствующие собственные векторы вещественной симметрической ленточной матрицы
EVLSB	Вычисляет все собственные значения вещественной симметрической ленточной матрицы
Вещественная несимметрическая матрица	
EVCRG	Вычисляет все собственные значения и собственные векторы вещественной несимметрической матрицы
EVLRG	Вычисляет все собственные значения вещественной несимметрической матрицы
Комплексная эрмитова матрица	
EVAHF	Вычисляет несколько наибольших или несколько наименьших собственных значений комплексной эрмитовой матрицы
EVBHF	Вычисляет собственные значения комплексной эрмитовой матрицы, принадлежащие заданному интервалу
EVCHF	Вычисляет все собственные значения и собственные векторы комплексной эрмитовой матрицы
EVEHF	Вычисляет наибольшее или наименьшее собственное значение и соответствующий собственный вектор комплексной эрмитовой матрицы
EVFHF	Вычисляет принадлежащие заданному числовому интервалу собственные значения и соответствующие собственные векторы комплексной эрмитовой матрицы
EVLHF	Вычисляет все собственные значения комплексной эрмитовой матрицы

Комплексная неэрмитова матрица	
EVCCG	Вычисляет все собственные значения и собственные векторы комплексной неэрмитовой матрицы
EVLCG	Вычисляет все собственные значения комплексной неэрмитовой матрицы
Вещественная матрица Хессенберга	
EVCRH	Вычисляет все собственные значения и собственные векторы вещественной верхней матрицы Хессенберга
EVLRH	Вычисляет все собственные значения вещественной верхней матрицы Хессенберга
Комплексная матрица Хессенберга	
EVCCH	Вычисляет все собственные значения и собственные векторы комплексной верхней матрицы Хессенберга
EVLCH	Вычисляет все собственные значения комплексной верхней матрицы Хессенберга

Таблица 8.2. Синтаксис вызова подпрограмм из табл. 8.1

Синтаксис вызова	*Автоматически выделяемая память*
Вещественная симметрическая матрица	
CALL EVASF(*n*, *neval*, *a*, *Lda*, *small*, *eval*)	5*n*
CALL EVBSF(*n*, *mxeval*, *a*, *Lda*, *elow*, *ehigh*, *neval*, *eval*)	6*n*
CALL EVCSF(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	4*n*
CALL EVESF(*n*, *nevec*, *a*, *Lda*, *small*, *eval*, *evec*, *ldevec*)	10*n*
CALL EVFSF(*n*, *mxeval*, *a*, *Lda*, *elow*, *ehigh*, *neval*, *eval*, *evec*, *Ldevec*)	10*n*
CALL EVLSF(*n*, *a*, *Lda*, *eval*)	3*n*
Вещественная симметрическая ленточная матрица	
CALL EVASB(*n*, *neval*, *a*, *Lda*, *ncoda*, *small*, *eval*)	*n*(*ncoda* + 4)
CALL EVBSB(*n*, *mxeval*, *a*, *Lda*, *ncoda*, *elow*, *ehigh*, *neval*, *eval*)	*n*(*ncoda* + 6)
CALL EVCSB(*n*, *a*, *Lda*, *ncoda*, *eval*, *evec*, *Ldevec*)	*n*(*ncoda* + 3)
CALL EVESB(*n*, *nevec*, *a*, *Lda*, *ncoda*, *small*, *eval*, *evec*, *ldevec*)	*n*(3*ncoda* + 7)
CALL EVFSB(*n*, *mxeval*, *a*, *Lda*, *ncoda*, *elow*, *ehigh*, *neval*, *eval*, *evec*, *Ldevec*)	*n*(3*ncoda* + 9)
CALL EVLSB(*n*, *a*, *Lda*, *ncoda*, *eval*)	*n*(*ncoda* + 2)

Вещественная несимметрическая матрица	
CALL EVCRG(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	$2n(n + 1) + 8n$
CALL EVLRG(*n*, *a*, *Lda*, *eval*)	$n(n + 6)$
Комплексная эрмитова матрица	
CALL EVAHF(*n*, *neval*, *a*, *Lda*, *small*, *eval*)	$2n^2 + 7n$
CALL EVBHF(*n*, *mxeval*, *a*, *Lda*, *elow*, *ehigh*, *neval*, *eval*)	$2n^2 + 9n + mxeval$
CALL EVCHF(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	$3n^2 + 6n$
CALL EVEHF(*n*, *nevec*, *a*, *Lda*, *small*, *eval*, *evec*, *Ldevec*)	$n(2n + nevec + 13)$
CALL EVFHF(*n*, *mxeval*, *a*, *Lda*, *elow*, *ehigh*, *neval*, *eval*, *evec*, *Ldevec*)	$2n^2 + n * mxeval + 12n + + mxeval$
CALL EVLHF(*n*, *a*, *Lda*, *eval*)	$2n^2 + 6n$
Вещественная несимметрическая матрица	
CALL EVCCG(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	$2n^2 + 6n$
CALL EVLCG(*n*, *a*, *Lda*, *eval*)	$2n^2 + 6n$
Вещественная матрица Хессенберга	
CALL EVCRH(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	$3n^2 + 4n$
CALL EVLRH(*n*, *a*, *Lda*, *eval*)	$n^2 + 4n$
Комплексная матрица Хессенберга	
CALL EVCCH(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	$4n^2 + 8n$
CALL EVLCH(*n*, *a*, *Lda*, *eval*)	$2n^2 + 2n$

Замечание. В случае двойной точности в подпрограммах автоматически выделяется $M - n$ байт памяти, где M – число байт, автоматически выделяемых в случае одинарной точности.

Таблица 8.3. Параметры подпрограмм из табл. 8.1

Имя	*Смысл*	*Типы*
a	Массив формы (*Lda*, *n*), содержащий элементы матрицы *A*. Как правило, *Lda* задается равным *n* – порядку матрицы *A*	REAL(4) или REAL(8); COMPLEX(4) или COMPLEX(8)
ehigh	Верхняя граница интервала поиска собственных значений	То же
elow	Нижняя граница интервала поиска собственных значений	"
eval	В подпрограммах EVA* и EVE* – вектор размера *neval*, содержащий максимальные, если *small* = .FALSE., или минимальные, если *small* = .TRUE., собственные значения матрицы *A*, отсортированные по убыванию их величин	См. замечание после таблицы

eval	В подпрограммах EVB* и EVF*- вектор размера *mxeval*, содержащий собственные значения матрицы *A*, отсортированные по убыванию их величин. Собственные значения расположены в интервале (*elow*, *ehigh*). Значимы только первые *neval* значений	См. замечание после таблицы
eval	В подпрограммах EVC* и EVL* – вектор размера *n*, содержащий все собственные значения матрицы *A*, отсортированные по убыванию их величин	То же
evec	Массив формы (*Ldevec*, *n*), содержащий собственные векторы. Собственный вектор с номером *j*, соответствующий собственному значению *eval(j)*, записан в столбце *j* массива *evec*. Каждый собственный вектор нормализован таким образом, что его евклидова длина равна единице	"
Lda	Ведущий размер массива *a* по первому измерению	"
Ldevec	Ведущий размер массива *evec* по первому измерению	"
mxeval	Максимальное число вычисляемых собственных значений	"
n	Порядок матрицы *A*	"
ncoda	Число кодиагоналей в симметрической ленточной матрице *A*	"
neval	Число вычисляемых в случае подпрограмм EVA* и EVE* или найденных в случае подпрограмм EVB* и EVF* собственных значений	"
nevec	Число вычисляемых в случае подпрограммы EVE* собственных значений	"
small	Если *small* = .TRUE., то вычисляются наименьшие собственные значения, если *small* =.FALSE., то – наибольшие	LOGICAL(4)

Замечание. Тип параметра *eval* является комплексным в случае вещественной несимметрической или комплексной неэрмитовой матрицы и – вещественным во всех остальных случаях. Тип параметра *evec* в случае вещественной несимметрической или комплексной матрицы совпадает с типом массива *a*; в случае вещественной несимметрической матрицы параметры *evec* имеет тип COMPLEX(4) или COMPLEX(8), если тип *a* соответственно REAL(4) или REAL(8). Это объясняется тем, что собственные значения и векторы вещественной несимметрической матрицы в общем случае являются комплексными.

Таблица 8.4. Возможные ошибки подпрограмм из табл. 8.1

Тип	*Код*	*Причина ошибки*	*Где возникает*
3	1	Итерационный процесс вычисления собственных значений не сходится. Будут возвращены наилучшие оценки	EVASF, EVESF, EVASB, EVAHF, EVEHF
3	1	Число собственных значений (*neval*) в заданном интервале превышает *mxeval*. Собственные значения подпрограммой не возвращаются; *neval* содержит число найденных собственных значений	EVBSF, EVFSF, EVBSB, EVFSB, EVBHF, EVFHF
3	1	Итерационный процесс вычисления собственных значений не сошелся за 100 итераций	EVCSF, EVLSF
3	1	Метод обратных итерации не сходится. Собственные векторы для найденных собственных значений вычислены с ошибкой	EVESB
3	2	То же	EVESF, EVFSF, EVFSB
3	2	Итерационный процесс вычисления собственных векторов не сходится. Собственным векторам присвоены нулевые значения	EVEHF, EVFHF
3	2	Потеря ортогональности собственными векторами	EVESB
3	3	То же	EVESF, EVFSF, EVFSB
3	2	Матрица не является эрмитовой. Она имеет на диагонали элемент с маленькой мнимой частью	EVAHF, EVBHF
3	3	То же	EVEHF, EVFHF
3	1	"	EVCHF, EVLHF
4	1	Итерационный процесс вычисления собственных значений не сходится. Результат не возвращается	EVCRG, EVLRG, EVCSB, EVLSB, EVCHF, EVLHF, EVCCG, EVLCG
4	1	Итерационный процесс вычисления собственных значений не сходится. Результат не возвращается	EVCRH, EVLRH, EVCCH EVLCH
4	2	Входная матрица не является эрмитовой. Она имеет на диагонали элемент с мнимой частью	EVAHF, EVBHF, EVCHF, EVEHF, EVFHF, EVLHF

Все подпрограммы можно разделить на две группы. В первой подпрограммы вычисляют как собственные значения, так и собственные векторы. Во второй – только собственные значения. Ниже будут рассмотрены подробно подпрограммы с вещественной симметрической и несимметрической

матрицами, относящиеся к первой группе. Подпрограммы второй группы в IMSL являются частными случаями подпрограмм из первой группы и поэтому в подробном описании не нуждаются. Для процедур с матрицей Хессенберга в качестве образца рассматривается и иллюстрируется примером подпрограмма EVCCH.

Подпрограммы с вещественной симметрической ленточной матрицей и с комплексными эрмитовыми и неэрмитовыми матрицами основаны на тех же идеях, что и подпрограммы с вещественной матрицей. Поэтому излагаются менее детально.

8.2.2.2. Вычисление всех собственных значений и собственных векторов вещественной симметрической матрицы

Выполняет подпрограмма EVCSF (DEVCSF):

CALL EVCSF(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *Ldevec*.

Выходные данные: *eval*, *evec*.

Параметры описаны в табл. 8.3.

Вычисляемые собственные значения возвращаются вещественным вектором *eval*, а собственные векторы – вещественным массивом *evec* формы (*Ldevec*, *n*). Собственный вектор *j*, соответствующий собственному значению *eval*(*j*), записан в столбце *j* массива *evec*. Каждый собственный вектор нормализован таким образом, что его евклидова длина равна единице.

Описание:

Первоначально, используя ортогональные преобразования подобия, матрица приводится к симметрическому трехдиагональному виду (жордановой форме). Затем для вычисления собственных значений симметрической трехдиагональной матрицы применяется *QR*-алгоритм. Приведение симметрической матрицы к трехдиагональному виду основано на процедуре TRED1 пакета EISPACK. *QR*-алгоритм, называемый также *PWK*-алгоритмом, описан в [66, p. 169]. Далее найденные собственные значения применяются для поиска собственных векторов (см. [66, p. 169, 172]).

Пример. Вычисляются собственные значения и собственные векторы симметрической матрицы; функцией EPISF возвращается оценка точности решения.

```
program evc1
 use msimsl
 use text_transfer
 integer(4), parameter :: n = 3, Lda = n, Ldevec = n
 real(4) :: a(Lda, n), eval(n), evec(Ldevec, n), pi
```

```
a = reshape((/        7.0,   -8.0,   -8.0,                  &
             -8.0,  -16.0,  -18.0,                          &
             -8.0,  -18.0,   13.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
call evcsf(n, a, Lda, eval, evec, Ldevec)   ! Вычисляем все собственные значения и векторы
pi=episf(n, n, a, Lda, eval, evec, Ldevec)  ! Вычисляем оценку точности решения
call wrrrn('eval', 1, n, eval, 1, 0)        ! Вывод результата
call wrrrn('evec', n, n, evec, Ldevec, 0)
print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evc1
```

Результат:

```
                eval
-27.90          22.68          9.22

                  evec
              1          2          3
  1      0.2945    -0.2722     0.9161
  2      0.8521    -0.3591    -0.3806
  3      0.4326     0.8927     0.1262
Оценка точности решения = 0.076
```

8.2.2.3. Вычисление нескольких наибольших или наименьших собственных значений и им соответствующих собственных векторов вещественной симметрической матрицы

Выполняет подпрограмма EVESF (DEVESF):

CALL EVESF(*n*, *nevec*, *a*, *Lda*, *small*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *small*, *Ldevec*.

Выходные данные: *eval*, *evec*.

Параметры описаны в табл. 8.3.

Число вычисляемых собственных значений равно *neval*. Вычисляются наименьшие собственные значения, если *small* = .TRUE., и наибольшие – в противном случае.

Описание:

Посредством ортогональных преобразований матрица приводится к трехдиагональной форме. Прототипом преобразовывающей процедуры является подпрограмма TRED2 пакета EISPACK, описанная в [69]. Для вычисления собственных значений трехдиагональной матрицы применяется рациональный *QR*-алгоритм, приведенный в [66, p. 169]. Собственные векторы трехдиагональной матрицы вычисляются методом обратных итераций; затем выполняется ортогонализация найденных векторов; подробнее см. в [51]. Собственные векторы исходной матрицы находятся в результате обратного преобразования собственных векторов трехдиагональной матрицы (см. процедуру TRBAK1 пакета EISPACK).

Пример. В примере используется матрица A, данная в [46, p. 55]. Вычисляются два наибольших собственных значения и их собственные векторы. Также оценивается точность результата.

```
program eve1
 use msimsl; use text_transfer
 integer(4), parameter :: n = 4, Lda = n, Ldevec = n
 integer(4) :: nevec
 logical(4) :: small
 real(4) :: a(Lda, n), eval(n), evec(Ldevec, n), pi
 a = reshape((/      5.0,    4.0,    1.0,    1.0,                &
                     4.0,    5.0,    1.0,    1.0,                &
                     1.0,    1.0,    4.0,    2.0,                &
                     1.0,    1.0,    2.0,     4.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 nevec = 2
 small = .false.                          ! Вычисляем собственные значения и векторы
 call evesf(n, nevec, a, Lda, small, eval, evec, Ldevec)
 ! Вычисляем оценку точности решения
 pi = episf(n, nevec, a, Lda, eval, evec, Ldevec)
 call wrrrn('eval', 1, nevec, eval, 1, 0); call wrrrn('evec', n, nevec, evec, Ldevec, 0)
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program eve1
```

Результат (собственный вектор, отвечающий собственному значению *eval*(*j*), размещен в столбце *j* массива *evec*):

```
          eval
        1          2
   10.000      5.000

              evec
                1          2
   1        0.6325    -0.3162
   2        0.6325    -0.3162
   3        0.3162     0.6325
   4        0.3162     0.6325
Оценка точности решения = 0.011
```

8.2.2.4. Вычисление собственных значений и собственных векторов вещественной симметрической матрицы из заданного интервала

Выполняет подпрограмма EVFSF (DEVFSF):

CALL EVFSF(*n*, *mxeval*, *a*, *Lda*, *elow*, *ehigh*, *neval*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *mxeval*, *a*, *Lda*, *elow*, *ehigh*, *Ldevec*.

Выходные данные: *neval*, *eval*, *evec*.

Параметры описаны в табл. 8.3.

Вычисляемые собственные значения принадлежат интервалу (*elow*, *ehigh*). Число найденных значений равно *neval*. Причем *neval* не должно превышать *mxeval*. В противном случае возникнет ошибка. Найденные собственные значения возвращаются вещественным вектором *eval* и расположены в нем в порядке убывания значений. Хотя размер вектора *eval* равен *mxeval*, результатом являются только первые его *neval* элементов. Найденные собственные векторы возвращаются вещественным массивом *evec* формы (*Ldevec*, *n*). Так же как и для собственных значений, результатом являются только первые *neval* столбцов *evec*. Каждый собственный вектор нормализован таким образом, что его евклидова длина равна единице.

Пример. Вычисляются принадлежащие интервалу (0.001, 1.0) собственные значения и им соответствующие собственные векторы 3×3-матрицы Гильберта, в которой $a_{ij} = 1/(i+j-1)$. Также оценивается точность результата.

```
program evf1
 use msimsl; use text_transfer
 integer(4), parameter :: n = 3, Lda = n, mxeval = 3, Ldevec = n
 integer(4) :: neval
 real(4) :: a(Lda, n), ehigh, elow, eval(mxeval), evec(Ldevec, mxeval), pi
 do j = 1, n                                  ! Задаем матрицу Гильберта
  a(:, j) = (/ (1.0 / real(i + j - 1), i = 1, n) /)
 end do
 elow  = 0.001                                ! Задаем интервал для собственных значений
 ehigh = 1.0                                  ! Вычисляем собственные значения и векторы
 call evfsf(n, mxeval, a, Lda, elow, ehigh, neval, eval, evec, Ldevec)
 ! Оценка точности решения
 pi = episf(n, neval, a, Lda, eval, evec, Ldevec)
 print *, 'neval = ', neval                   ! Число полученных решений
 call wrrrn('eval', 1, neval, eval, 1, 0); call wrrrn('evec', n, neval, evec, Ldevec, 0)
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evf1
```

Результат:

```
neval = 2
          eval
   0.1223        0.0027

              evec
                 1          2
   1        -0.5474    -0.1277
   2         0.5283     0.7137
   3         0.6490    -0.6887
Оценка точности решения =  0.008
```

8.2.2.5. Вычисление всех собственных значений и собственных векторов вещественной несимметрической матрицы

Выполняет подпрограмма EVCRG (DEVCRG):

CALL EVCRG(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *Ldevec*.

Выходные данные: *eval*, *evec*.

Параметры описаны в табл. 8.3.

Описание:

Первоначально выполняется балансировка (масштабирование) входной матрицы. Цель балансировки улучшить обусловленность задачи вычисления собственных значений за счет выравнивания норм строк и столбцов матрицы. Далее, используя ортогональные преобразования подобия, сбалансированная матрица преобразовывается в вещественную верхнюю матрицу Хессенберга. Затем для поиска собственных значений и векторов этой матрицы применяется *QR*-алгоритм, использующий двойной сдвиг. Собственные векторы нормализуются таким образом, что евклидова длина каждого равна единице.

Подпрограмма EVCRG основана на процедурах пакета EISPACK: BALANCE, выполняющей балансировку матрицы; ORTHES и ORTRAN, преобразовывающих матрицу в форму Хессенберга; HQR2, реализующей *QR*-алгоритм. Проблема также обсуждается в [51].

Пример. Используется матрица *A*, данная в [46, p. 82]. Вычисляются ее собственные значения и собственные векторы.

```
program evc1
 use msimsl; use text_transfer
 integer(4), parameter :: n = 3, Lda = n, Ldevec = n
 complex(4) :: eval(n), evec(Ldevec, n)
 real(4) :: a(Lda, n), pi
 a = reshape((/      8.0,   -1.0,   -5.0,                  &
                    -4.0,    4.0,   -2.0,                  &
                    18.0,   -5.0,   -7.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call evcrg(n, a, Lda, eval, evec, Ldevec)        ! Находим собственные значения и векторы
 pi = epirg(n, n, a, Lda, eval, evec, Ldevec)     ! Оценка точности решения
 call wrcrn('eval', 1, n, eval, 1, 0)             ! Вывод результата
 call wrcrn('evec', n, n, evec, Ldevec, 0)
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evc1
```

Результат:

```
                              eval
    ( 2.000, 4.000)        ( 2.000, -4.000)        ( 1.000, 0.000)
```

```
                    evec
          1                  2                  3
( 0.3162, 0.3162)  ( 0.3162, 0.3162)  ( 0.4082, 0.0000)
( 0.0000, 0.6325)  ( 0.0000, 0.6325)  ( 0.8165, 0.0000)
( 0.6325, 0.0000)  ( 0.6325, 0.0000)  ( 0.4082, 0.0000)
Оценка точности решения =  0.015
```

8.2.2.6. Вычисление всех собственных значений и собственных векторов комплексной эрмитовой матрицы

Выполняет подпрограмма EVCHF (DEVCHF):

CALL EVCHF(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *Ldevec*.

Выходные данные: *eval*, *evec*.

Параметры описаны в табл. 8.3. Отметим, однако, что *eval* имеет вещественный тип.

Описание:

Первоначально выполняются унитарные преобразования подобия, приводящие исходную матрицу к эквивалентному вещественному симметрическому трехдиагональному виду. Затем неявный *QL*-алгоритм применяется для поиска собственных значений и векторов этой матрицы. Найденные вещественные собственные векторы и трехдиагональная эрмитова матрица, полученная в результате преобразований исходной матрицы, используются далее для вычисления собственных векторов исходной проблемы.

Процедура преобразования матрицы основана на процедуре HTRIDI пакета EISPACK; *QL*-алгоритм – на процедуре IMTQL2 того же пакета.

Пример. Вычисляются собственные значения и собственные векторы комплексной эрмитовой матрицы. Также функцией EPIHF оценивается точность результата.

```
program evc2
 use msimsl; use text_transfer
 integer(4), parameter :: n = 3, Lda = n, Ldevec = n
 real(4) :: eval(n), pi
 complex(4) :: a(Lda, n), evec(Ldevec, n)
 a = reshape((/     (1.0, 0.0),  (1.0, -7.0),  (0.0, -1.0),          &
                    (1.0, 7.0),   (5.0, 0.0),(10.0, -3.0),           &
                    (0.0, 1.0),  (10.0, 3.0),   (-2.0, 0.0)/), shape = (/ Lda, n /), order = (/ 2, 1 /))
 call evchf(n, a, Lda, eval, evec, Ldevec)        ! Находим собственные значения и векторы
 pi = epihf(n, n, a, Lda, eval, evec, Ldevec)     ! Оценка точности решения
 call wrrrn('eval', 1, n, eval, 1, 0)             ! Вывод результата
 call wrcrn('evec', n, n, evec, Ldevec, 0)
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evc2
```

Результат:

```
              eval
 15.38      -10.63        -0.75

                          evec
                   1                  2                  3
 1    ( 0.0631,-0.4075)  (-0.0598,-0.3117)  ( 0.8539, 0.0000)
 2    ( 0.7703, 0.0000)  (-0.5939, 0.1841)  (-0.0313,-0.1380)
 3    ( 0.4668, 0.1366)  ( 0.7160, 0.0000)  ( 0.0808,-0.4942)
Оценка точности решения =  0.066
```

8.2.2.7. Вычисление всех собственных значений и собственных векторов комплексной неэрмитовой матрицы

Выполняет подпрограмма EVCCG (DEVCCG):

CALL EVCCG(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *Ldevec*.

Выходные данные: *eval*, *evec*.

Параметры описаны в табл. 8.3.

Описание:

Первоначально выполняется балансировка входной матрицы. Далее, используя унитарные преобразования подобия, сбалансированная матрица преобразовывается в комплексную верхнюю матрицу Хессенберга. Поиск собственных значений и векторов этой матрицы выполняется с применением *QR*-алгоритма.

Подпрограмма EVCCG основана на процедурах пакета EISPACK: CBAL, выполняющей балансировку матрицы; CORTH, преобразовывающей матрицу в форму Хессенберга; COMQR2, реализующей *QR*-алгоритм; CBABK2, выполняющей обратные преобразования.

Пример. Используется матрица *A*, данная в [46, p. 116]. О матрице известно, что ее собственные значения равны $1 + 5i$, $2 + 6i$, $3 + 7i$, $4 + 8i$. Вычисляются собственные значения и векторы этой матрицы.

```
program evc2
 use msimsl; use text_transfer
 integer(4), parameter :: n = 4, Lda = n, Ldevec = n
 complex(4) :: a(Lda, n), eval(n), evec(Ldevec, n)
 a = reshape((/      (5.0, 9.0),   (5.0, 5.0),  (-6.0, -6.0),  (-7.0, -7.0),        &
                     (3.0, 3.0),  (6.0, 10.0),  (-5.0, -5.0),  (-6.0, -6.0),        &
                     (2.0, 2.0),   (3.0, 3.0),   (-1.0, 3.0),  (-5.0, -5.0),        &
                     (1.0, 1.0),   (2.0, 2.0),  (-3.0, -3.0),     (0.0, 4.0) /),    &
                  shape = (/ Lda, n /), order = (/ 2, 1 /))
 call evccg(n, a, Lda, eval, evec, Ldevec)        ! Находим собственные значения и векторы
 pi = epicg(n, n, a, Lda, eval, evec, Ldevec)     ! Оценка точности решения
```

```
 call wrcrn('eval', 1, n, eval, 1, 0)              ! Вывод результата
 call wrcrn('evec', n, n, evec, Ldevec, 0)
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evc2
```

Результат:

```
                                  eval
 ( 4.000, 8.000)     ( 3.000, 7.000)     ( 2.000, 6.000)     ( 1.000, 5.000)

                                  evec
                 1                   2                   3                   4
 1  ( 0.5774, 0.0000)   ( 0.5774, 0.0000)   ( 0.3780, 0.0000)   ( 0.7559, 0.0000)
 2  ( 0.5774, 0.0000)   ( 0.5774, 0.0000)   ( 0.7559, 0.0000)   ( 0.3780, 0.0000)
 3  ( 0.5774, 0.0000)   ( 0.0000, 0.0000)   ( 0.3780, 0.0000)   ( 0.3780, 0.0000)
 4  ( 0.0000, 0.0000)   ( 0.5773, 0.0000)   ( 0.3780, 0.0000)   ( 0.3780, 0.0000)
Оценка точности решения =  0.018
```

8.2.2.8. Вычисление всех собственных значений и собственных векторов комплексной матрицы Хессенберга

Выполняет подпрограмма EVCCH (DEVCCH):

CALL EVCCH(*n*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *Ldevec*.

Выходные данные: *eval*, *evec*.

Параметры описаны в табл. 8.3. Возможная ошибка – в табл. 8.4.

Описание:

Используется *QR*-алгоритм. Прототипом EVCCH является процедура COMQR2 пакета EISPACK.

Пример. Вычисляются собственные значения и собственные векторы матрицы *A*. Также оценивается точность результата.

```
program evc3
 use msimsl; use text_transfer
 integer(4), parameter :: n = 4, Lda = n, Ldevec = n
 complex(4) :: a(Lda, n), eval(n), evec(Ldevec, n)
 a = reshape((/      (5.0, 9.0),   (5.0, 5.0),  (-6.0, -6.0),  (-7.0, -7.0),        &
                     (3.0, 3.0),  (6.0, 10.0),  (-5.0, -5.0),  (-6.0, -6.0),        &
                     (0.0, 0.0),   (3.0, 3.0),   (-1.0, 3.0),  (-5.0, -5.0),        &
                     (0.0, 0.0),   (0.0, 0.0),  (-3.0, -3.0),    (0.0, 4.0) /),     &
                  shape = (/ Lda, n /), order = (/ 2, 1 /))
 call evcch(n, a, Lda, eval, evec, Ldevec)        ! Находим собственные значения и векторы
 pi = epicg(n, n, a, Lda, eval, evec, Ldevec)     ! Оценка точности решения
 call wrcrn('eval', 1, n, eval, 1, 0)             ! Вывод результата
 call wrcrn('evec', n, n, evec, Ldevec, 0)
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evc3
```

Результат:

```
                    eval
( 8.22, 12.22)    ( 3.40,  7.40)    ( 1.60,  5.60)    ( -3.22,  0.78)

                    evec
                1                2                3                4
1  ( 0.7167, 0.0000)  (-0.0704, 0.0000)  (-0.3678, 0.0000)  ( 0.5429, 0.0000)
2  ( 0.6402, 0.0000)  (-0.0046, 0.0000)  ( 0.6767, 0.0000)  ( 0.4298, 0.0000)
3  ( 0.2598, 0.0000)  ( 0.7477, 0.0000)  (-0.3005, 0.0000)  ( 0.5277, 0.0000)
4  (-0.0948, 0.0000)  (-0.6603, 0.0000)  ( 0.5625, 0.0000)  ( 0.4920, 0.0000)
Оценка точности решения =  0.024
```

8.2.2.9. Примеры применения подпрограмм, вычисляющих собственные значения и собственные векторы

Приводятся примеры использования неразобранных в разд. 8.2.2 подпрограмм.

Пример для EVASF, вычисляющей несколько наибольших или наименьших собственных значений вещественной симметрической матрицы. Вычисляются *neval* = 3 наибольших собственных значения 10×10-матрицы Гильберта, в которой $a_{ij} = 1/(i + j - 1)$.

```
program eva1
 use msimsl
 integer(4), parameter :: Lda = 10, n = Lda, neval = 3
 integer(4) :: i, j
 real(4) :: a(Lda, n), eval(neval)
 logical(4) :: small
 do j = 1, n                                  ! Задаем матрицу Гильберта
   a(:, j) = (/ (1.0 / real(i + j - 1), i = 1, n) /)
 end do
 small = .false.
 call evasf(n, neval, a, Lda, small, eval)    ! Находим 3 наибольших собственных значения
 call wrrrn('eval', 1, neval, eval, 1, 0)     ! Вывод результата
end program eva1
```

Результат:

```
          eval
  1.752     0.343     0.036
```

Пример для EVBSF, вычисляющей собственные значения вещественной симметрической матрицы из заданного интервала. Используется матрица A, данная в [46, p. 56]. Известны ее собственные значения: -1, 5, 5 и 15. Вычисляются собственные значения из интервала (1.5, 5.5). Их число *neval* = 2.

```
program evb1
 use msimsl
```

```
integer(4), parameter :: n = 4, Lda = n, mxeval = 4
integer(4) :: neval
real(4) :: a(Lda, n), ehigh, elow, eval(mxeval)
a = reshape((/      6.0,    4.0,    4.0,    1.0,                &
                    4.0,    6.0,    1.0,    4.0,                &
                    4.0,    1.0,    6.0,    4.0,                &
                    1.0,    4.0,    4.0,    6.0 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
elow = 1.5;  ehigh = 5.5
! Находим собственные значения, принадлежащие интервалу (1.5, 5.5)
call evbsf(n, mxeval, a, Lda, elow, ehigh, neval, eval)
print *, 'neval = ', neval
call wrrrn('eval', 1, neval, eval, 1, 0)        ! Вывод результата
end program evb1
```

Результат:

```
neval =        2
          eval
   5.000          5.000
```

Пример для EVCSB, вычисляющей все собственные значения и собственные векторы вещественной симметрической ленточной матрицы

$$A=\begin{pmatrix} 5 & -4 & 1 & 0 & 0 & 0 \\ -4 & 6 & -4 & 1 & 0 & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 \\ 0 & 0 & 1 & -4 & 6 & -4 \\ 0 & 0 & 0 & 1 & -4 & 5 \end{pmatrix},$$

данной в [46, p. 75]. Собственный вектор λ_k матрицы равен

$$\lambda_k = 16\sin^4\left(\frac{k\pi}{2n+2}\right)$$

Вычисляются собственные значения и векторы вещественной симметрической ленточной матрицы A. Также выполняется оценка точности решения.

```
program evc3
 use msimsl; use text_transfer
 integer(4), parameter :: n = 6, ncoda = 2, Lda = ncoda + 1, Ldevec = n
 real(4) :: a(Lda, n), eval(n), evec(Ldevec, n), pi
 ! Ленточное представление симметрической матрицы A
 a = reshape((/      0.0,     0.0,     1.0,     1.0,     1.0,     1.0,      &
                     0.0,    -4.0,    -4.0,    -4.0,    -4.0,    -4.0,      &
                     5.0,     6.0,     6.0,     6.0,     6.0,     5.0 /),    &
                 shape = (/ Lda, n /), order = (/ 2, 1 /))
 ! Находим собственные значения и собственные векторы
 call evcsb(n, a, Lda, ncoda, eval, evec, Ldevec)
```

```
! Оценка точности решения и вывод результата
pi = episb(n, n, a, Lda, ncoda, eval, evec, Ldevec)
call wrrrn('eval', 1, n, eval, 1, 0); call wrrrn('evec', n, n, evec, Ldevec, 0)
print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения pi =', .false.)), pi
end program evc3
```

Результат:

eval

1	2	3	4	5	6
14.45	10.54	5.98	2.42	0.57	0.04

evec

	1	2	3	4	5	6
1	0.2319	-0.4179	-0.5211	0.5211	-0.4179	0.2319
2	-0.4179	0.5211	0.2319	0.2319	-0.5211	0.4179
3	0.5211	-0.2319	0.4179	-0.4179	-0.2319	0.5211
4	-0.5211	-0.2319	-0.4179	-0.4179	0.2319	0.5211
5	0.4179	0.5211	-0.2319	0.2319	0.5211	0.4179
6	-0.2319	-0.4179	0.5211	0.5211	0.4179	0.2319

Оценка точности решения pi = 0.024

8.2.3. Обобщенная проблема собственных значений

8.2.3.1. Список, вызовы, параметры и ошибки процедур

Приводимые в табл. 8.5 подпрограммы вычисляют собственные значения и собственные векторы вещественной или комплексной матрицы обобщенной проблемы $Az = \lambda Bz$. В табл. 8.6 дается синтаксис вызова подпрограмм; их параметры описаны в табл. 8.7, а возможные ошибки – в табл. 8.8.

Таблица 8.5. Подпрограммы, вычисляющие собственные значения и собственные векторы обобщенной проблемы $Az = \lambda Bz$

Подпрограмма	*Назначение*
Вещественные симметрические матрица A и B (B вдобавок – положительно определенная)	
GVCSP	Вычисляет все собственные значения и собственные векторы обобщенной собственной системы $Az = \lambda Bz$
GVLSP	Вычисляет все собственные значения обобщенной собственной системы $Az = \lambda Bz$
Вещественные несимметрические матрицы A и B	
GVCRG	Вычисляет все собственные значения и собственные векторы обобщенной собственной системы $Az = \lambda Bz$
GVLRG	Вычисляет все собственные значения обобщенной собственной системы $Az = \lambda Bz$

Комплексные неэрмитовы матрицы A и B	
GVCCG	Вычисляет все собственные значения и собственные векторы обобщенной собственной системы $Az = \lambda Bz$
GVLCG	Вычисляет все собственные значения обобщенной собственной системы $Az = \lambda Bz$

Таблица 8.6. Синтаксис вызова подпрограмм из табл. 8.5

Синтаксис вызова	*Автоматически выделяемая память*
Вещественные симметрические матрица A и B (B вдобавок – положительно определенная)	
CALL GVCSP(*n*, *a*, *Lda*, *b*, *Ldb*, *eval*, *evec*, *Ldevec*)	$n^2 + 5n$
CALL GVLSP(*n*, *a*, *Lda*, *b*, *Ldb*, *eval*)	$n^2 + 4n$
Вещественные несимметрические матрицы A и B	
CALL GVCRG(*n*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*, *evec*, *Ldevec*)	$3n^2 + 4n$
CALL GVLRG(*n*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*)	$2n^2 + 4n$
Комплексные неэрмитовы матрицы A и B	
CALL GVCCG(*n*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*, *evec*, *Ldevec*)	$4n^2 + 4n$
CALL GVLCG(*n*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*)	То же

Замечание. В случае двойной точности в подпрограммах автоматически выделяется $M - n$ байт памяти, где M – число байт, автоматически выделяемых в случае одинарной точности.

Таблица 8.7. Параметры подпрограмм из табл. 8.5

Имя	*Смысл*	*Типы*
a	Массив формы (*Lda*, *n*), содержащий элементы матрицы *A*. Как правило, *Lda* задается равным *n* – порядку матрицы *A*	REAL(4) или REAL(8); COMPLEX(4) или COMPLEX(8)
alpha	Комплексный вектор размера *n*, содержащий скаляры α_j; Если $\beta_j \neq 0$, то собственное значение $\lambda_j = \alpha_j/\beta_j$, где $\alpha_j = alpha(j)$, $\beta_j = beta(j)$	COMPLEX(4) или COMPLEX(8)
b	Массив формы (*Ldb*, *n*), содержащий элементы матрицы *B*. Матрица *B* в случае GVCSP (DGVCSP) является вещественной симметрической положительно определенной. Как правило, *Ldb* задается равным *n* – порядку матрицы *B*	Совпадает с типом массива *a*
beta	Вещественный вектор размера *n*, содержащий скаляры β_j	REAL(4) или REAL(8)

eval	Вещественный вектор размера *n*, содержащий собственные значения матрицы *A*, отсортированные по убыванию их величин	REAL(4) или REAL(8)
evec	Массив формы (*Ldevec*, *n*), содержащий собственные векторы. Собственный вектор с номером *j* хранится в столбце *j* массива *evec* и соответствует собственному значению с тем же номером, которое записано либо в *eval*(*j*), либо в *alpha*(*j*) и *beta*(*j*). Каждый собственный вектор нормализован таким образом, что его евклидова длина равна единице	Совпадает с типом массива *a*
Lda	Ведущий размер массива *a* по первому измерению	То же
Ldb	Ведущий размер массива *b* по первому измерению	"
Ldevec	Ведущий размер массива *evec* по первому измерению	"
n	Порядок матриц *A* и *B*	"

Замечание. Тип параметра *evec* в случае вещественной несимметрической или комплексной матрицы совпадает с типом массива *a*; в случае вещественной симметрической матрицы параметр *evec* имеет тип COMPLEX(4), если тип *a* – REAL(4), и COMPLEX(8), если тип *a* – REAL(8).

Таблица 8.8. Возможные ошибки подпрограмм из табл. 8.5

Тип	*Код*	*Причина ошибки*	*Где возникает*
4	1	Итерационный процесс вычисления собственных значений не сходится. Результат не возвращается	GVCSP, GVLSP, GVCRG, GVLRG, GVCCG, GVLCG
4	2	Матрица *B* не является положительно определенной	GVCSP, GVLSP

Все подпрограммы можно разделить на две группы. В первой подпрограммы вычисляют как собственные значения, так и собственные векторы. Во второй – только собственные значения. Ниже будут рассмотрены и проиллюстрированы примерами подпрограммы с вещественными матрицами *A* и *B*, принадлежащие к первой группе.

8.2.3.2. Вычисление всех собственных значений и собственных векторов вещественной симметрической обобщенной системы $Az = \lambda Bz$

Выполняет подпрограмма GVCSP (DGVCSP):

CALL GVCSP(*n*, *a*, *Lda*, *b*, *Ldb*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *b*, *Ldb*, *Ldevec*.

Выходные данные: *eval*, *evec*.

Параметры описаны в табл. 8.7. Отметим, однако, что массив *b* формы (*Ldb*, *n*) представляет вещественную симметрическую положительно определенную $n \times n$-матрицу *B*.

Вычисляемые собственные значения возвращаются вещественным вектором *eval*, а собственные векторы – вещественным массивом *evec* формы (*Ldevec*, *n*). Собственный вектор *j*, соответствующий собственному значению *eval*(*j*), записан в столбце *j* массива *evec*. Каждый собственный вектор нормализован таким образом, что его евклидова длина равна единице.

Замечания:

1. Возможные ошибки перечислены в табл. 8.8.
2. Качество полученного результата оценивается функцией GPISP.

Описание:

Первоначально, применяя разложение Холецкого, матрица B представляется в виде $B = R^TR$, где R – верхняя треугольная матрица. Это позволяет привести систему $Az = \lambda Bz$ к виду $(R^TAR^{-1})(Rz) = \lambda(Rz)$. Затем вычисляются собственные значения и собственные векторы матрицы $C = R^{-T}AR^{-1}$. Обобщенные собственные векторы матрицы A равны $z = R^{-1}x$, где x – собственные векторы C. Подход изложен в [63]. Разложение Холецкого основано на процедуре LFTDS пакета IMSL. Собственные значения и собственные векторы матрицы C вычисляются методом, положенным в основу процедуры EVCSF. Проблема также обсуждается в [51].

Пример. Находятся обобщенные собственные значения и собственные векторы; функцией GPISP возвращается оценка точности решения.

```
program evc4
 use msimsl; use text_transfer
 integer(4), parameter :: n = 3, Lda = n, Ldb = n, Ldevec = n
 real(4) :: a(Lda, n), b(Ldb, n), eval(n), evec(Ldevec, n), pi
 a = reshape((/      1.1,    1.2,    1.4,        &
                     1.2,    1.3,    1.5,        &
                     1.4,    1.5,     1.6 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
 b = reshape((/      2.0,    1.0,    0.0,        &
                     1.0,    2.0,    1.0,        &
                     0.0,    1.0,     2.0 /), shape = (/ Ldb, n /), order = (/ 2, 1 /))
 ! Вычисляем все собственные значения и векторы
 call gvcsp(n, a, Lda, b, Ldb, eval, evec, Ldevec)
 ! Вычисляем оценку точности решения и выводим результат
 pi = gpisp(n, n, a, Lda, b, Ldb, eval, evec, Ldevec)
 call wrrrn('eval', 1, n, eval, 1, 0); call wrrrn('evec', n, n, evec, Ldevec, 0)
 print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evc4
```

Результат:

```
                eval
  1.386       -0.058       -0.003
```

evec

	1	2	3
1	0.6431	-0.1147	-0.6817
2	-0.0224	-0.3591	0.7266
3	0.7655	0.7174	-0.0858

Оценка точности решения = 0.298

8.2.3.3. Вычисление всех собственных значений и собственных векторов вещественной обобщенной системы $Az = \lambda Bz$

Выполняет подпрограмма GVCRG (DGVCRG):

CALL GVCRG(*n*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*, *evec*, *Ldevec*)

Входные данные: *n*, *a*, *Lda*, *b*, *Ldb*, *Ldevec*.

Выходные данные: *alpha*, *beta*, *evec*.

Параметры описаны в табл. 8.7. Отметим, однако, что если $beta(i) \neq 0$, то собственное значение $\lambda_i = alpha(i)/beta(i)$ ($i = 1, ..., n$). Причем *alpha* – комплексный вектор, а *beta* – вещественный. Размеры *alpha* и *beta* равны *n*.

Замечания:

1. Возможные ошибки перечислены в табл. 8.8.
2. Качество полученного результата оценивается функцией GPIRG.

Описание:

Собственные значения в рассматриваемой проблеме могут быть бесконечно большими. Поэтому вместо λ подпрограмма GVCRG возвращает комплексный вектор *alpha* и вещественный вектор *beta*, и если $beta(i) \neq 0$, то $\lambda_i = alpha(i)/beta(i)$. Для проблемы с маленьким значением $beta(i)$ можно решить эквивалентную проблему $Bx = \mu Ax$, где $\mu = \lambda - 1$.

На первом этапе применяется *QZ*-алгоритм и одновременно матрица A преобразовывается в верхнюю форму Хессенберга, а матрица B – в верхнюю треугольную форму. Затем ортогональными преобразованиями матрица A приводится к верхней псевдотреугольной форме и вычисляются собственные значения и собственные векторы редуцированной проблемы.

Подпрограмма GVCRG основана на *QZ*-алгоритме, предложенном в [64], и реализована в том виде, как это выполнено в процедурах QZHES, QZIT и QZVAL пакета EISPACK.

Пример. Находятся обобщенные собственные значения и собственные векторы для систем $Az = \lambda Bz$ и $Bx = \mu Ax$, где $\mu = \lambda - 1$. Функция GPIRG возвращает оценку точности решения.

```
program evc5
 use msimsl, nouse => beta; use text_transfer
 integer(4), parameter :: n = 3, Lda = n, Ldb = n, Ldevec = n
```

```
integer(4) :: i
real(4) :: a(Lda, n), b(Ldb, n), beta(n), pi
complex(4) :: alpha(n), eval(n), evec(Ldevec,n)
a = reshape((/        1.0,    0.5,    0.0,                &
                    -10.0,    2.0,    0.0,                &
                      5.0,    1.0,    0.5 /), shape = (/ Lda, n /), order = (/ 2, 1 /))
b = reshape((/        0.5,    0.0,    0.0,                &
                      3.0,    3.0,    0.0,                &
                      4.0,    0.5,    1.0 /), shape = (/ Ldb, n /), order = (/ 2, 1 /))
! Вычисляем все собственные значения и собственные векторы
call gvcrg(n, a, Lda, b, Ldb, alpha, beta, evec, Ldevec)
eval = alpha / beta                          ! Вычисляем собственные значения
! Вычисляем оценку точности решения и выводим результат
pi = gpirg(n, n, a, Lda, b, Ldb, alpha, beta, evec, Ldevec)
call wrcrn('eval', 1, n, eval, 1, 0); call wrcrn('evec', n, n, evec, Ldevec, 0)
print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
! Находим обратные величины (меняем местами a и b и решаем Bx = μAx)
call gvcrg(n, b, Ldb, a, Lda, alpha, beta, evec, Ldevec)
eval = alpha / beta                          ! Вычисляем собственные значения
! Вычисляем оценку точности решения и выводим результат
pi = gpirg(n, n, b, Ldb, a, Lda, alpha, beta, evec, Ldevec)
call wrcrn('eval reciprocals', 1, n, eval, 1, 0); call wrcrn('evec', n, n, evec, Ldevec, 0)
print "(1x, a, f6.3)", trim(ru_doswin('Оценка точности решения =', .false.)), pi
end program evc5
```

Результат:

eval

(0.833, 1.993)	(0.833,-1.993)	(0.500, 0.000)

evec

	1	2	3
1	(-0.197, 0.150)	(-0.197, 0.150)	(0.000, 0.000)
2	(-0.069, 0.568)	(-0.069, 0.568)	(0.000, 0.000)
3	(0.782, 0.000)	(0.782, 0.000)	(1.000, 0.000)

Оценка точности решения = 0.301

eval reciprocals

(2.000, 0.000)	(0.179, 0.427)	(0.179, 0.427)

evec

	1	2	3
1	(0.000, 0.000)	(-0.197, 0.150)	(-0.197, 0.150)
2	(0.000, 0.000)	(-0.069, 0.568)	(-0.069, 0.568)
3	(1.000, 0.000)	(0.782, 0.000)	(0.782, 0.000)

Оценка точности решения = 0.283

8.3. ОЦЕНКА ТОЧНОСТИ РЕШЕНИЯ ПРОБЛЕМЫ СОБСТВЕННЫХ ЗНАЧЕНИЙ

Рассматриваемые в разделе функции, используя ранее вычисленные собственные значения и собственные векторы, позволяют оценить точность полученного результата.

8.3.1. Список, вызовы, параметры и ошибки процедур

Оценка точности решения проблемы собственных значений с различными матрицами возвращается приводимыми в табл. 8.9 функциями; синтаксис их вызова дается в табл. 8.10.

Таблица 8.9. Функции, возвращающие оценку точности решения

Функция	*Вид проблемы, для которой вычисляется оценка*
EPISF	С вещественной симметрической матрицей
EPISB	С вещественной симметрической ленточной матрицей
EPIRG	С вещественной несимметрической матрицей
EPIHF	С комплексной эрмитовой матрицей
EPICG	С комплексной неэрмитовой матрицей
GPISP	Обобщенная с вещественной симметрической матрицей
GPIRG	Обобщенная с вещественной несимметрической матрицей
GPICG	Обобщенная с комплексной неэрмитовой матрицей

Таблица 8.10. Синтаксис вызова функций из табл. 8.9

Синтаксис вызова	*Автоматически выделяемая память*
pi = EPISF(*n*, *neval*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	*n*
pi = EPISB(*n*, *neval*, *a*, *Lda*, *ncoda*, *eval*, *evec*, *Ldevec*)	"
pi = EPIRG(*n*, *neval*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	2*n*
pi = EPIHF(*n*, *neval*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	"
pi = EPICG(*n*, *neval*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)	"
pi = GPISP(*n*, *neval*, *a*, *Lda*, *b*, *Ldb*, *eval*, *evec*, *Ldevec*)	"
pi = GPIRG(*n*, *neval*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*, *evec*, *Ldevec*)	4*n*
pi = GPICG(*n*, *neval*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*, *evec*, *Ldevec*)	"

Замечание. В случае двойной точности автоматически выделяется памяти в 2 раза больше.

Все параметры функций табл. 8.10 являются входными. Смысл параметров описан в табл. 8.3 и 8.7. Однако ряд параметров, требующих уточнения,

приведен в табл. 8.11. В табл. 8.12 перечислены ошибки, которые могут возникнуть при вызове функций из табл. 8.10.

Таблица 8.11. Некоторые параметры функций из табл. 8.9

Имя	*Смысл*	*Тип*
alpha	Комплексный вектор размера *neval*, содержащий числители собственных значений	Совпадает с типом массива *a*
beta	Вектор размера *neval*, содержащий знаменатели собственных значений	Совпадает с типом массива *a*
eval	Вектор размера *neval*, содержащий собственные значения матрицы *A*	См. замечание 1 после табл. 8.3
evec	Массив формы (*Ldevec*, *n*), содержащий *neval* собственных векторов матрицы *A*	То же
neval	Число собственных значений (собственных векторов) на базе которых вычисляется оценка точности решения проблемы	INTEGER(4)

Таблица 8.12. Возможные ошибки функций из табл. 8.9

Тип	*Код*	*Причина ошибки*	*Где возникает*
3	1	Оценка точности решения больше 100	EPICG, EPIHF, EPIRG, EPISB, EPISF, GPICG
3	2	Нулевой собственный вектор	EPICG, EPIHF, EPIRG, EPISB, EPISF, GPICG, GPIRG, GPISP
3	3	Нулевая матрица *A*	EPICG, EPIHF, EPIRG, EPISB, EPISF, GPICG, GPIRG, GPISP
3	4	Нулевая матрица *B*	GPICG, GPIRG, GPISP

В качестве примера дадим описания функций EPISF и GPICG.

Функция EPISF (DEPISF) возвращает оценку точности решения обычной проблемы собственных значений с вещественной симметрической матрицей. Ее синтаксис:

pi = EPISF(*n*, *neval*, *a*, *Lda*, *eval*, *evec*, *Ldevec*)

Входные данные: *n*, *neval*, *a*, *Lda*, *eval*, *evec*, *Ldevec*.

Выходные данные: EPISF.

Параметры описаны в табл. 8.3 и 8.11.

Описание:

Пусть $m = neval$, $\lambda = eval$, $x_j = evec(:, j)$, $\varepsilon_м$ – машинная точность. Тогда оценка точности решения τ равна

$$\tau = \max_{1 \le j \le m} \frac{\| Ax_j - \lambda_j x_j \|_1}{10 \ n \ \varepsilon_м \ \| A \|_1 \ \| x_j \|_1}.$$

Значение τ зависит от используемого компьютера. Точность решения EVCSF считается отличной, если $\tau < 1$, хорошей, если $1 \le \tau \le 100$, и плохой, если $\tau > 100$. Более подробно см. в [69, p. 124-125].

Пример для EPISF приведен при рассмотрении EVCSF.

Функция GPICG (DGPICG) возвращает оценку точности решения обобщенной проблемы собственных значений с комплексной неэрмитовой матрицей. Ее синтаксис:

pi = GPICG(*n*, *neval*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*, *evec*, *Ldevec*)

Входные данные: *n*, *neval*, *a*, *Lda*, *b*, *Ldb*, *alpha*, *beta*, *evec*, *Ldevec*.

Выходные данные: GPICG.

Параметры описаны в табл. 8.3, 8.7 и 8.11.

Собственное значение с номером *j* равно *alpha*(*j*) / *beta*(*j*); соответствующий этому собственному значению собственный вектор размещается в столбце *j* массива *evec*.

Описание:

Пусть $m = neval$, $x_j = evec(:, j)$, $\varepsilon_м$ – машинная точность. Тогда оценка точности решения τ равна

$$\tau = \max_{1 \le j \le m} \frac{\| \beta_j A x_j - \alpha_j B x_j \|_1}{\varepsilon_м \left(\beta_j \| A \|_1 + |\alpha_j| \| B \|_1 \right) \| x_j \|_1}.$$

Для вычисления 1-нормы комплексного вектора *v* используется формула

$$\| v \|_1 = \sum_{i=1}^{n} \left(|\mathrm{REAL}(v_i)| + |\mathrm{AIMAG}(v_i)| \right).$$

Значение τ зависит от используемого компьютера. Точность решения EVCSF считается отличной, если $\tau < 1$, хорошей, если $1 \le \tau \le 100$, и плохой, если $\tau > 100$. Более подробно см. в [42, p. 77-79].

9. РЕШЕНИЕ ПРЯМОУГОЛЬНЫХ СИСТЕМ $Ax \approx b$ С ОГРАНИЧЕНИЯМИ И БЕЗ НИХ

9.1. МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

9.1.1. Постановка задачи

Рассмотрим задачу подбора эмпирической формулы по экспериментальным данным. Пусть, например, получены данные

x	1.0	2.0	3.0	4.0	5.0
y	5.0	9.1	28.1	63.9	125.2

и требуется найти коэффициенты c_i функции

$$f(x) = c_0 + c_1x + c_2x^2 + c_3x^3,$$

такие, что сумма отклонений

$$S = \sum_{i=0}^{3}[f(x_i) - y_i]^2 \tag{9.1}$$

минимальна.

В такой постановке задача подбора эмпирической формулы решается по *методу наименьших квадратов*.

Уравнение (9.1) называется *моделью*, а коэффициенты c_i – ее *параметрами*. В общем случае модель имеет вид

$$f(x) = c_0\varphi_0(x) + c_1\varphi_1(x) + \ldots + c_n\varphi_n(x),$$

в котором $\varphi_i(x)$ – модельные функции.

Например, $f(x) = c_0 + c_1x$ является линейной моделью, а $f(x) = c_0e^{c_1x}$ – экспоненциальной. Вопросы выбора оптимальной модели обсуждаются, например, в [1].

Задачу наименьших квадратов можно записать в матричном виде. Составим прежде систему

$$\begin{array}{ccccccccc} c_0\varphi_0(x_0) & + & c_1\varphi_1(x_0) & + & \ldots & + & c_n\varphi_n(x_0) & \approx & y_0, \\ c_0\varphi_0(x_1) & + & c_1\varphi_1(x_1) & + & \ldots & + & c_n\varphi_n(x_1) & \approx & y_1, \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ c_0\varphi_0(x_m) & + & c_1\varphi_1(x_m) & + & \ldots & + & c_n\varphi_n(x_m) & \approx & y_m. \end{array} \tag{9.2}$$

или

$$Ac \approx y,$$

где

$$A=\begin{pmatrix}\varphi_0(x_0) & \varphi_1(x_0) & \dots & \varphi_n(x_0)\\ \varphi_0(x_1) & \varphi_1(x_1) & \dots & \varphi_n(x_1)\\ \dots & \dots & \dots & \dots\\ \varphi_0(x_m) & \varphi_1(x_m) & \dots & \varphi_n(x_m)\end{pmatrix},\ c=\begin{pmatrix}c_0\\ c_1\\ \dots\\ c_m\end{pmatrix},\ y=\begin{pmatrix}y_0\\ y_1\\ \dots\\ y_m\end{pmatrix}.$$

Система (9.2) может быть *переопределенной*, если $m > n$, и *недоопределенной* при $m < n$.

В методе наименьших квадратов параметры c_i выбираются таковыми, что минимизируется сумма квадратов невязок

$$\min_c \sum_{i=0}^{m}\left[(y-Ac)_i\right]^2 . \tag{9.3}$$

Так, в случае уравнения (9.1) минимизируются невязки

$$\min_c \sum_{i=0}^{3}\left[y_i-(c_0+c_1x_i+c_2x_i^2+c_3x_i^3)\right]^2 .$$

Используя 2-норму, выражение (9.3) можно записать так:

$$\min_c \|y-Ac\|_2^2=(y-Ac)^T(y-Ac) .$$

9.1.2. Применение *QR*-разложения в методе наименьших квадратов

Перейдем к привычным для систем линейных уравнений обозначениям и рассмотрим задачу минимизации $\|b-Ax\|_2^2$ с тремя параметрами и пятью наблюдениями, в которой A – верхняя треугольная матрица:

$$A=\begin{pmatrix}1&2&1\\0&3&5\\0&0&6\\0&0&0\\0&0&0\end{pmatrix},\quad b=\begin{pmatrix}5\\9\\18\\4\\3\end{pmatrix}.$$

Выполним преобразования

$$b-Ax=\begin{pmatrix}b_{(1)}\\ b_{(2)}\end{pmatrix}-\begin{pmatrix}R\\0\end{pmatrix}x=\begin{pmatrix}b_{(1)}-Rx\\ b_{(2)}\end{pmatrix},$$

где

$$R=\begin{pmatrix}1&2&1\\0&3&5\\0&0&6\end{pmatrix},\quad b_{(1)}=\begin{pmatrix}5\\9\\18\end{pmatrix},\quad b_{(2)}=\begin{pmatrix}4\\3\end{pmatrix}.$$

Тогда для рассматриваемого случая

$$\|b - Ax\|_2^2 = \|b_{(1)} - Rx\|_2^2 + \|b_{(2)}\|_2^2 \qquad (9.4)$$

Очевидно, что, поскольку $\|b_{(2)}\|_2^2$ не зависит от x, минимум (9.4) достигается, когда $\|b_{(1)} - Rx\|_2^2 = 0$, т. е. когда $x_2 = 18/6 = 3$, $x_1 = (9 - 5x_2)/3 = -2$, $x_0 = 5 - x_2 - 2x_1 = 6$. Таким образом, вектор $x = (6, -2, 3)^T$ является решением задачи наименьших квадратов.

Из приведенного примера следует, что задачу наименьших квадратов можно решать, приводя общего вида $m \times n$-матрицу A к верхней треугольной матрице, сохраняя при этом 2-норму. Такие преобразования выполняются посредством QR-разложения матрицы, получаемого в результате применения последовательности ортогональных преобразований Хаусхолдера. О том, что ортогональное преобразование P сохраняет 2-норму, говорят равенства

$$\|Px\|_2 = \sqrt{(Px)^T(Px)} = \sqrt{x^T P^T P x} = \sqrt{x^T x} = \|x\|_2 .$$

Алгоритм решения задачи наименьших квадратов $Ax \approx b$:

1. Найти QR-разложение $A = QR$ и выделить вектор $b_{(1)}$.
2. Решить систему линейных уравнений $Rx = Q^{-1}b_{(1)}$, в которой, поскольку Q является ортогональной матрицей, $Q^{-1} = Q^T$.

Пример. Решается задача $Ax \approx b$; QR-разложение матрицы A выполняется приведенной в разд. 7.3.1 программой *QRhous*.

```
program Least0
 use msimsl
 integer(4), parameter :: nra = 5, nca = 3
 real(4) :: a(nra, nca), r(nra, nca), u(nra), b(nra), b2(nra), q(nra, nra), x(nca), res(nra)
 a = reshape((/      1.0,    1.0,    1.0,                &
                     1.0,    2.0,    4.0,                &
                     1.0,    3.0,    9.0,                &
                     1.0,    4.0,   16.0,                &
                     1.0,    5.0,   25.0 /), shape = (/ nra, nca /), order = (/ 2, 1 /))
 b = (/ 1.0, 2.3, 4.6, 3.1, 1.2 /)
 r = a; x = 0.0                          ! Сохраняем матрицу A для вычисления невязок
 ! Поиск QR-разложения. Матрица R возвращается на место матрицы A
 ! Текст подпрограмм QRhous и getQ приведен в разд. 7.3.1
 call QRhous(nra, nca, r)                ! Возвращает матрицу R и векторы Хаусхолдера
 call getQ(nra, nca, r, u, q)            ! Формируем матрицу Q
 b2 = matmul(transpose(q), b)            ! Преобразование b2 =Q^T b
 ! Решаем систему линейных уравнений Rx = b2(1), где b2(1) = b2(1:nca)
 call Lslrt(nca, r, nra, b2(1:nca), 2, x)  ! Верхняя треугольная матрица (ipath = 2)
 call wrrrn('x', 1, nca, x, 1, 0)        ! Вывод результатов
```

```
res = b - matmul(a, x)                    ! Вычисляем и выводим невязки
call wrrrn('res', 1, nra, res, 1, 0)
end program Least0
```

Результат:

```
                x
-3.020        4.491        -0.729

                           res
0.2571      -0.7486      .07029      -0.1886      -0.0229
```

9.1.3. Вырожденная задача наименьших квадратов

Задача наименьших квадратов является *вырожденной*, если ее решение не единственно. В такой задаче после выполнения QR-разложения $m \times n$-матрицы A число ненулевых строк матрицы R меньше n. Например, в QR-разложении 5×4-матрицы

$$A = \begin{pmatrix} 1.0 & 5.0 & 15.0 & 5.0 \\ 1.0 & 4.0 & 17.0 & 4.0 \\ 1.0 & 7.0 & 14.0 & 7.0 \\ 1.0 & 4.0 & 18.0 & 3.0 \\ 1.0 & 4.0 & 10.0 & 4.0 \end{pmatrix} = Q \begin{pmatrix} 2.24 & -10.29 & 33.09 & -10.29 \\ 0.0 & 3.03 & -1.78 & 3.03 \\ 0.0 & 0.0 & -5.97 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

матрица R имеет 3 ненулевых строки. Следовательно, число независимых столбцов матрицы A, так же как и R, равно трем и ранг матрицы A также равен трем.

Для выявления вырожденных матриц QR-разложение выполняется с выбором главного столбца, в качестве которого на каждом шаге принимается столбец с наибольшей 2-нормой.

Алгоритм QR-разложения с выбором главного столбца для шага k:

1. Вычислить 2-нормы для столбцов с номерами k, $k + 1$, ..., n, учитывая при вычислении 2-нормы столбца только элементы k, $k + 1$, ..., m, где m – число строк матрицы.
2. Найти столбец с наибольшей 2-нормой (пусть его номер равен kma).
3. Переставить столбцы k и kma.
4. Выполнить преобразование Хаусхолдера для рассматриваемых столбцов.

После разложения $A = QRP$, где P – матрица перестановок.

В результате такого разложения диагональные элементы матрицы R располагаются в порядке убывание их абсолютных значений. Тогда, если задача вырожденная, матрицу R можно представить в блочном виде

$$R = \begin{pmatrix} R_{(1)} & R_{(2)} \\ 0 & 0 \end{pmatrix},$$

где $R_{(1)}$ – верхняя треугольная матрица, а $R_{(2)}$ – прямоугольная матрица. Причем строка k матрицы R считается ненулевой, если $|R_{kk}| \le tol * |R_1|$, где tol – некоторый допуск.

В задаче наименьших квадратов минимизируется

$$\| b - Ax \|_2^2 = \| b - QRPx \|_2^2 = \| Q^T(b - QRPx) \|_2^2 = \| Q^T b - RPx \|_2^2 .$$

Переход от второго выражения к третьему позволителен, поскольку ортогональное преобразование Q^T не меняет 2-норму.

Введем обозначения $y = Px$, $c = Q^T b$ и разобьем y и c на части, такие же, как и в матрице R: $y = (y_{(1)}, y_{(2)})^T$ и $c = (c_{(1)}, c_{(2)})^T$. Тогда

$$\left\| Q^T b - RPx \right\|_2^2 = \left\| \begin{pmatrix} c_{(1)} \\ c_{(2)} \end{pmatrix} - \begin{pmatrix} R_{(1)} & R_{(2)} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_{(1)} \\ y_{(2)} \end{pmatrix} \right\|_2^2 = \| c_{(1)} - R_{(1)} y_{(1)} - R_{(2)} y_{(2)} \|_2^2 + \| c_{(2)} \|_2^2 .$$

Из последнего выражения следует, что его минимум достигается, когда

$$R_{(1)} y_{(1)} = c_{(1)} - R_{(2)} y_{(2)} .$$

Поскольку такое равенство возможно при произвольных значениях $y_{(2)}$, обычно полагают $y_{(2)} = 0$ и решают систему линейных уравнений $R_{(1)} y_{(1)} = c_{(1)}$, находя $y_{(1)}$, а затем x – решение задачи наименьших квадратов.

Алгоритм решения задачи наименьших квадратов $Ax \approx b$:

1. Вычислить $A = QRP$ и выделить $R_{(1)}$.
2. Найти $c = Q^T b$.
3. Решить $R_{(1)} y_{(1)} = c_{(1)}$.
4. Положить $y = (y_{(1)}, 0)^T$ и найти $x = P^T y$.

Пункт 4 возможен, поскольку матрица перестановок является ортогональной.

Пример. Решается задача наименьших квадратов с выбором главных столбцов.

```
program Least1
 use msimsl
 integer(4), parameter :: nra = 8, nca = 4, Lda = nra
 integer(4) :: i, ipvt(nca), kbasis
 real(4) :: a(Lda, nca), r(Lda, nca), q(nra, nra), u(nra), b(nra), c(nra), x(nca), res(nra), tol
 a = reshape((/      1.0,    5.0,   15.0,    5.0,                &
                     1.0,    4.0,   17.0,    4.0,                &
                     1.0,    7.0,   14.0,    7.0,                &
                     1.0,    3.0,   18.0,    3.0,                &
                     1.0,    1.0,   15.0,    1.0,                &
                     1.0,    8.0,   11.0,    8.0,                &
                     1.0,    3.0,    9.0,    3.0,                &
                     1.0,    4.0,   10.0,    4.0 /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
```

```
b = (/ 30.0, 31.0, 35.0, 29.0, 18.0, 35.0, 20.0, 22.0 /)
r = a; tol = 1.0e-4; x = 0.0                  ! Сохраняем матрицу A для вычисления невязок
call QRhous2(nra, nca, r, ipvt)               ! QR-разложение с выбором главных столбцов
kbasis = 1                                    ! Находим ранг матрицы A
do i = 2, nca
 if(abs(r(i, i)) <= tol * abs(r(1, 1))) exit
 kbasis = kbasis + 1
end do
print *, 'kbasis = ', kbasis                  ! Ранг матрицы A
! Текст подпрограммы getQ приведен в разд. 7.3.1
call getQ(nra, nca, r, u, q)                  ! Восстанавливаем матрицу Q
c = matmul(transpose(q), b)
! Решаем систему линейных уравнений R(1)y(1) = c(1), где c(1) = c(1: kbasis)
call Lslrt(kbasis, r, nra, c(1:kbasis), 2, x(1:kbasis))
call permu(nca, x, ipvt, 1, x)                ! Перестановка элементов вектора x
res = b - matmul(a, x)                        ! Вычисляем невязки и выводим результат
call wrrrn('x', 1, nca, x, 1, 0); call wrrrn('res', 1, nra, res, 1, 0)
end program Least1
! Подпрограмма QRhous2 вычисляет QR-разложения с выбором главных столбцов
subroutine QRhous2(m, n, a, ipvt)
 real(4) :: a(m, n), u(m)
 integer(4) :: k, ipvt(n)
 ipvt = (/ (k, k = 1, n) /)                   ! Инициализация вектора перестановок
 do k = 1, n
  ! Текст подпрограмм hvec и rowHouse приведен в разд. 7.2.1.2 и 7.2.1.3
  if(k < n) call change( )                    ! Выполняем перестановки столбцов
  call hvec(m - k + 1, a(k:m, k), u(k:m))     ! Вычисляем нормированный вектор Хаусхолдера
  call rowHouse(m, n, k, a, u)                ! Обнуляем столбец k матрицы A
  a(k + 1:m, k) = u(k + 1:m)                  ! Запоминаем вектор Хаусхолдера в обнуляемой
 end do                                       ! части матрицы A
contains
 subroutine change( )                         ! Перестановки столбцов
  real(4) :: snrm(n + 1 - k), hold(m)
  integer(4) :: kma(1)
  snrm = sum(a(k:, k:)**2, dim = 1)           ! Квадраты 2-норм исследуемых столбцов
  kma = k + maxloc(snrm) - 1
  if(kma(1) /= k) then                        ! Проверяем, нужны ли перестановки
   ipvt(k) = kma(1); ipvt(kma(1)) = k         ! Изменяем вектор перестановок
   hold = a(:, k); a(:, k) = a(:, kma(1)); a(:, kma(1)) = hold
  end if
 end subroutine change
end subroutine QRhous2
```

Результат:

```
kbasis =   3
```

```
                    x
  0.636      2.845      1.058      0.000

                                   res
-0.733     0.996     -0.365     0.783     -1.353     -0.036     1.306     -0.597
```

9.1.4. Решение задачи наименьших квадратов с помощью сингулярного разложения

Выразим квадрат 2-нормы, минимум которой определяется, через сингулярное разложение:

$$\|b - Ax\|_2^2 = \left\|b - U\Sigma V^T x\right\|_2^2 = \left\|U^T(b - U\Sigma V^T x)\right\|_2^2 = \left\|U^T b - \Sigma V^T x\right\|_2^2 .$$

Преобразования верны, поскольку U^T является ортогональной матрицей и умножение вектора $b - U\Sigma V^T x$ на U^T не приведет к изменению его 2-нормы.

Введем обозначения $d = U^T b$ и $z = V^T x$. Тогда

$$\| b - Ax \|_2^2 = \| d - \Sigma z \|_2^2 = (d_1 - \sigma_1 z_1)^2 + (d_2 - \sigma_2 z_2)^2 + \ldots + (d_n - \sigma_n z_n)^2 + d_{n+1}^2 + \ldots + d_m^2,$$

где σ_i – сингулярное число, m и n – соответственно число строк и столбцов в матрице A.

Очевидно, что минимум квадрата 2-нормы достигается, когда

$$\sigma_i z_i = d_i,\ i = 1, \ldots, n.$$

При этом $x = Vz$.

Если все сингулярные числа отличны от нуля, задача наименьших квадратов имеет единственное решение. В противном случае, если хотя бы одно сингулярное число равно нулю ($\sigma_i = 0$), множество решений бесконечно, поскольку возможен произвольный выбор z_i. В таких случаях обычно принимают $z_i = 0$.

Сингулярные разложения используются в задаче наименьших квадратов для распознавания матриц, близких к вырожденным, в тех случаях, когда посредством QR-разложения такое распознавание выполнить не удается.

Пример. Решается задача наименьших квадратов с применением сингулярного разложения.

```
program Least2
 use msimsl
 integer(4), parameter :: nra = 8, nca = 4, Lda = nra
 integer(4) :: ipath, irank
 real(4) :: a(Lda, nca), u(Lda, nra), v(nca, nca), b(nra), d(nra), s(nca), z(nca), x(nca), res(nra), tol
 a = reshape((/      1.0,    5.0,   15.0,    5.0,              &
                     1.0,    4.0,   17.0,    4.0,              &
                     1.0,    7.0,   14.0,    7.0,              &
                     1.0,    3.0,   18.0,    3.0,              &
```

```
                    1.0,    1.0,   15.0,    1.0,          &
                    1.0,    8.0,   11.0,    8.0,          &
                    1.0,    3.0,    9.0,    3.0,          &
                    1.0,    4.0,   10.0,    4.0 /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
b = (/ 30.0, 31.0, 35.0, 29.0, 18.0, 35.0, 20.0, 22.0 /)
ipath = 11                          ! Вычисляем все сингулярные векторы
z = 0.0; tol = 10.0 * amach(4)      ! Допуск
! Вычисляем сингулярное разложение матрицы A
call Lsvrr(nra, nca, a, Lda, ipath, tol, irank, s, u, Lda, v, nca)
d = matmul(transpose(u), b)         ! Вектор d
where(s > tol) z = d(1:nca) / s     ! z_i = d_i / σ_i
x = matmul(v, z)                    ! x = Vz
print *, 'irank = ', irank          ! Вывод результатов
res = b - matmul(a, x)              ! Вычисляем невязки и выводим результат
call wrrrn('x', 1, nca, x, 1, 0); call wrrrn('res', 1, nra, res, 1, 0)
end program Least2
```

Результат:

```
irank =  3

                    x
 0.636     1.423     1.058     1.423

                                    res
-0.733     0.996    -0.365     0.783    -1.353    -0.036     1.306    -0.597
```

9.2. ПОДПРОГРАММЫ IMSL ДЛЯ РЕШЕНИЯ ПЕРЕОПРЕДЕЛЕННОЙ И НЕДООПРЕДЕЛЕННОЙ ПРОБЛЕМЫ $Ax \approx b$

9.2.1. Перечень подпрограмм

Перечень подпрограмм, решающих методом наименьших квадратов проблему $Ax \approx b$, приведен в табл. 9.1. При этом указывается лишь имя подпрограммы, работающей с одинарной точностью. Имена подпрограмм, принимающих данные двойной точности, и подпрограмм 2-го уровня не приводятся.

Таблица 9.1. Подпрограммы, решающие проблему $Ax \approx b$

Подпрограмма	*Назначение*
	Проблема $Ax \approx b$ без ограничений
LSBRR	Решает методом наименьших квадратов проблему $Ax \approx b$ с итерационным уточнением корней
LSQRR	Решает методом наименьших квадратов проблему $Ax \approx b$ без итерационного уточнения корней

LQRRV	Решает методом наименьших квадратов проблему $Ax \approx b$, используя блочное преобразование Хаусхолдера
LQRSL	Вычисляют преобразование координат и завершают решение проблемы $Ax \approx b$ методом наименьших квадратов, используя ранее найденное подпрограммой LQRRR (разд. 7.4.1) преобразование Хаусхолдера
Проблема $Ax \approx b$ с линейными ограничениями	
LCLSQ	Решает методом наименьших квадратов проблему $Ax \approx b$ с линейными ограничениями

9.2.2. Проблема $Ax \approx b$ без ограничений

9.2.2.1. Решение проблемы $Ax \approx b$ методом наименьших квадратов с итерационным уточнением корней

Выполняет подпрограмма LSBRR (DLSBRR):

CALL LSBRR(*nra*, *nca*, *a*, *Lda*, *b*, *tol*, *x*, *res*, *kbasis*)

Входные данные: *nra*, *nca*, *a*, *Lda*, *b*, *tol*.

Выходные данные: *x*, *res*, *kbasis*.

nra и *nca* – соответственно число строк и столбцов в матрице *A*.

a – вещественный массив формы (*Lda*, *nca*), содержащий *nra*×*nca*-матрицу коэффициентов *A* решаемой проблемы.

Lda – ведущий размер массива *a* по первому измерению.

b – вещественный вектор размера *nra*, содержащий правую часть системы линейных уравнений $Ax \approx b$.

tol – вещественный скаляр, содержащий неотрицательный допуск для определения подмножества столбцов матрицы *A*, включаемых в решение. Если *tol* = 0, используются все min(*nra*, *nca*) столбцов.

x – вещественный вектор размера *nca*, содержащий решение. Компоненты вектора, соответствующие неиспользуемым столбцам матрицы *A*, приравниваются нулю.

res – вещественный вектор размера *nra*, содержащий невязку $b - Ax$.

kbasis – целочисленный скаляр, содержащий число столбцов матрицы *A*, использованных для получения решения.

Автоматически для решения предоставляется память:

- *nra***nca* + 4*nca* + *nra* – 1 байт в случае LSBRR;
- 2*nra***nca* + 7*nca* + 2*nra* – 2 байт в случае DLSBRR.

Память можно выделить явно, применив L2BRR (DL2BRR):

CALL L2BRR(*nra*, *nca*, *a*, *Lda*, *b*, *tol*, *x*, *res*, *kbasis*, *qr*, *qraux*, *ipvt*, *wk*)

qr – вещественный вектор размера *nra***nca*, представляющий *nra*×*nca*-матрицу, содержащую *QR*-разложение матрицы *A*.

qraux – вещественный вектор размера *nca*, содержащий данные об ортогональной матрице Q из QR-разложения.

ipvt – целочисленный вектор размера *nca*, содержащий информацию о перестановках столбцов при выполнении QR-разложения матрицы A.

wk – вещественный вектор размера *nra* + 2*nca* - 1.

Подробнее параметры *qr*, *qraux* и *ipvt* описаны при рассмотрении в разд. 7.4.1 подпрограммы LQRRR.

Замечания:

1. Информационная ошибка типа 4 с кодом 1 возникает, если входная матрица плохо обусловлена, и по этой причине итерационное уточнение корней не может быть эффективным.
2. Для управления памятью используется целочисленное значение 16 в массиве *iopts* и задаются соответствующие значения в массиве *ivals*(1:4) подпрограммы IUMAG. Значение 17 массива *iopts* и установки в массиве *ivals*(1:2) позволяют активизировать или отключать режим вычисления оценки числа обусловленности матрицы A. Подробно процесс использования этих настроек описан при рассмотрении LSVCR (разд. 7.6.2).

Описание:

Подпрограмма LSBRR вычисляет QR-разложение матрицы A с перестановками столбцов и определяет первые *kbasis* = *k* используемых столбцов из условия $|R_{kk}| \le tol * |R_{11}|$. В действительности это условие означает, что берутся столбцы, для которых число обусловленности не превышает 1.0/*tol*. Затем LQRSL масштабирует первые *kbasis* столбцов матрицы A, выравнивая их значения. Используемый для итерационного уточнения решения алгоритм основан на работах [26, 27] и также описан в [6].

Пример. Методом наименьших квадратов решается переопределенная система линейных уравнений с 8×4-матрицей A. Причем второй и четвертый столбцы A одинаковы. Подпрограмма LSBRR определяет, что базис составляют 3 столбца.

```
program Least3
 use msimsl
 integer(4), parameter :: nra = 8, nca = 4, Lda = nra
 real(4) :: a(Lda, nca), b(nra), x(nca), res(nra), tol = 1.0e-4
 a = reshape((/      1.0,    5.0,   15.0,    5.0,          &
                     1.0,    4.0,   17.0,    4.0,          &
                     1.0,    7.0,   14.0,    7.0,          &
                     1.0,    3.0,   18.0,    3.0,          &
                     1.0,    1.0,   15.0,    1.0,          &
                     1.0,    8.0,   11.0,    8.0,          &
                     1.0,    3.0,    9.0,    3.0,          &
                     1.0,    4.0,   10.0,    4.0 /), shape = (/ Lda, nca /), order = (/ 2, 1 /))
```

```
b = (/ 30.0, 31.0, 35.0, 29.0, 18.0, 35.0, 20.0, 22.0 /);
! Решаем переопределенную систему линейных уравнений
call Lsbrr(nra, nca, a, Lda, b, tol, x, res, kbasis)
print *, 'kbasis = ', kbasis                    ! Вывод решения и невязок
call wrrrn('x', 1, nca, x, 1, 0); call wrrrn('res', 1, nra, res, 1, 0)
end program Least3
```

Результат:

```
kbasis =   3

                    x
  0.636      2.845      1.058      0.000

                                    res
 -0.733      0.996     -0.365      0.783     -1.353     -0.036      1.306     -0.597
```

9.2.2.2. Решение проблемы $Ax \approx b$ методом наименьших квадратов без итерационного уточнения корней

Выполняет подпрограмма LSQRR (DLSQRR):

CALL LSQRR(*nra*, *nca*, *a*, *Lda*, *b*, *tol*, *x*, *res*, *kbasis*)

Описание параметров дано при рассмотрении LSBRR (разд. 9.2.2.1).

Автоматически для решения предоставляется память:

- $(nra + 4)nca$ - 1 байт в случае LSQRR;
- $(2nra + 7)nca$ - 2 байт в случае DLSQRR.

Память можно выделить явно, применив L2QRR (DL2QRR):

CALL L2QRR(*nra*, *nca*, *a*, *Lda*, *b*, *tol*, *x*, *res*, *kbasis*, *qr*, *qraux*, *ipvt*, *work*)

Описание параметров *qr*, *qraux* и *ipvt* дано при рассмотрении LSBRR (разд. 9.2.2.1).

work – рабочий вектор размера $2nca$ - 1.

<u>*Замечания*</u>:

1. Параметр *tol* используется так же, как и в LSBRR.
2. Управление памятью и порядком вычисления оценки числа обусловленности матрицы осуществляется так же, как и при работе с LSBRR.

Описание:

Первоначально LSQRR вычисляет *QR*-разложение матрицы *A*. Разрешаются перестановки всех столбцов. Столбец *k* входит в базис, если $|R_{kk}| \le tol * |R_{11}|$. После выявления неиспользуемых столбцов задача решается подпрограммой LQRSL (разд. 9.2.2.4). Решение формируется с учетом выполненных перестановок. Компоненты решения, соответствующие столбцам, не включенным в базис, приравниваются нулю.

Пример. Рассмотрим задачу поиска по методу наименьших квадратов коэффициентов c_i функции

$$f(x) = c_0 + c_1x + c_2x^2$$

по следующим данным:

x	2.0	4.0	6.0	8.0
y	4.999	9.001	12.999	17.00

В исходных данных строка матрицы A содержит 1, x и x^2 в заданных точках.

```
program Least4
 use msimsl
 integer(4), parameter :: nra = 4, nca = 3, Lda = nra
 real(4) :: a(Lda, nca), b(nra), c(nca), res(nra), tol = 1.0e-4
 a = reshape((/          1.0,     2.0,     4.0,                  &
                         1.0,     4.0,    16.0,                  &
                         1.0,     6.0,    36.0,                  &
                         1.0,     8.0,    64.0 /),               &
                    shape = (/ Lda, nca /), order = (/ 2, 1 /))
 b = (/ 4.999, 9.001, 12.999, 17.001 /)
 ! Решаем переопределенную систему линейных уравнений
 call Lsqrr(nra, nca, a, Lda, b, tol, c, res, kbasis)
 print *, 'kbasis = ', kbasis                  ! Вывод результатов
 call wrrrn('c', 1, nca, c, 1, 0)              ! Решение задачи наименьших квадратов
 call wrrrn('res', 1, nra, res, 1, 0)          ! Невязки
end program Least4
```

Результат:

```
kbasis =   3

                    c
     0.999        2.000          0.000

                          res
  -0.000400    0.001199    -0.001199    0.000400
```

Таким образом, функция имеет вид

$$f(x) = 0.999 + 2x.$$

9.2.2.3. Решение проблемы $Ax \approx b$ методом наименьших квадратов с использованием блочных преобразований Хаусхолдера

Выполняет подпрограмма LQRRV (DLQRRV):

CALL LQRRV(*nra*, *nca*, *numexc*, *a*, *Lda*, *x*, *Ldx*)

Входные данные: *nra*, *nca*, *numexc*, *a*, *Lda*, *Ldx*.

Выходные данные: *x*.

nra и *nca* – соответственно число строк и столбцов в матрице A.

numexc – число правых частей.

a – вещественный массив формы (*Lda*, *nca* + *numexc*), содержащий матрицу коэффициентов *A* решаемой линейной проблемы и правые части системы, находящиеся в сечении *a*(:*nra*, *nca* + 1:*nca* + *numexc*).

Lda – ведущий размер массива *a* по первому измерению.

x – вещественный массив формы (*Ldx*, *numexc*), содержащий в каждом из своих столбцов решение для соответствующей правой части системы.

Ldx – ведущий размер массива *x*.

Автоматически для решения предоставляется память:

- *Lda*(*nca*+*numexc*)+(*nca*+*numexc*+1)(*nb*+1) байт в случае LQRRV;
- 2(*Lda*(*nca*+*numexc*)+(*nca*+*numexc*+1)(*nb*+1)) байт в случае DLQRRV, где *nb* – размер блока. По умолчанию *nb* = 1.

Память можно выделить явно, применив L2RRV (DL2RRV):

CALL L2RRV(*nra*, *nca*, *numexc*, *a*, *Lda*, *x*, *Ldx*, *fac*, *Ldfac*, *wk*)

fac – рабочий массив формы (*Ldfac*, *nca* + *numexc*), содержащий на выходе *QR*-разложение Хаусхолдера матрицы *A*. Если после вычислений массив *a* не нужен, то *a* и *fac* могут адресовать одну и ту же область памяти.

Ldfac – ведущий размер массива *fac*. Если *a* и *fac* адресуют одну и ту же область памяти, то необходимо, чтобы *Lda* = *Ldfac*.

wk – рабочий вектор размера (*nca* + *numexc* + 1)(*nb* + 1).

<u>*Замечание*</u>. Если входная матрица вырожденная, то возникнет информационная ошибка типа 4 с кодом 1.

Настройки подпрограммы L2RRV (LQRRV) можно менять, вызывая подпрограмму IUMAG, задавая в ее массиве *iopts* значения 5, 6, 10 и 11.

Значение *iopts*, равное 5, позволяет изменять размер блока, занося его в массив *ivals*. На некоторых компьютерах задание размера блока большим чем 1 приводит к повышению производительности. По умолчанию *ivals*(1:2) = = (/ 1, IMACH(5) /), где IMACH(5) возвращает 2147483647, равное наибольшему целому числу типа INTEGER(4).

Значение *iopts*, равное 10, позволяет регулировать шаг разложения. Если *ivals*(1) = 1, то выполняются преобразования Хаусхолдера. Если *ivals*(1) = 2, то разложение не вычисляется и оно должно быть выполнено ранее. По умолчанию *ivals*(1) = 1. Применяется с L2RRV.

Значение *iopts*, равное 11, позволяет задавать вид вычислений по правилам:

- вычисляются $b \leftarrow Q^T b$ и $x \leftarrow Rb$, если *ivals*(1) = 1;
- вычисляется $b \leftarrow Q^T b$, если *ivals*(1) = 2;
- вычисляется $b \leftarrow Qb$, если *ivals*(1) = 3;
- вычисляется $x \leftarrow Rb$, если *ivals*(1) = 4.

По умолчанию *ivals*(1) = 1. Причем *ivals*(1) = 2 или *ivals*(1) = 3 можно задавать, лишь работая с L2RRV (DL2RRV).

Описание:

Подпрограмма LQRRV вычисляет QR-разложение матрицы A, используя блочные преобразования Хаусхолдера [6]. Перестановки столбцов не выполняются. В процессе преобразований определяется ортогональная матрица Q и верхняя треугольная матрица R, такие, что $A = QR$. Результирующее QR-разложение $nra \times nca$-матрицы A определяется так:

1. Инициализация: $A1 \leftarrow A$.
2. С параметром $k = 1, \min(nra - 1, nca)$ выполнить:
   2.1. Вычислить преобразование Хаусхолдера для столбца k матрицы A_k:

   $$H_k = E - \tau_k u_k u_k^T,$$

   где u_k имеет нули в первых k - 1 позициях, а τ_k – вещественный скаляр.

   2.2. Вычислить $A_k \leftarrow H_k A_{k-1} = A_{k-1} - \tau_k u_k \left(A_{k-1}^T u_k\right)^T$.

конец цикла 2.

Таким образом, $A_p \approx H_p H_{p-1} \ldots H_1 A = Q^T A = R$, где $p = \min(nra - 1, nca)$.

Матрица Q не вычисляется в LQRRV непосредственно, но информация, необходимая для восстановления преобразования Хаусхолдера, сохраняется. Если матрица Q нужна явно, то Q^T можно определить в процессе разложения. Назначение LQRRV – предоставить высокопроизводительный QR-решатель по методу наименьших квадратов. Применяется для хорошо обусловленных проблем. Обоснование алгоритма приведено в [6]. Во время QR-разложения наиболее трудоемким является шаг $A_k \leftarrow H_k A_{k-1}$. Подпрограмма LQRRV формирует блок из nb преобразований Хаусхолдера, произведение которых записывается в виде

$$H_k H_{k+1} \ldots H_{k+nb-1} = E + YTY^T,$$

где Y – нижняя трапециевидная $nra \times nb$-матрица; T – верхняя треугольная $nb \times nb$-матрица. Оптимальный размер блока nb зависит от используемого типа компьютера.

Пример. Решается переопределенная ($m > n$) система $Ax \approx b$ с двумя правыми частями с вещественной $m \times n$-матрицей A. Решение ищется методом наименьших квадратов, т. е. минимизируется 2-норма:

$$\min_{x_j \subset R^n} \left\| Ax_j - b_j \right\|_2 .$$

Когда матрица A имеет полный столбцовый ранг, то для каждой правой части существует единственное, определяемое по методу наименьших квадратов решение.

```
program Least5
 use msimsl
 integer(4), parameter :: nca = 3, nra = 5, numexc = 2, Lda = nra, Ldx = nca
 real(4) :: x(Ldx, numexc), a(Lda, nca + numexc)
 ! Матрица A и (первые 3 столбца) и 2 правые части системы
 a = reshape((/      1.0,    2.0,    4.0,    7.0,   10.0,          &
                     1.0,    4.0,   16.0,   21.0,   10.0,          &
                     1.0,    6.0,   36.0,   43.0,    9.0,          &
                     1.0,    8.0,   64.0,   73.0,   10.0,          &
                     1.0,   10.0,  100.0,  111.0,   10.0 /),       &
                 shape = (/ Lda, nca + numexc /), order = (/ 2, 1 /))
 call Lqrrv(nra, nca, numexc, a, Lda, x, Ldx)          ! QR-разложение и решение системы
 call wrrrn('solutions 1-2', nca, numexc, x, Ldx, 0)   ! Вывод решений (solutions 1-2)
 ! Вычисление и вывод невязок (residuals 1-2)
 a(:, nca + 1:) = matmul(a(:, 1:nca), x) - a(:, nca + 1:)
 call wrrrn('residuals 1-2', nra, numexc, a(1, nca + 1), Lda, 0)
end program Least5
```

Результат:

```
        solutions 1-2
                 1          2
   1         1.000      10.80
   2         1.000      -0.43
   3         1.000       0.04

        residuals 1-2
                 1          2
   1        0.0000     0.0857
   2        0.0000     0.3429
   3        0.0000     0.5143
   4        0.0000     0.3429
   5        0.0000     0.0857
```

9.2.2.4. вычисление преобразований координат и завершение решения проблемы $Ax \approx b$ методом наименьших квадратов

Выполняет подпрограмма LQRSL (LDQRSL):

CALL LQRSL(*nra*, *kbasis*, *qr*, *Ldqr*, *qraux*, *b*, *ipath*, *qb*, *qtb*, *x*, *res*, *ax*)

Входные данные: *nra*, *kbasis*, *qr*, *Ldqr*, *qraux*, *b*, *ipath*.

Выходные данные: *qb*, *qtb*, *x*, *res*, *ax*.

nra – число строк в матрице A.

kbasis – число столбцов в подматрице A_k матрицы A; $kbasis \le \min(nra, nca)$, где *nca* – число столбцов в матрице A (*nca* является параметром LQRRR). Значение $kbasis < \min(nra, nca)$, если ранг матрицы меньше $\min(nra, nca)$. Пользователь должен следить за величиной *kbasis*.

qr – вещественный массив формы (*Ldqr*, *nca*), содержащий информацию о *QR*-разложении *nra*×*nca*-матрицы *A*. Поставляется подпрограммой LQRRR (разд. 7.4.1).

Ldqr – ведущий размер массива *qr*.

qraux – вещественный вектор размера *nca*, содержащий информацию об ортогональной части *QR*-разложения в первых min(*nra*, *nca*) позициях. Поставляется подпрограммой LQRRR (DLQRRR).

b – вещественный вектор размера *nra* – правая часть системы $Ax \approx b$.

ipath – целочисленный параметр, задающий вид вычислений; *ipath* имеет вид *ijklm*, где *i*, *j*, *k*, *l* и *m* – десятичные числа, принимающие значения 0 или 1. Причем

- $i \neq 0$ означает, что вычисляется Qb;
- $j \neq 0$ означает, что вычисляется Q^Tb;
- $k \neq 0$ означает, что вычисляются Q^Tb и x;
- $l \neq 0$ означает, что вычисляются Q^Tb и $b - Ax$;
- $m \neq 0$ означает, что вычисляются Q^Tb и Ax.

Например, если *ipath* = 01101, то $i = 0, j = 1, k = 1, l = 0$ и $m = 1$.

qb – вещественный вектор размера *nra*, содержащий произведение Qb, если его вычисление задано параметром *ipath*.

qtb – вещественный вектор размера *nra*, содержащий Q^Tb, если вычисление Q^Tb задано параметром *ipath*.

x – вещественный вектор размера *kbasis*, содержащий решение проблемы $A_kx = b$, если эти вычисления заданы параметром *ipath*. Если в LQRRR применены перестановки, то *j*-й элемент *x* соответствуют столбцу *ipvt*(*j*) исходной матрицы *A*.

res – вещественный вектор размера *nra*, содержащий невязки ($b - Ax$), если их вычисления заданы параметром *ipath*.

ax – вещественный вектор размера *nra*, содержащий произведение Ax, если его вычисление задано параметром *ipath*.

<u>Замечания</u>:

1. Если какая-либо задаваемая параметром *ipath* задача не решается, то размер соответствующего вектора принимается равным единице.
2. Если задано решение проблемы $A_kx = b$ и верхняя треугольная матрица *R* из *QR*-разложения вырожденная, то возникнет *информационная ошибка* типа 4 с кодом 1.
3. Подпрограмма LQRSL работает совместно с LQRRR, от которой LQRSL получает *qr*, *qraux* и *ipvt*. Подматрица A_k = (*A*(*ipvt*(1)), *A*(*ipvt*(2)), ..., *A*(*ipvt*(*kbasis*))), где *A*(*ipvt*(*i*)) – столбец *ipvt*(*i*) исходной матрицы.

Описание:

Подпрограмма LQRSL основана на процедуре SQRSL пакета LINPACK [34]. Применяется для решения методом наименьших квадратов проблемы $Ax \approx b$, а также вычислений, задаваемых параметром *ipath*.

Запишем проблему $Ax \approx b$ в виде нормальных уравнений

$$A^T Ax \approx A^T b.$$

Вычисляемое LQRRR QR-разложение матрицы A имеет вид $AP = QR$, где P – матрица перестановок ($P^{-1} = P^T$), Q – ортогональная матрица ($Q^{-1} = Q^T$) и R – верхняя трапециевидная матрица. Тогда

$$(PR^T)(Q^TQ)R(P^Tx) \approx (PR^T)Q^Tb.$$

Если матрица A^TA невырожденная, то R – также невырожденная и нормальные уравнения примут вид $R(P^Tx) \approx Q^Tb$. Подпрограмма LQRSL может быть вызвана для вычисления Q^Tb и затем нахождения P^Tx (т. е. решение возвращается с переставленными элементами).

Пример. Найти, применив метод наименьших квадратов, коэффициенты c_i функции

$$f(x) = c_0 + c_1x + c_2x^2 + c_3x^3$$

по следующим данным:

x	1.0	2.0	3.0	4.0	5.0
y	5.0	9.1	28.1	63.9	125.2

В исходных данных строка матрицы A содержит 1, x, x^2 и x^3 в заданных точках.

```
program Least6
 use msimsl
 integer(4), parameter :: nca = 4, nra = 5, kbasis = 4, Lda = nra, Ldqr = nra
 integer(4) :: ipvt(nca), i, j
 real(4) :: a(Lda, nca), qr(Ldqr, nca), qraux(nca), conorm(nca),                    &
          c(kbasis), qb(1), qtb(nra), res(nra), ax(1), y(nra), x(nra)
 logical(4) :: pivot
 x = (/ 1.0, 2.0, 3.0, 4.0, 5.0 /)
 do i = 1, nra                          ! Формируем массив a
  a(i, :) = (/ 1.0, (x(i)**j, j = 1, nca - 1) /)
 end do
 y = (/ 1.0, 9.1, 28.1, 64.9, 125.9 /)
 pivot = .true.; ipvt = 1               ! QR-разложение
 call Lqrrr(nra, nca, a, Lda, pivot, ipvt, qr, Ldqr, qraux, conorm)
 ipath = 00110                          ! Решение проблемы
 call Lqrsl(nra, kbasis, qr, Ldqr, qraux, y, ipath, qb, qtb, c, res, ax)
 call wrirn('ipvt', 1, nca, ipvt, 1, 0)  ! Вывод результата
 call wrrrn('c', 1, kbasis, c, 1, 0)
 call wrrrn('res', 1, nra, res, 1, 0)    ! Невязки
end program Least6
```

Результат:

```
                ipvt
  1          2          3          4

                          c
          1          2          3          4
     -2.700      3.745     -1.146      1.108

                          res
          1          2          3          4          5
   -0.00714    0.02857   -0.04286    0.02857   -0.00714
```

Таким образом, функция имеет вид

$f(x) = -2.7 + 3.74x + -1.146x^2 + 1.108x^3$.

Порядок найденных коэффициентов c_i определяется массивом *ipvt*. В нашем случае $c_i = ipvt(i)$, поскольку *ipvt* = (/ 1, 2, 3, 4 /). Если же, например, *ipvt* = (/ 4, 3, 2, 1 /), то $c_i = ipvt(5 - i)$.

9.3. Проблема $Ax \approx b$ с линейными ограничениями

Решение методом наименьших квадратов проблемы $Ax \approx b$ с линейными ограничениями находит подпрограмма LCLSQ (DLCLSQ):

CALL LCLSQ(*nra*, *nca*, *ncon*, *a*, *Lda*, *b*, *c*, *Ldc*, *bl*, *bu*, *irtype*, *xlb*, *xub*, *x*, *res*)

Входные данные: *nra*, *nca*, *ncon*, *a*, *Lda*, *b*, *c*, *Ldc*, *bl*, *bu*, *irtype*, *xlb*, *xub*.

Выходные данные: *x*, *res*.

nra – число уравнений.

nca – число переменных.

ncon – число ограничений.

a – вещественный массив формы (*Lda*, *nca*), содержащий *nra*×*nca*-матрицу коэффициентов.

Lda – ведущий размер массива *a*; *Lda* ≥ *nra*. Как правило, *Lda* = *nra*.

b – вещественный вектор размера *nra*, содержащий правую часть системы уравнений.

c – вещественный массив формы (*Ldc*, *nca*), содержащий коэффициенты *ncon* ограничений. Если *ncon* = 0, массив *c* не адресуется.

Ldc – ведущий размер массива *c*; *Ldc* ≥ *ncon*. Как правило, *Ldc* = *ncon*.

bl – вещественный вектор размера *ncon*, содержащий нижние границы общих ограничений. Если нижняя граница ограничения *i* отсутствует, то *bl*(*i*) не адресуется.

bu – вещественный вектор размера *ncon*, содержащий верхние границы общих ограничений. Если верхняя граница ограничения *i* отсутствует, то *bu*(*i*) не адресуется. Если диапазон ограничений не задается, то *bl* и *bu* могут быть одним массивом.

irtype – целый вектор размера *ncon*, задающий вид ограничений; значения *irtype*(*i*) = 0, 1, 2 и 3 означают для ограничения *i* соответственно .EQ., .LE., .GE. и диапазон ограничений.

xlb – вещественный вектор размера *nca*, содержащий нижние границы для переменных. Если для переменной *i* нижняя граница не устанавливается, то *xlb*(*i*) должен получить значение 1.0E+30.

xub – вещественный вектор размера *nca*, содержащий верхние границы для переменных. Если для переменной *i* верхняя граница не устанавливается, то *xub*(*i*) должен получить значение -1.0E+30.

x – вещественный вектор размера *nca*, содержащий приблизительное решение.

res – вещественный вектор размера *nra*, содержащий невязки $b - Ax$.

Возможные информационные ошибки при вызовах LCLSQ перечислены в табл. 9.2.

Таблица 9.2. Возможные информационные ошибки

Тип	*Код*	*Причина ошибки*
3	1	Относительный допуск меньше машинной точности
4	2	Заданные для переменных границы несовместимы
4	3	Заданные границы для ограничений несовместимы
4	4	Превышено максимально допустимое число итераций

Целочисленные установки подпрограммы LCLSQ можно менять, вызывая подпрограмму IUMAG, задавая в ее массиве *iopts* значения 13 и 14. Значение *iopts*, равное 13, позволяет регулировать объем выводимой информации. По умолчанию для этого значения *iopts* элемент *ivals*(1) содержит 0. Задание *ivals*(1) = 1 увеличит объем вывода. Значение *iopts*, равное 14, позволяет изменять максимальное число итераций. Если *ivals*(1) = 0 – это значение действует по умолчанию, то допустимо 5max(*nra*, *nca*) итераций. Новое число итераций устанавливается при *iopts* = 14 в *ivals*(1).

Вещественные установки LCLSQ можно менять, вызывая SUMAG, задавая в ее массиве *iopts* значения 2 и 5. Значение *iopts* = 2 позволяет изменять используемый при вычислениях относительный допуск. По умолчанию он равен SQRT(AMACH(4)), где AMACH(4) – машинная точность. Новый допуск устанавливается при *iopts* = 14 в *svals*(1). Значение *iopts* = 5 позволяет изменять используемый при вычислениях абсолютный допуск. По умолчанию он равен SQRT(AMACH(4)). Новый допуск устанавливается при *iopts* = 14 в *svals*(1).

Описание:

Подпрограмма LCLSQ решает методом наименьших квадратов проблему $Ax \approx b$ с линейными ограничениями вида:

$$b_l \le Cx \le b_u,$$

$$x_l \le x \le x_u.$$

Здесь A – матрица коэффициентов уравнения решаемой проблемы, а C – матрица коэффициентов ограничений. Векторы b_l, b_u x_l и x_u – соответственно нижние и верхние границы ограничений и переменных. Система решается путем определения переменных $y \equiv Cx$ и затем решения системы с нижними и верхними границами для x и y. Уравнение $Cx - y = 0$ является набором эквивалентных ограничений и решается методом штрафных функций [49, p. 826-834].

Пример употребления LCLSQ приведен в разд. 3.6.

10. ПРОЦЕДУРЫ И ОПЕРАЦИИ БИБЛИОТЕКИ IMSL FORTRAN 90 MP

10.1. ВВЕДЕНИЕ

Библиотека IMSL Fortran 90 MP версии 4.0 (далее – IMSL 90) доступна только пользователям CVF. Она содержит процедуры и операции, реализующие алгоритмы численных методов. В ней, в отличие от рассмотренной ранее математической библиотеки IMSL 77, широко используются нововведения стандарта Фортрана 90. Они позволяют, как правило, сделать вызовы процедур IMSL более компактными. Кроме того, применение родовых интерфейсов дает возможность использовать одно родовое имя или одну операцию для разных типов данных. Например, имя процедуры LIN_SOL_GEN является родовым. В то же время ее интерфейс содержит четыре специфических имени: S_LIN_SOL_GEN, D_LIN_SOL_GEN, C_LIN_SOL_GEN и Z_LIN_SOL_GEN, обеспечивающих обращение к процедурам, выполняющим вычисления с разными типами данных, – REAL(4), REAL(8), COMPLEX(4) и COMPLEX(8). Также благодаря наличию необязательных параметров одно и то же имя используется для решения разных задач. Например, процедура LIN_SOL_GEN решает линейную систему $Ax = b$ без итерационного уточнения корней, если применяется вызов

```
call Lin_sol_gen(a, b, x)
```

В случае вызова

```
call Lin_sol_gen(a, b, x, nrhs = 0, ainv = inv, det = determinant)
```

вычисляется обратная матрица и ее определитель.

Еще один достигаемый в результате перезагрузки эффект – это включение в IMSL 90 операций и функций, выполняющих те же действия, что и вызов подпрограммы по родовому или соответствующему специфическому имени. Например, решение линейной системы $Ax = b$ будет найдено после вызова решателя по родовому имени

```
call Lin_sol_gen(a, b, x)
```

и в результате применения операции .ix.:

```
x = a .ix. b                    ! Применяем операцию .ix.
```

Собственные значения матрицы A вернут подпрограмма LIN_EIG_SELF и функция EIG:

```
call Lin_eig_self(a, d)         ! Вычисляем собственные значения матрицы A
d = eig(a)                      ! Те же результаты вернет функции EIG
```

Другой особенностью IMSL 90 является использование в качестве формальных параметров массивов, перенимающих форму. Это позволяет исключить из списка фактических параметров протяженности передаваемых в процедуры массивов. В то же время необязательными параметрами, например такими, как *nrows* или *nrhs*, можно управлять размером сечений переданных в процедуру массивов.

Конечно же, и IMSL 77 и IMSL 90 реализуют одни и те же численные алгоритмы. Поэтому, выигрывая в компактности IMSL 90, не расширяет (за редким исключением) возможности специалиста, работающего на персональном компьютере. Более того, количество реализованных в ней методов несколько меньше, чем в IMSL 77. Так, IMSL 90 не содержит процедуры решения линейных систем с некоторыми видами матриц, например с персимметрической или циклической. Возможно, такие процедуры появятся в последующих версиях библиотеки.

Однако по мере внедрения многопроцессорных ЭВМ преимущества IMSL 90 станут очевидными, поскольку конструкции Фортрана 90, изначально ориентированные на параллельные вычислители, обеспечат (при наличии соответствующего компилятора) повышение быстродействия по сравнению с повсеместно распространенными ныне однопроцессорными компьютерами.

Замечание. Компиляторы FPS 4.0 и CVF 6.1 не создают код, пригодный для параллельных вычислений. Поэтому пользователю, имеющему многопроцессорный вычислитель и желающему воспользоваться заложенными в библиотеку IMSL 90 возможностями, придется позаботиться о приобретении другого компилятора.

В главе приводятся процедуры IMSL 90 для систем линейных уравнений, сингулярных разложений и вычисления собственных значений и собственных векторов, а также операции и функции библиотеки, заменяющие отдельные вызовы процедуры. Для сравнения программы, иллюстрирующие процедуры IMSL 90, снабжаются аналогичными вызовами процедур библиотеки IMSL 77.

Чтобы иметь возможность применять процедуры и операции IMSL 90, построителю надо указать имена библиотек imsl.lib, imsls_err.lib и imslmpistub.lib, расположенных в директории LIB папки IMSL. Для этого в среде CVF выполните цепочку действий: Project – Settings – выбрать закладку Link – в поле Obect/library modules дописать имена названных библиотек, используя в качестве разделителя пробел. В исходном тексте необходимо выполнить ссылку

```
use imslf90
```

которая содержит директивы и операторы:

```
module imslf90
 !dec$objcomment lib:'imsl.lib'
 !dec$objcomment lib:'imsls_err.lib'
 !dec$objcomment lib:'imslmpistub.lib'!dec$objcomment lib:'imsl.lib'
 !dec$objcomment lib:'imsls_err.lib'
 !dec$objcomment lib:'imslmpistub.lib'
 use build_error_structure_int
 use error_option_packet
 use error_post_int
 use fast_2dft_int
 use fast_3dft_int
 use fast_dft_int
 use Lin_eig_gen_int
 use Lin_geig_gen_int
 use Lin_sol_gen_int
 use Lin_sol_lsq_int
 use Lin_sol_svd_int
 use Lin_sol_tri_int
 use Lin_svd_int
 use Linear_operators
 use mp_types
 use nrm2_int
 use rand_gen_int
 use show_int
 use sort_real_int
 use spline_fitting_int
 use surface_fitting_int
 use pblsq_int
 use pnlsq_int
 use pde_1d_mg_int
end module
```

Можно снизить число просматриваемых компилятором модулей, включая в программу лишь отдельные ссылки приведенного выше модуля imslf90.

Вызов процедур двух библиотек (IMSL 77 и IMSL 90) станет возможным, если в программе есть ссылка

```
use imsl
```

содержащая операторы

```
module imsl
 use Numerical_Libraries
 use imslf90
end module
```

При описании процедур IMSL 90, которые, обладая родовым интерфейсом, работают как с вещественными, так и с комплексными матрицами, вводится понятие *самоприсоединенной матрицы*, обобщающее привычные понятия симметрической и эрмитовой матриц. Иными словами, в случае вещественных данных самоприсоединенной является симметрическая матрица, а в случае комплексных – эрмитова.

10.2. ОПЕРАЦИИ БИБЛИОТЕКИ IMSL 90

Определенные в библиотеке IMSL 90 операции приведены в табл. 10.1, в которой имена A и B использованы для обозначения матриц, имена x и b – для обозначения векторов. В последней графе таблицы приводятся либо выполняющие аналогичные действия встроенные функции Фортрана, либо подпрограммы IMSL 90.

Таблица 10.1. Операции библиотеки IMSL 90

Операции IMSL 90	*Матричные операции*	*Процедуры-аналоги*
A .x. B	AB	MATMUL(A, B)
.i. A	A^{-1}	LIN_SOL_GEN, LIN_SOL_LSQ
.t. A .h. A	A^T A^H	TRANSPOSE(A) CONJG(TRANSPOSE(A))
$x = A$ .ix. b	$A^{-1}b$	LIN_SOL_GEN, LIN_SOL_LSQ
$x = b$ .xi. A	bA^{-1}	LIN_SOL_GEN, LIN_SOL_LSQ
A .tx. B или (.t. A) .x. B A .hx. B или (.h. A) .x. B	A^TB A^HB	MATMUL(TRANSPOSE(A), B) MATMUL(CONJG(TRANSPOSE(A)), B)
B .xt. A или B .x. (.t. A) B .xh. A или B .x. (.h. A)	BA^T BA^H	MATMUL(B, TRANSPOSE(A)) MATMUL(B, CONJG(TRANSPOSE(A)))

<u>*Замечания*</u>:

1. В случае решения линейной системы с несколькими правыми частями в операциях .ix. и .xi. на месте векторов x и b должны стоять двумерные массивы (см. разд. 10.4.1 и 10.6.1).
2. Операции .h., .hx. и .xh. предназначены для работы с эрмитовыми матрицами.

Ссылки на все необходимые для использования операций и функций IMSL 90 модули содержит модуль Linear_operators:

```
module Linear_operators

! Интерфейсы операций IMSL 90
use operation_i
use operation_ix
```

```
use operation_t
use operation_h
use operation_tx
use operation_hx
use operation_x
use operation_xi
use operation_xt
use operation_xh
! Интерфейсы функций IMSL 90
use cond_int
use det_int
use diag_int
use diagonals_int
use fft_int
use ifft_int
use fft_box_int
use ifft_box_int
use eye_int
use Lin_eig_self_int
use Lin_sol_self_int
use norm_int
use orth_int
use rand_int
use rank_int
use svd_int
use unit_int
use eig_int
use chol_int
use isnan_int
end module
```

Поэтому операции можно применять, если программа имеет ссылку

```
use Linear_operators
```

или ссылку на модули, содержащие интерфейсы соответствующих операций. Например, для употребления операции .x. необходимо включить ссылку

```
use operation_x
```

10.3. ФУНКЦИИ БИБЛИОТЕКИ IMSL 90

Библиотека IMSL 90 содержит приведенные в табл. 10.2 функции, которые наряду с приведенными выше операторами используются для выполнения матричных вычислений и в некоторых случаях могут заменить вызовы соответствующих подпрограмм библиотеки. Необязательные параметры функций указаны в квадратных скобках.

Таблица 10.2. Функции библиотеки IMSL 90 для линейных систем

Функция	*Что возвращает*
R = CHOL(A)	Матрицу R из R^TR-разложения Холецкого положительно определенной самоприсоединенной матрицы A
Q = ORTH(A, [R = R])	Матрицы Q и R из QR-разложения матрицы A
U = UNIT(A)	Матрицу с нормализованными столбцами, имеющими евклидову длину, равную единице. Нормализуются массивы рангов 2 и 3 или каждое двуранговое сечение массива
d = DET(A)	Определитель матрицы A (также возвращается подпрограммой LIN_SOL_GEN)
k = RANK(A)	Ранг матрицы A
p = NORM(A[, [*type* =] i])	В зависимости от значения параметра *type* возвращает 1-, 2- или ∞-норму матрицы или вектора
c = COND(A)	Число обусловленности матрицы A, равное отношению наибольшего и наименьшего сингулярных значений матрицы или HUGE(A), если последняя величина меньше названного отношения
Z = EYE(n)	Единичную $n \times n$-матрицу Z
A = DIAG(x)	Диагональную матрицу, содержащую на диагонали элементы вектора x
x = DIAGONALS(A)	Вектор x, содержащий диагональные элементы матрицы A
A = RAND(A)	Матрицу A, заполненную случайными числами ($0.0 \le a_{ij} < 1.0$)
l = isNaN(A)	.TRUE., если в матрице A есть не число, и .FALSE. – в противном случае. Параметром функции может быть и скаляр
s = SVD(A, [u = U, v = V])	Матрицы U, V и Σ (в виде вектора s) сингулярного разложения матрицы $A = U\Sigma V^T$ (также возвращаются подпрограммой LIN_SVD)
e = EIG(A, [b = B, d = d, v = V, w = W])	Собственные значения (вектор e) и собственные векторы (матрица V) матрицы A; также возвращаются подпрограммой LIN_EIG_SELF в случае самоприсоединенной матрицы, LIN_EIG_GEN – в случае матрицы общего вида и LIN_GEIG_GEN – в случае обобщенной собственной проблемы $Av = \lambda Bv$, с которой, в частности, применяются параметры $b = B$, $d = d$ и $w = W$ (см. разд. 10.5.4)

Обращение к приведенным в таблице функциям станет возможным, если в программе есть ссылка

```
use Linear_operators
```

или ссылка на модуль, содержащий интерфейс соответствующей функции. Например, для употребления функции COND необходимо иметь ссылку

```
use cond_int
```

10.4. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

10.4.1. Процедура LIN_SOL_GEN для линейных систем с матрицей общего вида

Система линейных уравнений $Ax = b$ с полной общего вида матрицей A и связанные с ее решением дополнительные вычисления выполняются процедурой LIN_SOL_GEN. Дополнительные задачи включают:

- определение LU-разложения матрицы A с частичным выбором ведущего элемента;
- вычисление определителя матрицы A;
- нахождение обратной матрицы A^{-1};
- решение $A^Tx = b$ или $Ax = b$ по ранее найденному LU-разложению A. Система может иметь несколько правых частей.

Обязательные параметры LIN_SOL_GEN.
Входные/выходные параметры: *a*, *b*.
Выходные параметры: *x*.

a – массив формы (*n*, *n*), содержащий *n*×*n*-матрицу *A*.

b – массив формы (*n*, *nb*), содержащий *n*×*nb*-матрицу с правыми частями. Если *nb* > 1, то решается система с несколькими правыми частями.

x – - массив формы (*n*, *nb*), содержащий *n*×*nb*-матрицу решений.

Необязательные параметры LIN_SOL_GEN.
Входные параметры: *nrows*, *nrhs*, *iopt*.
Входные/выхоые параметры: *pivots*.
Выходные параметры: *det*, *ainv*.

nrows = *n* – используется сечение входного массива *a*(1:*n*, 1:*n*). По умолчанию *n* = SIZE(*a*, 1).

nrhs = *nb* – используется сечение входного массива *b*(1:*n*, 1:*nb*). По умолчанию *nb* = SIZE(*b*, 2).

pivots = *pivots*(:) – целочисленный массив размера *n*, содержащий данные о перестановках строк.

det = *det*(1:2) – вектор размера 2, представляющий определитель матрицы *A*. Тип и разновидность типа *det* такие же, как у массива *a*. Если матрица *A* невырожденная, то определитель равен ABS(*det*(1))**det(2) * (*det*(1)/ABS(*det*(1))), причем ABS(*det*(1)) = RADIX(*det*). В противном случае *det*(1) = 0.0 и *det*(2) = = -HUGE(ABS(*det*(1))).

ainv = *ainv*(:, :) – массив, содержащий A^{-1} (когда A невырожденная матрица).

iopt = *iopt*(:) – массив производного типа приведенных в табл. 10.3 необязательных опций LIN_SOL_GEN. Тип массива – *s_options*.

Таблица 10.3. Опции LIN_SOL_GEN

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_sol_gen_set_small	1
То же	lin_sol_gen_save_LU	2
"	lin_sol_gen_solve_A	3
"	lin_sol_gen_solve_ADJ	4
"	lin_sol_gen_no_pivoting	5
"	lin_sol_gen_scan_for_NaN	6
"	lin_sol_gen_no_sing_mess	7
"	lin_sol_gen_A_is_sparse	8

iopt(IO) = ?_options(?_lin_sol_gen_set_small, *small*) – задает значение *small*, на которое замещается диагональный элемент матрицы U, если он меньше величины *small*. При этом используется знак диагонального элемента, если он вещественный, или направление, если – комплексный. Система объявляется вырожденной. Решение, если не возникает переполнения, вычисляется с измененной матрицей. Значение по умолчанию – наименьшее число, для которого можно вычислить обратную величину.

iopt(IO) = ?_options(?_lin_sol_gen_set_save_LU, ?_dummy) – LU-разложение матрицы A будет сохраняться на месте массива a. Если затем подпрограмма применяется для решения линейных систем с той же матрицей, то необходимо задавать параметр *pivots* = *pivots*(:). Для повышения быстродействия на месте диагональных элементов матрицы A запоминаются обратные величины диагональных элементов матрицы U.

iopt(IO) = ?_options(?_lin_sol_gen_solve_A, ?_dummy) – ранее вычисленное LU-разложение матрицы A будет использовано для решения линейной системы $Ax = b$.

iopt(IO) = ?_options(?_lin_sol_gen_solve_ADJ, ?_dummy) – ранее вычисленное LU-разложение матрицы A будет использовано для решения линейной системы $A^T x = b$.

iopt(IO) = ?_options(?_lin_sol_gen_no_pivoting, ?_dummy) – перестановки строк выполняться не будут. Если *pivots* = *pivots*(:) присутствует в списке фактических параметров, то на выходе *pivots*(i) = i, для $i = 1, ..., n$.

iopt(IO) = ?_options(?_lin_sol_gen_scan_for_NaN, ?_dummy) – проверяет входные массивы a и b на наличие в них NaN. По умолчанию проверка не выполняется.

iopt(IO) = ?_options(?_lin_sol_gen_no_sing_mess, ?_dummy) – отключается вывод сообщений о том, что матрица *A* вырожденная.

iopt(IO) = ?_options(?_lin_sol_gen_A_is_sparse, ?_dummy) – задает выполнение более эффективного (в случае разреженной матрицы) цикла вычисления *LU*-разложения *A*. Матрица, однако, вводится как полная.

Описание:

Для решения линейной системы $Ax = b$ процедура LIN_SOL_GEN использует метод исключений Гаусса с частичным выбором ведущего элемента (разд. 5.2.3.1).

Пример 1. Решается система линейных уравнений с матрицей общего вида. Матрицы *A* и *b* заполняются случайными числами. Для сравнения приводятся выполняющие те же вычисления процедура LSLRG из IMSL 77 и операция .ix. библиотеки IMSL 90.

```
program sol_gen1
 use Lin_sol_gen_int; use rand_gen_int
 use text_transfer                    ! Для вывода русского текста
 use Numerical_Libraries              ! Для вызова процедур библиотеки IMSL 77
 use Linear_operators                 ! Для операции .ix.
 implicit none
 integer(kind(1)), parameter :: n = 4 ! Порядок системы
 real(kind(1e0)), parameter :: one = 1.0
 real(kind(1e0)) :: err
 real(kind(1e0)) :: a(n, n), b(n, n), x(n, n), res(n, n), y(n * n)
 integer(kind(1)) :: i
 call rand_gen(y)                     ! Генерируем вектор случайных чисел
 a = reshape(y, (/ n, n /))           ! и формируем матрицу A
 call rand_gen(y)                     ! Формируем n правых частей
 b = reshape(y, (/ n, n /))
 call Lin_sol_gen(a, b, x)            ! Решаем линейную систему Ax = b
 ! Проверка точности решения по величине невязок
 res = b - matmul(a, x)               ! Невязки
 err = maxval(abs(res)) / sum(abs(a) + abs(b))
 if(err <= sqrt(epsilon(one))) then
  print *, trim(ru_doswin('Удачное завершение примера 1 для LIN_SOL_GEN', .false.))
  call wrrrn('x', n, n, x, n, 0)      ! Вывод результата
 end if
 x = 0.0
 do i = 1, n                          ! Используем процедуру библиотеки IMSL 77
   call Lslrg(n, a, n, b(:, i), 1, x(:, i))  ! Решаем линейную систему Ax = b
 end do
 x = 0.0
 ! Используем операцию .ix. библиотеки IMSL 90. Имеем тот же результат
 x = a .ix. b
end program sol_gen1
```

Пример 2. Вычисляются обратная матрица и определитель матрицы A. Результат верен, если $AA^{-1} = I$ и $\det(A^{-1}) = \det(A)^{-1}$. Аналогичные действия выполняются процедурами IMSL 77, а затем операцией .i. и функцией DET библиотеки IMSL 90.

```
program sol_gen2
 use Lin_sol_gen_int; use rand_gen_int
 use text_transfer                          ! Для вывода русского текста
 use Numerical_Libraries, nouse => fac      ! Для вызова процедур библиотеки IMSL 77
 use Linear_operators                       ! Для операции .i. и функции DET
 implicit none
 integer(4), parameter :: n = 5
 real(kind(1e0)), parameter :: one = 1.0, zero = 0.0
 real(kind(1e0)) err
 real(kind(1e0)) a(n, n), b(n, 0), inv(n, n), x(n, 0), res(n, n), &
                 y(n * n), determinant(2), inv_determinant(2), det1, det2, det_a, det_i
 integer(4) :: i, ipvt(n)
 call rand_gen(y)                           ! Генерация матрицы A
 a = reshape(y, (/ n, n /))
 ! Вычисляем обратную матрицу и определитель матрицы A
 call Lin_sol_gen(a, b, x, nrhs = 0, ainv = inv, det = determinant)
 ! Вычисляем определитель обратной матрицы
 call Lin_sol_gen(inv, b, x, nrhs = 0, det = inv_determinant)
 res = matmul(a, inv)
 do i=1, n
  res(i, i) = res(i, i) - one               ! Невязки AA^-1 - I
 end do
 err = sum(abs(res)) / sum(abs(a))          ! Относительная ошибка
 if(err <= sqrt(epsilon(one))) then
  if(determinant(1) == inv_determinant(1) .and.                          &
    (abs(determinant(2) + inv_determinant(2)) <= abs(determinant(2)) * sqrt(epsilon(one)))) then
   print *, trim(ru_doswin('Удачное завершение примера 2 для LIN_SOL_GEN', .false.))
   call wrrrn('inv', n, n, inv, n, 0)       ! Вывод A^-1
   ! Вычисляем и выводим определитель матрицы A^-1
   print *, abs(inv_determinant(1))**inv_determinant(2) *                &
          (inv_determinant(1) / abs(inv_determinant(1)))
  end if
 end if
 inv = 0.0
 ! Те же вычисления процедурами библиотеки IMSL 77
 call Linrg(n, a, n, inv, n)                ! Вычисляем A^-1
 call wrrrn('inv', n, n, inv, n, 0)         ! Вывод A^-1
 call Lftrg(n, inv, n, inv, n, ipvt)        ! Находим LU-разложение матрицы A^-1
 call Lfdrg(n, inv, n, ipvt, det1, det2)    ! Вычисляем определитель (det1 и det2)
 print *, det1 * 10**det2                   ! Выводим определитель матрицы A^-1
```

```
inv = 0.0
! Используем операцию .i. и функцию DET библиотеки IMSL 90
det_a = det(a)                          ! Определитель матрицы A
inv = .i. a                             ! Находим A-1
det_i = det(inv)                        ! Определитель матрицы A-1
print *, det_i, 1.0 / det_a
end program sol_gen2
```

Пример 3. Решается линейная система $Ax = b$ с итерационным уточнением корней. Первоначально вычисляется LU-разложение матрицы A, имеющей одинарную точность. Затем разложение используется в цикле, осуществляющем итерационное уточнение решения. Используется приведенный в разд. 5.2.7.2 алгоритм. Ошибка снижается за счет применения при вычислении невязок двойной точности. Инициализация массива *iopti*(1:2) = *s_option* (0, 0.0) обеспечивает вычисление LU-разложения матрицы A. Оно возвращается массивом a и вектором *ipivots*. Затем настройки меняются так, что LU-разложение не вычисляется, а используется имеющееся. Та же система решается процедурой DLSARG из IMSL 77.

```
program sol_gen3
 use Lin_sol_gen_int; use rand_gen_int
 use text_transfer                      ! Для вывода русского текста
 use Numerical_Libraries                ! Для вызова процедур библиотеки IMSL 77
 implicit none
 integer(4), parameter :: n = 5, itmax = 100
 real(kind(1e0)), parameter :: one = 1.0, zero = 0.0
 real(kind(1d0)), parameter :: d_zero=0.0d0
 integer(4) :: ipivots(n), k
 real(kind(1e0)) :: a(n, n), b(n, 1), x(n, 1), w(n * n)
 real(kind(1e0)) :: change_new, change_old
 real(kind(1d0)) :: c(n, 1), d(n, n), y(n, 1)
 type(s_options) :: iopti(2) = s_options(0, zero)
 call rand_gen(w)                       ! Генерируем матрицу A
 a = reshape(w, (/ n, n /))
 call rand_gen(b(1:n,1))                ! Задаем правую часть линейной системы
 d = a; c = b                           ! Сохраняем данные в массивах двойной точности
 y = d_zero                             ! Начальное значение y
 change_old = huge(one)
 ! Задаем режим сохранения LU-разложения матрицы A
 iopti(1) = s_options(s_lin_sol_gen_save_LU, zero)
 k = 0                                  ! Число итераций
 do
  ! Вектор невязок. Вычисляем, используя двойную точность
  b = c - matmul(d, y)
  ! Решаем линейную систему. На первой итерации сохраняем, а на последующих
  ! используем LU-разложение матрицы A
```

```
 call Lin_sol_gen(a, b, x, pivots = ipivots, iopt = iopti)
 y = y + x
 change_new = sum(abs(x))
 k = k + 1
 ! Выходим из цикла, когда прекращается снижение изменений
 ! или превышено допустимое число итераций
 if(change_new >= change_old .or. k > itmax) exit
 change_old = change_new
 ! Задаем режим использования ранее найденного LU-разложения матрицы A
 ! LU-разложение вычисляться не будет
 iopti(2) = s_options(s_lin_sol_gen_solve_A, zero)
end do
print *, trim(ru_doswin('Удачное завершение примера 3 для LIN_SOL_GEN', .false.))
call dwrrrn('y', 1, n, y, 1, 0)                ! Вывод результата
! Те же вычисления процедурой DLSARG библиотеки IMSL 77
! Решаем линейную систему Ax = b и выполняем итерационное уточнение корней
call dlsarg(n, d, n, c, 1, y)
call dwrrrn('y', 1, n, y, 1, 0)                ! Вывод результата DLSARG
end program sol_gen3
```

Пример 4. Выполняется оценка матричной экспоненты. Решается дифференциальное уравнение

$$\frac{dy}{dt} = Ay$$

с начальным значением $y(0) = y_0$. В примере матрица A вещественная и не зависит от t. Решением является

$$y(t) = e^{At}y_0.$$

Для поиска решения вычисляются собственные значения (D) и собственные векторы (V) матрицы A, что позволяет представить A в виде произведения

$$A = VDV^{-1},$$

в котором D – диагональная матрица. Это дает возможность найти решение из равенства

$$y(t) = Ve^{Dt}z_0,$$

где

$$z_0 = V^{-1}y_0$$

возвращается LIN_SOL_GEN в результате решения линейной системы $Vz_0 = y_0$ с комплексной матрицей V. Значения $y(t)$ вещественные, однако для промежуточных вычислений используются комплексные величины.

Собственные значения и собственные векторы возвращаются подпрограммой EVCRG библиотеки IMSL 77. Вместо EVCRG можно употребить функцию EIG из библиотеки IMSL 90.

```
program sol_gen4
```

```
use Lin_sol_gen_int; use rand_gen_int
use Numerical_Libraries                    ! Для вызова процедур IMSL 77
use text_transfer                          ! Для вывода русского текста
use Linear_operators                       ! Для функции EIG
implicit none
integer(4), parameter :: n = 32, k = 128
real(kind(1e0)), parameter :: one = 1.0, t_max = 1.0, delta_t = t_max / (k - 1)
real(kind(1e0)) :: err, a(n, n), ytemp(n * n)
real(kind(1e0)) :: t(k), y(n, k), y_prime(n, k)
complex(kind(1e0)) :: e(n), v(n, n), z_0(n, 1), y_0(n, 1)
integer(4) :: i
call rand_gen(ytemp)                       ! Генерация матрицы A
a = reshape(ytemp, (/ n, n /))
! Вычисляет все собственные значения и собственные векторы
! вещественной несимметрической матрицы A
call evcrg(n, a, n, e, v, n)
! Тот же результат получается, если вместо EVCRG вызвать EIG
! e = eig(a, w = v)
! Случайные начальные значения для обыкновенного дифференциального уравнения
call rand_gen(ytemp(1:n))
y_0(1:n, 1) = ytemp(1:n)
! Решаем систему Vz_0 = y_0 с комплексной матрицей V
call Lin_sol_gen(v, y_0, z_0)
t = (/ (i * delta_t, i = 0, k - 1) /)       ! Задаем вектор t
! Вычисляем y и y' при разных значениях t(1:k)
y = matmul(v, exp(spread(e, 2, k) * spread(t, 1, n)) * spread(z_0(1:n, 1), 2, k))
call wrrrn('y', 1, n, y, 1, 0)             ! Вывод результата
y_prime = matmul(v, spread(e, 2, k) *                                  &
                 exp(spread(e, 2, k) * spread(t, 1, n)) * spread(z_0(1:n, 1), 2, k))
! Проверяем равенство y' - Ay = 0
err = sum(abs(y_prime - matmul(a, y))) / (sum(abs(a)) * sum(abs(y)))
if(err <= sqrt(epsilon(one))) then
 print *, trim(ru_doswin('Удачное завершение примера 4 для LIN_SOL_GEN', .false.))
end if
end program sol_gen4
```

10.4.2. Процедура LIN_SOL_SELF для линейных систем с самоприсоединенной матрицей

Процедура LIN_SOL_SELF решает линейную систему $Ax = b$ с самоприсоединенной (симметрической или эрмитовой) матрицей A. Система может иметь несколько правых частей. Дополнительно, задавая необязательные параметры, можно найти:

- разложение матрицы A с симметричным выбором диагонального элемента;
- определитель матрицы A;

- матрицу A^{-1};
- решение $Ax = b$ по ранее найденному разложению A.

Также, если матрица является положительно определенной, необязательным параметром можно задать разложение матрица A по Холецкому.

Обязательные параметры LIN_SOL_ SELF.
Входные/выходные параметры: *a*, *b*.
Выходной параметры: *x*.
a – массив формы (*n*, *n*), содержащий самоприсоединенную *n*×*n*-матрицу *A*.
b – массив формы (*n*, *nb*), содержащий *n*×*nb*-матрицу с правыми частями.
x – массив формы (*n*, *nb*), содержащий *n*×*nb*-матрицу решений.

Необязательные параметры LIN_SOL_SELF.
Входные параметры: *nrows*, *nrhs*, *iopt*.
Входные/выходные параметры:*pivots*.
Выходные параметры: *det*, *ainv*.

Параметры *nrows* = *n*, *nrhs* = *nb*, *det* = *det*(1:2) и *ainv* = *ainv*(:, :) имеют то же описания, что и одноименные параметры рассмотренной в предшествующем разделе процедуры LIN_SOL_GEN.

pivots = *pivots*(:) – целочисленный массив размера *n* + 1, содержащий в первых *n* позициях данные о перестановках строк. Позиция *n* + 1 содержит номер первого диагонального элемента, не превышающего по величине определяемого ниже числа *small*.

iopt = *iopt*(:) – массив производного типа приведенных в табл. 10.4 необязательных опций LIN_SOL_SELF. Тип массива – *s_options*.

Таблица 10.4. Опции LIN_SOL_SELF

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_sol_self_set_small	1
То же	lin_sol_self_save_factors	2
"	lin_sol_self_no_pivoting	3
"	lin_sol_self_use_Cholesky	4
"	lin_sol_self_solve_A	5
"	lin_sol_self_scan_for_NaN	6
"	lin_sol_self_no_sing_mess	7

iopt(IO) = ?_options(?_lin_sol_self_set_small, *small*) – задает значение *small*, на которое замещается диагональный элемент матрицы *U*, если он меньше величины *small*. Система при этом объявляется вырожденной. Матрица *U* появляется при использовании метода Аазена, предполагающего решение трехдиагональной системы $Tu = v$ с вычислением *LU*-разложения с частичным выбором ведущего элемента. Когда же используется разложение

Холецкого, то *small* заменяет диагональный элемент вычисляемой верхней треугольной матрицы R, когда он меньше *small*. Значение по умолчанию – наименьшее число, для которого можно вычислить обратную величину.

iopt(IO) = ?_options(?_lin_sol_self_save_factors, ?_dummy) – разложение матрицы A сохраняется на месте массива a. Если затем подпрограмма будет использоваться для решения линейных систем с той же матрицей, то необходимо задавать параметр *pivots* = *pivots*(:). Для повышения быстродействия, когда применяется метод Холецкого, на месте диагональных элементов матрицы A запоминаются обратные величины диагональных элементов матрицы R.

iopt(IO) = ?_options(?_lin_sol_self_no_pivoting, ?_dummy) – перестановки строк выполняться не будут. Если *pivots* = *pivots*(:) присутствует в списке фактических параметров, то на выходе в случае метода Аазена $pivots(i) = i + 1$, для $i = 1, ..., n - 1$, в случае метода Холецого $pivots(i) = i$, для $i = 1, ..., n$.

iopt(IO) = ?_options(?_lin_sol_self_use_Cholesky, ?_dummy) – вычисляется $PAP^T = R^TR$-разложение Холецкого вместо разложения по методу Аазена.

iopt(IO) = ?_options(?_lin_sol_self_solve_A, ?_dummy) – ранее вычисленное разложение матрицы A будет использовано для решения линейной системы $Ax = b$.

iopt(IO) = ?_options(?_lin_sol_self_scan_for_NaN, ?_dummy) – проверяет входные массивы a и b на наличие в них NaN. По умолчанию проверка не выполняется.

iopt(IO) = ?_options(?_lin_sol_self_no_sing_mess, ?_dummy) – отключается вывод сообщений о том, что матрица A вырожденная.

Описание:

Процедура LIN_SOL_SELF решает линейную систему с невырожденной самоприсоединенной матрицей. По умолчанию для ее разложения используется метод Аазена. Его вычислительная сложность $n^3/3$. Разложение имеет вид:

$$PAP^T = LTL^T,$$

где P – матрица перестановок; L – нижняя унитреугольная матрица; T – трехдиагональная самоприсоединенная матрица, т. е. решение исходной системы сводится к решению более простых систем $u = L^{-1}Pb$, $Tv = u$ и $x = P^TL^{-T}v$. Детали алгоритма см. в [6].

Когда матрица является самоприсоединенной положительно определенной, следует задать метод Холецкого, вычисляющий разложение $PAP^T = R^TR$, в котором R – верхняя треугольная матрица. Решение исходной задачи сводится к решению систем $u = R^{-T}Pb$ и $x = P^TL^{-1}u$.

О назначении перестановок см. разд. 5.3.3.3.

Пример. Методом наименьших квадратов с использованием нормальных уравнений решается переопределенная система линейных уравнений $Cx \approx d$,

где C – вещественная $m \times n$-матрица ($m \geq n$). Для решения формируется симметрическая матрица $A = C^T C$ и правая часть $b = A^T d$. Затем относительно x решается система $Ax = b$. Заметим, что с позиции точности результата приведенный метод хуже, чем используемый в LIN_SOL_LSQ для решения переопределенных систем линейных уравнений $Cx \approx d$. Для сравнения результат получается при помощи процедуры LSLDS из IMSL 77 и затем операцией .ix. библиотеки IMSL 90. Описание метода нормальных уравнений см. в [6, разд. 5.3.3].

```
program sol_self
 use lin_sol_self_int; use rand_gen_int
 use Numerical_Libraries                    ! Для вызова процедур IMSL 77
 use text_transfer                          ! Для вывода русского текста
 use Linear_operators                       ! Для операции .ix. и функции NORM
 implicit none
 integer(4), parameter :: m = 64, n = 32
 real(kind(1e0)), parameter :: one = 1.0
 real(kind(1e0)) err
 real(kind(1e0)), dimension(n, n) :: a, b, x, res, y(m * n), c(m, n), d(m, n)
 integer(4) :: i
 call rand_gen(y); c = reshape(y,(/ m, n /))    ! Генерируем две прямоугольные матрицы
 call rand_gen(y); d = reshape(y,(/ m, n /))
 a = matmul(transpose(c), c)                    ! Формируем систему нормальных уравнений
 b = matmul(transpose(c), d)
 ! Решаем систему Ax = b с несколькими правыми частями
 call Lin_sol_self(a, b, x)
 call check( )
 ! Те же решения получим, вызвав n раз процедуру LSLDS из IMSL 77
 ! для симметрической положительно определенной вещественной матрицы
 ! В случае эрмитовой матрицы следует использовать LSLDH или LFIDH
 do i = 1, n
  call Lslds(n, a, n, b(:, i), x(:, i))
 end do
 call check( )
 ! Получаем решение, применив операцию .ix.. Матрица A – симметрическая
 x = a .ix. b
 ! Проверка результата
 err = norm(b - (a .x. x)) / (norm(a) + norm(b))
 if(err <= sqrt(epsilon(one))) then
  print *, trim(ru_doswin('Удачное завершение примера для LIN_SOL_SELF', .false.))
 end if
contains
 subroutine check( )                        ! Выполняет проверку результата
  res = b - matmul(a, x)                    ! Невязки и относительная ошибка
```

```
  err = maxval(abs(res)) / sum(abs(a) + abs(b))
  if(err <= sqrt(epsilon(one))) then
   print *, trim(ru_doswin('Удачное завершение примера для LIN_SOL_SELF', .false.))
  end if
 end subroutine check
end program sol_self
```

10.4.3. Процедура LIN_SOL_TRI для линейных систем с трехдиагональной матрицей

Процедура LIN_SOL_TRI решает множественную линейную систему

$$A_j x_i = y_j, j = 1, ..., k,$$

в которой каждая матрица A_j является трехдиагональной $n \times n$-матрицей. По умолчанию метод основан на LU-разложении с использованием циклической редукции; как альтернатива может быть использован метод исключения Гаусса с частичным выбором.

Обязательные параметры LIN_SOL_TRI.

Входные/выходные параметры: c, d, b, y.

c – массив формы $(2n, k)$, содержащий верхние диагонали трехдиагональных матриц A_j. Каждая верхняя диагональ находится в сечении $c(1{:}n - 1, j)$. Сечение $c(n, 1{:}k)$ не используется.

d – массив формы $(2n, k)$, содержащий главные диагонали матриц A_j. Каждая диагональ находится в сечении $d(1{:}n, j)$.

b – массив формы $(2n, k)$, содержащий нижние диагонали матриц A_j. Каждая нижняя диагональ находится в сечении $b(2{:}n, j)$. Сечение $b(1, 1{:}k)$ не используется.

y – массив формы $(2n, k)$, содержащий правые части y_j линейной системы. Каждая правая часть находится в сечении $y(1{:}n, j)$. В этом же сечении возвращается и найденное решение x_j.

Размер задачи по умолчанию равен: n = SIZE(d, 1)/2 и k = SIZE(d, 2).

<u>Замечание</u>. Входные данные в обязательных массивах в процессе решения не сохраняются. Поэтому они, если нужны далее, должны быть запомнены в отдельных массивах или файлах.

Необязательные параметры LIN_SOL_TRI.

Входные параметры: *ncols*, *nprob*, *iopt*.

ncols = n – задает используемые сечения входных массивов $c(1{:}n - 1, 1{:}k)$, $d(1{:}n, 1{:}k)$, $b(2{:}n, 1{:}k)$ и $y(1{:}n, 1{:}k)$. По умолчанию n = SIZE(d, 1)/2.

nprob = k – число решаемых линейных систем. По умолчанию k = SIZE(d, 2).

iopt = *iopt*(:) – массив производного типа, приведенных в табл. 10.5 необязательных опций LIN_SOL_TRI. Тип массива – *s_options*.

Таблица 10.5. Опции LIN_SOL_TRI

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_sol_tri_set_small	1
То же	lin_sol_tri_set_jolt	2
"	lin_sol_tri_scan_for_NaN	3
"	lin_sol_tri_factor_only	4
"	lin_sol_tri_solve_only	5
"	lin_sol_tri_use_Gauss_elim	6

iopt(IO) = ?_options(?_lin_sol_tri_set_small, *small*) – при вычислении обратной величины числа последнее будет замещаться на 2*jolt*, если оно меньше *small*. По умолчанию *small* = $0.25\varepsilon_м$.

iopt(IO) = ?_options(?_lin_sol_tri_set_jolt, *jolt*) – устанавливает значение *jolt*. По умолчанию *jolt* = $\varepsilon_м$ = EPSILON(*jolt*).

iopt(IO) = ?_options(?_lin_sol_tri_scan_for_NaN, ?_dummy) – проверяет входные массивы *c*, *d*, *b* и *y* на наличие в них NaN.

iopt(IO) = ?_options(?_lin_sol_tri_factor_only, ?_dummy) – выполняются только *LU*-разложения матриц A_j. Линейные системы не решаются. По умолчанию находятся *LU*-разложения матриц и решения систем.

iopt(IO) = ?_options(?_lin_sol_tri_solve_only, ?_dummy) – решаются линейные системы $A_j x_j = y_j$; при решении используются ранее найденные *LU*-разложения матриц A_j.

iopt(IO) = ?_options(?_lin_sol_tri_use_Gauss_elim, ?_dummy) – вычисление *LU*-разложений матриц будет выполняться методом исключения Гаусса с частичным выбором. По сравнению с алгоритмом циклической редукции метод исключения более точен и стабилен, но обладает большей вычислительной сложностью.

Описание:

Подпрограмма LIN_SOL_TRI решает *k* линейных трехдиагональных систем, каждая из которых имеет размер $n \times n$. При решении плохо обусловленных или вырожденных систем при необходимости выполняется регуляция элементов входной матрицы – они увеличиваются на устанавливаемую опцией lin_sol_tri_set_jolt величину. Это позволяет избежать в процессе разложения матрицы переполнения и деления на нуль. Увеличение значения элемента *a* выполняется при вычислении a^{-1}. Фактически вычисляется величина $(a + t)^{-1}$, где $t = 0$, если $|a| \geq small$. В противном случае $t = 2jolt$. Регуляция необходима, так как трехдиагональный решатель вызывается в процедурах LIN_SVD и LIN_EIG_SELF для выполнения обратных итераций. При использовании заданных по умолчанию значений *small* и *jolt* трехдиаго-

нальная матрица масштабируется так, что ее максимальный элемент близок к единице. Как правило, масштабирование, если система невырожденная, можно опустить. Детали алгоритма см. в [55].

Пример. Решается множественная трехдиагональная линейная система с $n \times n$-матрицами. Скаляр добавляется к главной диагонали, чтобы обеспечить положительную определенность матриц A_j. Правые части системы вычисляются по предварительно сгенерированным в массиве x решениям. Далее этот массив используется для вычисления невязок. Для сравнения те же решения получаются в результате неоднократных вызовов процедуры DLSLTR библиотеки IMSL 77.

```
program sol_tri
 use lin_sol_tri_int; use rand_gen_int
 use Numerical_Libraries                    ! Для вызова процедур IMSL 77
 use text_transfer                          ! Для вывода русского текста
 implicit none
 integer(4) :: i
 integer(4), parameter :: n = 7
 real(kind(1d0)), parameter :: one = 1d0, zero = 0d0
 real(kind(1d0)) err
 real(kind(1d0)), dimension(2 * n, n) :: d, b, c, x, y
 real(kind(1d0)) :: res(n, n), t(n)
 ! Копии массивов для процедуры DLSLTR из IMSL 77
 real(kind(1d0)), dimension(2 * n, n) :: d_save, b_save, c_save, y_save
 ! Генерируем верхние, главные и нижние диагонали матриц Aj и решения xj,
 ! которые затем используем для вычисления правых частей yj
 c = zero; d = zero; b = zero; x = zero
 do i = 1, n
  call rand_gen(c(1:n, i)); call rand_gen(d(1:n, i))
  call rand_gen(b(1:n, i)); call rand_gen(x(1:n, i))
 end do
 ! Добавляем скаляр t к главной диагонали каждой системы,
 ! что обеспечит ее положительную определенность
 ! = sum(c + d + b, dim = 1)
 d(1:n, 1:n) = d(1:n, 1:n) + spread(t, dim = 1, ncopies = n)
 ! Вычисляем правые части yj, получая их в результате умножения Ax
 ! Чтобы получить все правые части в одном выражении, необходимо применить EOSHIFT
 y(1:n, 1:n) = d(1:n, 1:n) * x(1:n, 1:n) +                                  &
     c(1:n, 1:n) * eoshift(x(1:n, 1:n), shift = +1, dim = 1) +              &
     b(1:n, 1:n) * eoshift(x(1:n, 1:n), shift = -1, dim = 1)
 ! Запомним данные для процедуры DLSLTR из IMSL 77
 d_save = d; b_save = b; c_save = c; y_save = y
 ! Вычисляем решения xj, которые возвращаются в массиве y
 call Lin_sol_tri(c, d, b, y)
```

```
 res = x(1:n, 1:n) - y(1:n, 1:n)                ! Невязки и относительная ошибка
 err = sum(abs(res)) / sum(abs(x(1:n, 1:n)))
 if(err <= sqrt(epsilon(one))) then
  print *, trim(ru_doswin('Удачное завершение примера для LIN_SOL_TRI', .false.))
 end if
 do i = 1, n
  ! Решаем линейную систему Ajxj = yj, размещая решение в yj
  call dlsltr(n, b_save(1:n, i), d_save(1:n, i), c_save(1:n, i), y_save(1:n, i))
 ! Вывод результата. Сначала полученного процедурой LIN_SOL_TRI из IMSL 90,
 ! а затем - DLSLTR из IMSL 77
  call dwrrrn('y_F90', 1, n, y(1:n, i), 1, 0)
  call dwrrrn('y_F77', 1, n, y_save(1:n, i), 1, 0)
  read *                                        ! Ожидаем нажатия Enter
 end do
end program sol_tri
```

10.5. СИНГУЛЯРНЫЕ И СОБСТВЕННЫЕ ЗНАЧЕНИЯ

10.5.1. Процедура LIN_SVD для вычисления сингулярного разложения

Процедура LIN_SVD находит SVD-разложение (разд. 7.6.1) прямоугольной матрицы

$$A = U\Sigma V^T.$$

Обязательные параметры LIN_SVD.

Входные/выходные параметры: *a*.

Выходные параметры: *s*, *u*, *v*.

a – массив формы (*m*, *n*), представляющий матрицу *A*.

s – вектор размера min(*m*, *n*), представляющий диагональную матрицу Σ и содержащий вещественные сингулярные значения, расположенные в неубывающем порядке.

u – массив формы (*m*, *m*), хранящий левые сингулярные векторы (матрицу *U*).

v – массив формы (*n*, *n*), хранящий правые сингулярные векторы (матрицу *V*).

Необязательные параметры LIN_SVD.

Входные параметры: *mrows*, *ncols*, *iopt*.

Выходные параметры: *rank*.

mrows = *m* – в вычислениях будет использовано сечение *a*(1:*m*, 1:*n*). По умолчанию *m* = SIZE(*a*, 1).

ncols = *n* – в вычислениях будет использовано сечение *a*(1:*m*, 1:*n*). По умолчанию *n* = SIZE(*a*, 2).

rank = *k* – число сингулярных значений, превышающих величину *small*; справедливо неравенство *k* <= min(*m*, *n*).

iopt = *iopt*(:) – массив производного типа приведенных в табл. 10.6 необязательных опций LIN_SVD. Тип массива – *s_options*.

Таблица 10.6. Опции LIN_SVD

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_svd_set_small	1
То же	lin_svd_overwrite_input	2
"	lin_svd_scan_for_NaN	3
"	lin_svd_use_QR	4
"	lin_svd_skip_Orth	5
"	lin_svd_use_Gauss_elim	6
"	lin_svd_set_perf_ratio	7

iopt(IO) = ?_options(?_lin_svd_set_small, *small*) – задает значение *small*; если сингулярное значение меньше *small*, оно принимается равным нулю и ранг матрицы *A* снижается на единицу. Значение по умолчанию – наименьшее число, для которого можно вычислить обратную величину.

iopt(IO) = ?_options(?_lin_svd_overwrite_input, ?_dummy) – входной массив *a* в процессе вычислений сохраняться не будет (память будет использована другими массивами).

iopt(IO) = ?_options(?_lin_svd_scan_for_NaN, ?_dummy) – проверяет входной массив *a* на наличие в нем NaN. По умолчанию проверка не выполняется.

iopt(IO) = ?_options(?_lin_svd_use_QR, ?_dummy) – для вычисления сингулярных значений и сингулярных векторов будет использован рациональный *QR*-алгоритм. По умолчанию сингулярные векторы вычисляются при помощи обратных итераций.

iopt(IO) = ?_options(?_lin_svd_skip_Orth, ?_dummy) – отключается режим окончательной ортогонализации сингулярных векторов, если для вычисления сингулярных значений применяются обратные итерации. Это приводит к повышению производительности вычислений. Однако сингулярные векторы могут оказаться неортогональными. По умолчанию ортогонализация выполняется.

iopt(IO) = ?_options(?_lin_svd_use_Gauss_elim, ?_dummy) – если вычисление сингулярных значений выполняется посредством обратных итераций, то сингулярные векторы будут находиться стандартным методом исключения с частичным выбором ведущего элемента. По умолчанию сингулярные векторы вычисляются по алгоритму циклической редукции.

iopt(IO) = ?_options(?_lin_svd_set_perf_ratio, *perf_ratio*) – если индекс производительности не превышает *perf_ratio*, то для приближения нормализованных сингулярных векторов используются невязки. В противном случае сингулярные векторы вычисляются заново, причем вместо циклической ре-

дукции применяется стандартный метод исключения; вместо обратных итераций – стандартный *QR*-алгоритм (разд. 8.1.4.1). Увеличение *perf_ratio* приведет к снижению вероятности описанных замен. По умолчанию *perf_ratio* = 4.

Описание:

Подпрограмма LIN_SVD применяет для получения *SVD*-разложения прямоугольной матрицы *QR*-алгоритм. Посредством ортогональных преобразований подобия входная матрица приводится к верхней двухдиагональной форме. Затем вычисляется сингулярное разложение этой матрицы; разложение $AV = U\Sigma$ является произведением промежуточных матриц. С деталями можно ознакомиться в [6].

Пример. Вычисляется *SVD*-разложение матрицы *A*. Проверка точности выполняется по величине невязок $R = AV - US$. Также вызывается выполняющая аналогичные действия процедура DLSVRR из IMSL 77, вслед применяется функция SVD из IMSL 90. Качество решения повышается за счет использования двойной точности.

```
program svd1
 use Lin_svd_int; use rand_gen_int
 use Numerical_Libraries                  ! Для вызова процедур IMSL 77
 use text_transfer                        ! Для вывода русского текста
 use Linear_operators                     ! Для функции SVD
 implicit none
 integer(kind(1)), parameter :: n = 5
 real(kind(1d0)), parameter :: one = 1d0
 real(kind(1d0)) err
 real(kind(1d0)), dimension(n, n) :: a, u, v, s(n), y(n * n)
 integer(kind(1)) :: irank
 call rand_gen(y)                         ! Формируем n×n-матрицу A
 a = reshape(y,(/ n, n /))
 call Lin_svd(a, s, u, v)                 ! Вычисляем сингулярное разложение матрицы A
 ! Вычисляем и оцениваем невязки AV – US.
 err = sum(abs(matmul(a, v) - u * spread(s, dim = 1, ncopies = n))) / sum(abs(s))
 ! Если воспользоваться операцией .x. и функциями NORM и DIAG, то
 ! err = norm((a .x. v) - (u .x. diag(s))) / norm(s)
 if(err <= sqrt(epsilon(one))) then
  print *, trim(ru_doswin('Удачное завершение примера для LIN_SVD', .false.))
  call out( )                             ! Вывод результата
 end if
 ! Те же вычисления выполнит процедура DLSVCR библиотеки IMSL 77
 call dlsvrr(n, n, a, n, 1, 10.0 * dmach(4), irank, s, u, n, v, n)
 call out( )                              ! Вывод результата
 ! Обратимся теперь к функции SVD библиотеки IMSL 90
```

```
 s = svd(a, u = u, v = v)
 call out( )                          ! Вывод результата
contains
 subroutine out                       ! Выводит результат
  call dwrrrn ('u', n, n, u, n, 0)    ! Левые сингулярные векторы
  call dwrrrn ('s', 1, n, s, 1, 0)    ! Сингулярные значения
  call dwrrrn ('v', n, n, v, n, 0)    ! Правые сингулярные векторы
 end subroutine out
end program svd1
```

Замечание. Выражения u*SPREAD(s, dim = 1, $ncopies$ = n), u .x. DIAG(s) и MATMUL(u, DIAG(s)) вернут один и тот же результат.

10.5.2. Процедура LIN_EIG_SELF для вычисления собственных значений самоприсоединенной матрицы

Процедура LIN_EIG_SELF вычисляет собственные значения и собственные векторы самоприсоединенной (симметрической или эрмитовой) матрицы A, что дает разложение

$$A = VDV^T,$$

в котором D – вещественная диагональная матрица, составленная из собственных значений $n \times n$-матрицы A; V – ортогональная $n \times n$-матрица, содержащая собственные векторы матрицы A.

Обязательные параметры LIN_EIG_SELF.

Входные/выходные параметры: a.

Выходные параметры: d.

a – массив формы (n, n), представляющий матрицу A.

d – вектор размера n, содержащий собственные значения, расположенные в неубывающем порядке их абсолютных величин.

Необязательные параметры LIN_EIG_SELF.

Входные параметры: n, $iopt$.

Выходные параметры: v.

$nrows = n$ – в вычислениях будет использовано сечение $a(1{:}n,\ 1{:}n)$. По умолчанию n = SIZE(a, 1).

$v = v(:, :)$ – массив формы (n, n), хранящий ортогональную матрицу V. Имеет такой же тип и разновидность типа, как и массив a.

$iopt = iopt(:)$ – массив производного типа приведенных в табл. 10.7 необязательных опций LIN_EIG_SELF. Тип массива – *s_options*.

Таблица 10.7. Опции LIN_EIG_SELF

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_eig_self_set_small	1
То же	lin_eig_self_overwrite_input	2
"	lin_eig_self_scan_for_NaN	3
"	lin_eig_self_use_QR	4
"	lin_eig_self_skip_Orth	5
"	lin_eig_self_use_Gauss_elim	6
"	lin_eig_self_set_perf_ratio	7

Все приведенные в табл. 10.7 опции имеют тот же смысл, что и аналогичные опции подпрограммы LIN_SVD из предшествующего раздела, с той лишь разницей, что речь идет не о сингулярных, а о собственных значениях и векторах.

Описание:

Применяется *QR*-алгоритм (разд. 8.1.4.1). Посредством ортогональных преобразований подобия входная матрица приводится к трехдиагональной форме. Затем находятся ее собственные значения и собственные векторы.

Пример. Вычисляются собственные и сингулярные значения симметрической матрицы, в качестве которой берется матрица $A = C + C^T$. Модули собственных значений такой матрицы должны совпадать с ее соответствующими сингулярными значениями. Первоначально результат возвращается LIN_EIG_SELF, затем процедурой EVASF из IMSL 77 и вслед функцией EIG из IMSL 90.

```
program eig_self1
 use Lin_eig_self_int; use Lin_sol_svd_int; use rand_gen_int
 use Numerical_Libraries                  ! Для вызова процедур IMSL 77
 use text_transfer                        ! Для вывода русского текста
 use Linear_operators                     ! Для функций EIG и SVD
 implicit none
 integer, parameter :: n = 6
 real(kind(1e0)), parameter :: one = 1.0
 real(kind(1e0)) :: a(n, n), b(n, 0), d(n), s(n), x(n, 0), y(n * n)
 call rand_gen(y)                         ! Генерация матрицы A
 a = reshape(y, (/ n, n /))
 a = a + transpose(a)                     ! Формирование симметрической матрицы A
 call Lin_eig_self(a, d)                  ! Вычисляем собственные значения матрицы A
 call Lin_sol_svd(a, b, x, nrhs = 0, s = s)  ! и для сравнения ее сингулярные значения
 ! Модули собственных и сингулярных значений симметрической матрицы равны
 if(sum(abs(abs(d) - s)) <= sqrt(epsilon(one)) * s(1)) then
```

```
 print *, trim(ru_doswin('Удачное завершение примера для LIN_EIG_SELF', .false.))
 call wrrrn('d', 1, n, d, 1, 0)                ! Собственные значения
 call wrrrn('s', 1, n, s, 1, 0)                ! Сингулярные значения
end if
! EVASF вернет те же значения, что и LIN_EIG_SELF
call evasf(n, n, a, n, .true., d)
call wrrrn('d', 1, n, d, 1, 0)                 ! Собственные значения
! Те же результаты получим, применив функции библиотеки IMSL 90
d = EIG(a)                                     ! Собственные значения
s = SVD(a)                                     ! Сингулярные значения
end program eig_self1
```

10.5.3. Процедура LIN_EIG_GEN для вычисления собственных значений матрицы общего вида

Процедура LIN_EIG_GEN вычисляет собственные значения и собственные векторы общего вида $n \times n$-матрицы A или A^T, что дает разложение

$$A = VEV^T,$$

в котором E – комплексная диагональная матрица, составленная из собственных значений $n \times n$-матрицы A; V – ортогональная $n \times n$-матрица, содержащая собственные векторы A. Дополнительно LIN_EIG_GEN может приводить A к верхней треугольной форме, к форме Шура, к верхней треугольной форме с диагональными 2×2-блочными матрицами или к форме Хессенберга.

Обязательные параметры LIN_EIG_GEN.

Входные/выходные параметры: *a*.

Выходные параметры: *e*.

a – массив формы (*n*, *n*), представляющий матрицу *A*.

e – комплексный вектор размера *n*, содержащий собственные значения, расположенные в неубывающем порядке их абсолютных величин. Разновидность типа *e* такая же, как у массива *a*.

Необязательные параметры LIN_EIG_GEN.

Входные параметры: *nrows*, *iopt*.

Выходные параметры: *v*, *v_adj*, *tri*.

nrows = *n* – в вычислениях будет использовано сечение *a*(1:*n*, 1:*n*). По умолчанию *n* = SIZE(*a*, 1).

v = *v*(:, :) – комплексный массив формы (*n*, *n*), представляющий матрицу *V* с собственными векторами матрицы *A*. Имеет такую же разновидность типа, как и массив *a*. Невязки $AV - VT$ малы.

v_adj = *u*(:, :) – комплексный массив формы (*n*, *n*), представляющий матрицу *U* с собственными векторами матрицы A^T; имеет такую же разновидность типа, как и массив *a*. Невязки $A^T U - U\bar{E}$ малы.

tri = *t*(:, :) – комплексный массив, представляющий верхнюю треугольную матрицу *T*, являющуюся преобразованием матрицы *A* в форму Шура. Дополнительно в массиве *v* может быть возвращена унитарная матрица *W*, такая, что невязки *AW* – *WT* малы.

iopt = *iopt*(:) – массив производного типа приведенных в табл. 10.8 необязательных опций LIN_EIG_GEN. Тип массива – *s_options*.

Таблица 10.8. LIN_EIG_GEN

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_eig_gen_set_small	1
То же	lin_eig_gen_overwrite_input	2
"	lin_eig_gen_scan_for_NaN	3
"	lin_eig_gen_no_balance	4
"	lin_eig_gen_set_iterations	5
"	lin_eig_gen_in_Hess_form	6
"	lin_eig_gen_out_Hess_form	7
"	lin_eig_gen_out_block_form	8
"	lin_eig_gen_out_tri_form	9
"	lin_eig_gen_continue_with_V	10
"	lin_eig_gen_no_sorting	11

Первые 3 опции табл. 10.8 имеют тот же смысл, что и аналогичные опции подпрограммы LIN_SVD из разд. 10.5.1 с той лишь разницей, что речь идет не о сингулярных, а о собственных значениях и векторах.

iopt(IO) = ?_options(?_lin_eig_no_balance, ?_dummy) – отказ от предварительного масштабирования входной матрицы перед вычислением собственных значений. Применяется с некоторыми другими необязательными опциями. По умолчанию матрица подвергается предварительной балансировке.

iopt(IO) = ?_options(?_lin_eig_gen_set_iterations, ?_dummy) – устанавливает максимально допустимое число итераций для выявления каждой блочной диагональной подматрицы. Значение по умолчанию – 52 итерации.

iopt(IO) = ?_options(?_lin_eig_gen_in_Hess_form, ?_dummy) – входная матрица является верхней Хессенберговой. Это опция применяется, чтобы избежать начальную стадию преобразований входной матрицы (см. разд. 8.1.4). По умолчанию входная матрица первоначально преобразовывается в форму Хессенберга.

iopt(IO) = ?_options(?_lin_eig_gen_out_Hess_form, ?_dummy) – выходная матрица будет преобразована в верхнюю форму Хессенберга H_1. Необязательный параметр *v* = *v*(:, :), если задан, возвращает массив, содержащий ортогональную матрицу Q_1, такую, что $AQ_1 - Q_1H_1 \approx 0$. Требует одновременно-

го задания опции ?_lin_eig_no_balance. По умолчанию матрица приводится к диагональной форме.

iopt(IO) = ?_options(?_lin_eig_gen_out_block_form, ?_dummy) – выходная матрица будет приведена к верхней форме Хессенберга H_2, являющейся блочной верхней треугольной. Размеры диагональных блоков либо 2×2, либо единичные. В первом случае матрица имеет одну ненулевую нижнюю кодиагональ. Необязательный параметр $v = v(:, :)$, если задан, возвращает массив, содержащий ортогональную матрицу Q_2, такую, что $AQ_2 - Q_2H_2 \approx 0$. Требует одновременного задания опции ?_lin_eig_no_balance. По умолчанию матрица преобразовывается в диагональную форму.

iopt(IO) = ?_options(?_lin_eig_gen_out_tri_form, ?_dummy) – выходная матрица преобразовывается к верхней треугольной форме T. Необязательный параметр $v = v(:, :)$, если задан, возвращает массив, содержащий унитарную матрицу W, такую, что $AW - WT \approx 0$. Верхняя треугольная матрица возвращается, если задан необязательный параметр $tri = t(:, :)$. Собственные значения матрицы A неупорядоченным образом расположены на диагонали матрицы T. Выходной вектор e в случае применения рассматриваемой опции содержит NaN. Требует одновременного задания опции ?_lin_eig_no_balance. По умолчанию матрица преобразовывается в диагональную форму.

iopt(IO) = ?_options(?_lin_eig_gen_continue_with_V, ?_dummy) – массив v необязательного параметра $v = v(:, :)$ инициализируется матрицей, подобной матрице A. Содержимое v преобразовывается в процессе работы алгоритма. Требует одновременного задания опции ?_lin_eig_no_balance. По умолчанию v инициализируется единичной матрицей.

iopt(IO) = ?_options(?_lin_eig_gen_no_sorting, ?_dummy) – не будет выполняться сортировка собственных значений по мере их вычислений. Это гарантирует соседство комплексно-сопряженных собственных значений в выходном векторе e. По умолчанию собственные значения располагаются в e в неубывающем порядке их абсолютных величин.

Описание:

На первом этапе выполняется масштабирование входной матрицы. Результирующая подобная матрица приводится к верхней форме Хессенберга. Далее работает QR-алгоритм с двойным сдвигом, преобразовывающий матрицу Хессенберга в верхнюю треугольную форму с единичными блоками или блоками размера 2×2 на главной диагонали. В процессе преобразований внедиагональные элементы приравниваются нулю, если их значение меньше пороговой величины. Затем верхняя блочная треугольная блочная матрица вращениями Гивенса преобразовывается в верхнюю треугольную форму. Собственные векторы верхней треугольной матрицы подвергаются обратным преобразованиям, с тем чтобы перейти к собственным векторам матри-

цы A. Процесс организован так, что минимизируется риск переполнения. В завершение собственные векторы нормализуются таким образом, что евклидова длина каждого равна единице.

Пример. Вычисляются собственные значения вещественной матрицы типа REAL(8). Для проверки результата вычисляются невязки $R = AV - VE$, которые должны быть малы. Первоначально результат возвращается LIN_EIG_GEN, затем процедурой DEVCRG библиотеки IMSL 77 и вслед функцией EIG библиотеки IMSL 90.

```
program eig_gen1
 use Lin_eig_gen_int; use rand_gen_int
 use Numerical_Libraries                  ! Для вызова процедур IMSL 77
 use text_transfer                        ! Для вывода русского текста
 use Linear_operators                     ! Для операций и функций библиотеки IMSL 90
 implicit none
 integer(4), parameter :: n = 32
 real(kind(1d0)), parameter :: one = 1d0
 real(kind(1d0)) a(n, n), y(n * n), err
 complex(kind(1d0)) e(n), v(n, n), e_t(n)
 call rand_gen(y)                         ! Генерируем матрицу A
 a = reshape(y, (/ n, n /))
 call Lin_eig_gen(a, e)                   ! Вычисляем только собственные значения
 call Lin_eig_gen(a, e_t, v = v)          ! Вычисляем также и собственные векторы
 ! Проверяем результат. Должно выполняться равенство AV = VD
 call check( )
 ! Приведем для сравнения вызов процедуры DEVCRG библиотеки IMSL 77,
 ! возвращающей все собственные значения и собственные векторы матрицы A
 call devcrg(n, a, n, e, v, n)
 call check( )                            ! Проверка результата
 ! Те же результаты получим, применив функции библиотеки IMSL 90
 e = EIG(a)                               ! Собственные значения
 e_t = EIG(a, w = v)                      ! Собственные векторы
 ! Проверка разложения
 err = norm((a .x. v) - (v .x. diag(e))) / (norm(a) + norm(e))
 if(err <= sqrt(epsilon(one))) then
  print *, trim(ru_doswin('Удачное завершение примера для LIN_EIG_GEN', .false.))
 end if
contains
 subroutine check( )                      ! Вычисляет относительную ошибку
  err = sum(abs(matmul(a, v) - v * spread(e, dim = 1, ncopies = n))) / sum(abs(e))
  if(err <= sqrt(epsilon(one))) then
   print *, trim(ru_doswin('Удачное завершение примера для LIN_EIG_GEN', .false.))
  end if
 end subroutine check
end program eig_gen1
```

10.5.4. Процедура LIN_GEIG_GEN для обобщенной проблемы собственных значений

Процедура LIN_GEIG_GEN решает обобщенную собственную проблему $Av = \lambda Bv$ с $n \times n$-матрицами A и B. Если матрица A или B является вырожденной, дополнительно вычисляются диагональные матрицы α и β и комплексная матрица V, такие, что $Av\beta = BV\alpha$.

Обязательные параметры LIN_GEIG_GEN.

Входные/выходные параметры: *a*, *b*.

Выходные параметры: *alpha*, *beta*.

a – массив формы (*n*, *n*), содержащий матрицу *A*.

b – массив формы (*n*, *n*), содержащий матрицу *B*.

alpha – комплексный вектор размера *n*, содержащий диагональ матрицы α; элементы расположены в *alpha* в неубывающем порядке их абсолютных величин.

beta – вещественный вектор размера *n*, содержащий диагональ матрицы β; элементы расположены в *beta* в неубывающем порядке их величин.

Если $beta(i) \neq 0$, то собственное значение $\lambda(i) = alpha(i)/beta(i)$.

Необязательные параметры LIN_GEIG_GEN.

Входные параметры: *nrows*, *iopt*.

Выходные параметры: *v*, *v_adj*, *tri*.

nrows = *n* – в вычислениях будут использованы сечения *a*(1:*n*, 1:*n*) и *b*(1:*n*, 1:*n*). По умолчанию *n* = SIZE(*a*, 1).

v = *v*(:, :) – комплексный массив формы (*n*, *n*), представляющий матрицу *V* с собственными векторами обобщенной проблемы. Имеет такую же разновидность типа, как и массив *a*.

iopt = *iopt*(:) – массив производного типа приведенных в табл. 10.9 необязательных опций LIN_GEIG_GEN. Тип массива – *s_options*.

Таблица 10.9. LIN_GEIG_GEN

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_geig_gen_set_small	1
То же	lin_geig_gen_overwrite_input	2
"	lin_geig_gen_scan_for_NaN	3
"	lin_geig_gen_self_adj_pos	4
"	lin_geig_gen_for_lin_sol_self	5
"	lin_geig_gen_for_lin_eig_self	6
"	lin_geig_gen_for_lin_sol_lsq	7
"	lin_geig_gen_for_lin_eig_gen	8

iopt(IO) = ?_options(?_lin_geig_gen_set_small, *small*) – допуск, умноженный на сумму абсолютных значений элементов матрицы *B*; используется в

LIN_SOL_LSQ и LIN_SOL_SELF для признания диагональных элементов пренебрежительно малыми. Такие величины могут быть заменены, если заданы опции lin_geig_gen_for_lin_sol_lsq и lin_geig_gen_for_lin_sol_self. По умолчанию *small* = EPSILON(*a*).

iopt(IO) = ?_options(?_lin_geig_gen_overwrite_input, ?_dummy) – входные массивы *a* и *b* не сохраняются. По умолчанию они сохраняются.

iopt(IO) = ?_options(?_lin_geig_gen_scan_for_NaN, ?_dummy) – проверка массивов *a* и *b* на наличие в них NaN. По умолчанию проверка не выполняется.

iopt(IO) = ?_options(?_lin_geig_gen_self_adj_pos, ?_dummy) – если матрицы *A* и *B* самоприсоединенные и вдобавок *B* положительно определенная, то используется алгоритм Холецкого и задача сводится к обычной проблеме на собственные значения для самоприсоединенных матриц.

iopt(IO) = ?_options(?_lin_geig_gen_for_lin_sol_self, ?_dummy) и

iopt(IO + 1) = ?_options((k = size of options for lin_sol_self), ?_dummy) – опции подпрограммы LIN_SOL_SELF следуют как данные в *iopt*.

iopt(IO) = ?_options(?_lin_geig_gen_for_lin_eig_self, ?_dummy) и

iopt(IO + 1) = ?_options((k = size of options for lin_eig_self), ?_dummy) – опции подпрограммы LIN_EIG_SELF следуют как данные в *iopt*.

iopt(IO) = ?_options(?_lin_geig_gen_for_lin_sol_lsq, ?_dummy) и

iopt(IO + 1) = ?_options((k = size of options for lin_sol_lsq), ?_dummy) – опции подпрограммы LIN_SOL_LSQ следуют как данные в *iopt*.

iopt(IO) = ?_options(?_lin_geig_gen_for_lin_eig_gen, ?_dummy) и

iopt(IO + 1) = ?_options((k = size of options for lin_eig_gen), ?_dummy) – опции подпрограммы LIN_EIG_GEN следуют как данные в *iopt*.

Описание:

Подпрограмма LIN_GEIG_GEN реализована на основе стандартного алгоритма, сводящего обобщенную собственную проблему к обычной. Первоначально вычисляется ортогональное разложение $BP^T = HR$, в котором ортогональная матрица H является произведением n - 1 перестановочных матриц и матриц преобразований Хаусхолдера. Перестановки столбцов, хранящиеся в матрице *P*, выполняются таким образом, что рассматриваемая на текущем шаге строка имеет наибольшую евклидову длину. Матрица *R* является верхней треугольной, каждый диагональный элемент которой превышает, если *R* невырожденная, используемый по умолчанию допуск $t = \varepsilon_м\|B\|$. Если *R* вырожденная, *A* и *B* меняются местами и ортогональное разложение вычисляется еще раз. Если вырожденными являются и *A* и *B*, то проблема объявляется вырожденной и не решается. Обмен матриц A и B учитывается выходными диагональными матрицами α и β. Таким образом, задача сводится к обычной собственной проблеме Cx = λx, где матрица $C = H^TAP^TR^{-1}$ и $RPv = x$. Если матрицы *A* и *B* самоприсоединенные и вдобавок *B* положительно оп-

ределенная, то можно задать более эффективный метод Холецкого решения проблемы, предусматривающий разложение $R^TRR = PBP^T$, где R – верхняя треугольная, а P – перестановочная матрицы, т. е. проблема сводится к обычной самоприсоединенной задаче на собственные значения $Cx = \lambda x$, в которой $C = R^{-T}PAP^TR^{-1}$ и $RPv = x$.

Пример. Вычисляются собственные значения обобщенной проблемы. Решение проверяется после получения собственных векторов по невязкам $AV - BV\alpha\beta^{-1}$, которые должны быть малы. Если матрица B невырожденная, то $\beta = I$. Если B вырожденная, но A невырожденная, то некоторые диагональные элементы β равны нулю. Они соответствуют бесконечным собственным значениям. Помимо LIN_GEIG_GEN, для сравнения дается вызов подпрограммы DGVCRG из библиотеки IMSL 77 и функции EIG из IMSL 90. Все 3 процедуры возвращают собственные значения обобщенной проблемы.

```
program geig_gen1
 use lin_geig_gen_int; use rand_gen_int
 use Numerical_Libraries, nouse => beta   ! Для вызова процедур IMSL 77
 use text_transfer                        ! Для вывода русского текста
 use Linear_operators                     ! Для операций и функций библиотеки IMSL 90
 implicit none
 integer(4), parameter :: n = 32
 real(kind(1d0)), parameter :: one = 1d0
 real(kind(1d0)) :: a(n, n), b(n, n), beta(n), beta_t(n), err, y(n * n)
 complex(kind(1d0)) :: alpha(n), alpha_t(n), v(n, n)
 call rand_gen(y); a = reshape(y, (/ n, n /))  ! Генерируем матрицы A и B
 call rand_gen(y); b = reshape(y, (/ n, n /))
 call Lin_geig_gen(a, b, alpha, beta)          ! Собственные значения обобщенной проблемы
 ! Собственные значения и собственные векторы, причем A * V = B * V * values
 call Lin_geig_gen(a, b, alpha_t, beta_t, v = v)
 ! Берем собственные значения от первого вызова и собственные векторы от второго
 ! для вычисления невязок. Они должны быть малы
 call check( )                                 ! Проверка результата
 ! Приведем для сравнения вызов процедуры DGVCRG библиотеки IMSL 77,
 ! возвращающей все собственные значения и собственные векторы обобщенной проблемы
 call dgvcrg(n, a, n, b, n, alpha, beta, v, n)
 call check( )                                 ! Проверка результата
 ! Те же результаты получим, применив функцию EIG из IMSL 90
 alpha = eig(a, b = b, d = beta)               ! Собственные значения
 alpha_t = eig(a, b = b, d = beta_t, w = v)    ! Собственные значения и собственные векторы
 ! Вычисляем невязки
 err = norm((a .x. v .x. diag(beta)) - (b .x. v .x. diag(alpha)), 1) /                    &
     (norm(a, 1) * norm(beta, 1) + norm(b, 1) * norm(alpha, 1))
 if(err <= sqrt(epsilon(one))) then
  print *, trim(ru_doswin('Удачное завершение примера для LIN_GEIG_GEN', .false.))
```

```
 end if
contains
 subroutine check( )                    ! Вычисляет относительную ошибку
  err = sum(abs(matmul(a, v) -                                                  &
            matmul(b, v) * spread(alpha / beta, dim = 1, ncopies = n))) / sum(abs(a) + abs(b))
  if(err <= sqrt(epsilon(one))) then
   print *, trim(ru_doswin('Удачное завершение примера для LIN_GEIG_GEN', .false.))
  end if
end subroutine check
end program geig_gen1
```

10.6. РЕШЕНИЕ ЛИНЕЙНЫХ СИСТЕМ МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ

10.6.1. Процедура LIN_SOL_LSQ для решения прямоугольных линейных систем методом наименьших квадратов

Процедура LIN_SOL_LSQ решает методом наименьших квадратов прямоугольную систему линейных уравнений $Ax \approx b$. Дополнительно, используя необязательные параметры, можно вычислить:

- разложение матрицы A с выполнением перестановок строк и столбцов;
- определитель матрицы A;
- обобщенное обращение матрицы $A^{\dagger}$;
- решение системы $Ax \approx b$ или $A^T y \approx b$ при вычисленном ранее разложении A;
- немасштабированную матрицу ковариаций $C = (A^T A)^{-1}$.

Обязательные параметры LIN_SOL_LSQ.

Входные/выходные параметры: a, b.

Выходной параметры: x.

a – массив формы (m, n), содержащий $m \times n$-матрицу A.

b – массив формы (m, nb), содержащий $m \times nb$-матрицу с правыми частями. Если задано решение системы $A^T x \approx b$, массив b имеет форму (n, nb).

x – массив формы (n, nb), содержащий $n \times nb$-матрицу решений. Если задано решение системы $A^T x \approx b$, массив x имеет форму (m, nb).

Необязательные параметры LIN_SOL_LSQ.

Входные параметры: *mrows*, *ncols*, *nrhs*, *iopt*.

Входные/выходные параметры: *pivots*, *trans*

Выходные параметры: *det*, *ainv*, *cov*.

mrows = m – задает используемое сечение входного массива $a(1{:}m, 1{:}n)$. По умолчанию m = SIZE(a, 1).

ncols = n – задает используемое сечение входного массива $a(1{:}m, 1{:}n)$. По умолчанию m = SIZE(a, 2).

nrhs = *nb* – задает используемое сечение входного массива *b*(1:*n*, 1:*nb*). По умолчанию *nb* = SIZE(*b*, 2). Ранг массива *b* должен быть равен двум.

pivots = *pivots*(:) – целочисленный вектор размера 2min(*m*, *n*) + 1, содержащий данные о перестановках строк (в первых 2min(*m*, *n*) позициях) и столбцов матрицы *A*. Последний элемент массива содержит приблизительный ранг матрицы *A*.

trans = *trans*(:) – вектор размера 2min(*m*, *n*), содержащий данные, позволяющие восстановить ортогональное разложение матрицы *A*. Тип и разновидность типа *trans* такие же, как у массива *a*.

det = *det*(1:2) – вектор размера 2, представляющий произведение определителей матриц *Q*, *P* и *R*. Тип и разновидность типа *det* такие же, как у *a*. Если матрица *A* невырожденная, то определитель равен ABS(*det*(1))**.*det*(2) * (*det*(1)/ABS(*det*(1))), причем ABS(*det*(1)) = RADIX(*det*). В противном случае *det*(1) = 0.0 и *det*(2) = -HUGE(ABS(*det*(1))).

ainv = *ainv*(:, :) – массив формы (*n*, *m*), содержащий $A^{\dagger}$ – обобщенное обращение матрицы *A*. Тип и разновидность типа *ainv* такие же, как у *a*.

cov = *cov*(:, :) – массив формы (*n*×*n*), содержащий немасштабированную матрицу ковариаций $C = (A^T A)^{-1}$. Тип и разновидность типа *cov* такие же, как у *a*.

iopt = *iopt*(:) – массив производного типа приведенных в табл. 10.10 необязательных опций LIN_SOL_LSQ. Тип массива – *s_options*.

Таблица 10.10. Опции LIN_SOL_LSQ

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_sol_lsq_set_small	1
То же	lin_sol_lsq_save_QR	2
"	lin_sol_lsq_solve_A	3
"	lin_sol_lsq_solve_ADJ	4
"	lin_sol_lsq_no_row_pivoting	5
"	lin_sol_lsq_no_col_pivoting	6
"	lin_sol_lsq_scan_for_NaN	7
"	lin_sol_lsq_no_sing_mess	8

iopt(IO) = ?_options(?_lin_sol_gen_set_small, *small*) – задает значение *small*, на которое замещается диагональный элемент матрицы *R*, если он меньше *small*. Решение вычисляется с измененной матрицей. Значение по умолчанию – наименьшее число, для которого можно вычислить обратную величину.

iopt(IO) = ?_options(?_lin_sol_lsq_save_QR, ?_dummy) – будет сохраняться *QR*-разложение матрицы *A*. Для дальнейшего использования найденного *QR*-разложения необходимо при повторных вызовах LIN_SOL_LSQ задавать параметры *pivots* = *pivots*(:) и *trans* = *trans*(:). Если режим запоминания

разложения задан, то входные значения массивов a и b не сохраняются. Для повышения быстродействия на месте диагональных элементов матрицы A сохраняются обратные величины диагональных элементов матрицы R.

iopt(IO) = ?_options(?_lin_sol_lsq_solve_A, ?_dummy) – ранее вычисленное QR-разложение матрицы A используется для решения линейной системы $Ax \approx b$.

iopt(IO) = ?_options(?_lin_sol_lsq_solve_ADJ, ?_dummy) – ранее вычисленное QR-разложение матрицы A используется для решения линейной системы $A^T x \approx b$.

iopt(IO) = ?_options(?_lin_sol_lsq_no_row_pivoting, ?_dummy) – перестановки строк выполняться не будут. Если *pivots* = *pivots*(:) присутствует в списке фактических параметров, то на выходе *pivots*(i) = i, для i = 1, ..., min(m, n).

iopt(IO) = ?_options(?_lin_sol_lsq_no_col_pivoting, ?_dummy) – перестановки столбцов выполняться не будут. Если *pivots* = *pivots*(:) присутствует в списке фактических параметров, то на выходе *pivots*(i + min(m, n)) = i, для i = 1, ..., min(m, n).

iopt(IO) = ?_options(?_lin_sol_lsq_scan_for_NaN, ?_dummy) – проверяет входные массивы a и b на наличие в них NaN. По умолчанию проверка не выполняется.

iopt(IO) = ?_options(?_lin_sol_lsq_no_sing_mess,?_dummy) – отключается вывод сообщений о том, что матрица A вырожденная или что ее ранг $k < \min(m, n)$.

Описание:

В основу процедуры LIN_SOL_LSQ положен приведенный в гл. 8 подход. При разложении матрицы перестанавливаются и строки и столбцы. Это повышает вычислительную устойчивость алгоритма.

Пример. Методом наименьших квадратов решается система $Ac \approx y$, где A – вещественная $m \times n$-матрица ($m > n$), c – вектор коэффициентов многочленов Чебышева, y – вектор значений аппроксимируемой функции. Система описывает задачу представления функции

$$y(x) = e^x \cos(\pi x/2)$$

на дискретном наборе точек из отрезка $-1 \le x \le 1$ в виде многочлена, который, в свою очередь, задается как сумма $T_i(x)$ многочленов Чебышева:

$$u(x) = \sum_{i=0}^{n} c_i T_i(x) .$$

Для сохранения принятого при работе с многочленами Чебышева порядка их нумерации, нижняя граница соответствующих измерений массивов a и c устанавливается равной нулю.

После применения LIN_SOL_LSQ тот же результат получаем процедурой DLSQRR библиотеки IMSL 77 и вслед операцией .ix. из IMSL 90.

```
program sol_lsq
 use Lin_sol_lsq_int; use rand_gen_int
 use text_transfer                          ! Для вывода русского текста
 use Numerical_Libraries                    ! Для вызова процедур библиотеки IMSL 77
 use Linear_operators                       ! Для операции .ix.
 implicit none
 integer(4) :: i, kbasis
 integer(4), parameter :: m = 128, n = 8
 real(kind(1d0)), parameter :: one = 1d0, zero = 0d0
 real(kind(1d0)) :: a(m, 0:n), c(0:n, 1), pi_over_2, x(m), y(m, 1), u(m), v(m), w(m), delta_x
 real(kind(1d0)) :: x_save(m)
 call rand_gen(x)                           ! Генерация случайной сетки точек
 x = x * 2 - one                            ! Преобразовываем точки в отрезок [-1, 1]
 x_save = x                                 ! Для последующего употребления с DLSQRR и .ix.
 pi_over_2 = atan(one) * 2.0                ! Вычисляем константу π / 2
 y(1:m, 1) = exp(x) + cos(pi_over_2 * x)    ! Генерируем значения функции на сетке
 a(:, 0) = one; a(:, 1) = x                 ! Находим матрицу для многочленов Чебышева
 do i = 2, n
  a(:, i) = 2.0 * x * a(:, i - 1) - a(:, i - 2)
 end do
 call Lin_sol_lsq(a, y, c)                  ! Находим коэффициенты многочленов Чебышева
 call check( )                              ! Проверка результата
 ! Вызовем теперь процедуру DLSQRR библиотеки IMSL 77
 ! Вновь генерируем значения функции на сетке x_save
 y(1:m, 1) = exp(x_save) + cos(pi_over_2 * x_save)
 ! Учтем, что протяженность a по второму измерению, а c по первому равна n + 1
 call dlsqrr(m, n + 1, a, m, y(:, 1), 1d-4, c(:, 1), x, kbasis)
 call check( )                              ! Проверка результата
 ! То же решение даст операция .ix.. Генерируем прежде значения функции на сетке x_save
 y(1:m, 1) = exp(x_save) + cos(pi_over_2 * x_save)
 c = a .ix. y                               ! Находим коэффициенты многочленов Чебышева
 call check( )                              ! Проверка результата
contains
 subroutine check( )
  ! Генерируем равномерную сетку на отрезке [-1, 1]
  delta_x = 2 / real(m - 1, kind(one))
  do i = 1, m
   x(i) = -one + (i - 1) * delta_x
  end do
  u = zero; v = zero                        ! Вычисляем невязки, используя обратные
  do i = n, 0, -1                           ! рекуррентные соотношения
   w = 2 * x * u - v + c(i, 1); v = u; u = w
```

```
  end do
  y(1:m, 1) = exp(x) + cos(pi_over_2 * x) - (u - x * v)
  ! Число изменений знака невязок должно быть равно n + 1
  x = one; x = sign(x, y(1:m, 1))
  if(count(x(1:m - 1) /= x(2:m)) >= n + 1) then
   print *, trim(ru_doswin('Удачное завершение примера для LIN_SOL_LSQ', .false.))
  end if
end subroutine check
end program sol_lsq
```

10.6.2. Процедура LIN_SOL_SVD, использующая сингулярное разложение для решения прямоугольных линейных систем

Процедура LIN_SOL_SVD решает методом наименьших квадратов прямоугольную систему линейных уравнений $Ax \approx b$, используя сингулярное разложение $A = U\Sigma V^T$. Дополнительно, задавая необязательные параметры, можно вычислить:

- ранг матрицы A;
- ортогональные $m \times m$- и $n \times n$-матрицы U и V и диагональную $m \times n$-матрицу Σ, содержащую сингулярные значения матрицы A.

Обязательные параметры LIN_SOL_SVD.
Входные/выходные параметры: *a*, *b*.
Выходные параметры: *x*.
a – массив формы (*m*, *n*), содержащий $m \times n$-матрицу A.
b – массив формы (*m*, *nb*), содержащий $m \times nb$-матрицу с правыми частями.
x – массив формы (*n*, *nb*), содержащий $n \times nb$-матрицу решений.

Необязательные параметры LIN_SOL_SVD.
Входные параметры: *mrows*, *ncols*, *nrhs*, *iopt*.
Выходные параметры: *rank*, *u*, *s*, *v*.

mrows = *m* – задает используемое сечение входного массива *a*(1:*m*, 1:*n*). По умолчанию *m* = SIZE(*a*, 1).

ncols = *n* – задает используемое сечение входного массива *a*(1:*m*, 1:*n*). По умолчанию *m* = SIZE(*a*, 2).

nrhs = *nb* – задает используемое сечение входного массива *b*(1:*n*, 1:*nb*). По умолчанию *nb* = SIZE(*b*, 2). Ранг массива *b* должен быть равен двум.

rank = *k* – число сингулярных значений, не меньших поясняемой ниже величины *small*. Причем $k \le \min(m, n)$. Используется для оценки ранга матрицы A.

u = *u*(:, :) – массив формы (*m*, *m*), содержащий ортогональную матрицу U сингулярного разложения. Тип и разновидность типа *u* такие же, как у *a*.

s = *s*(:) – вектор размера min(*m*, *n*), содержащий диагональ матрицы Σ. Вектор всегда вещественный с параметром типа, как у массива *a*. Сингулярные значения неотрицательные и расположены в *s* в неубывающем порядке.

$v = v(:, :)$ – массив формы (n, n), содержащий ортогональную матрицу V сингулярного разложения. Тип и разновидность типа v такие же, как у a.

iopt = *iopt*(:) – массив производного типа приведенных в табл. 10.11 необязательных опций LIN_SOL_SVD. Тип массива – *s_options*.

Таблица 10.11. Опции LIN_SOL_SVD

Префикс опции = ?	*Имя опции*	*Значение*
s_, d_, c_, z_	lin_sol_svd_set_small	1
То же	lin_sol_svd_overwrite_input	2
"	lin_sol_svd_safe_reciprocal	3
"	lin_sol_svd_scan_for_NaN	4

iopt(IO) = ?_options(?_lin_sol_svd_set_small, *small*) – задает значение *small*, на которое замещается диагональный элемент матрицы Σ, если он меньше *small*. Решение вычисляется с измененной матрицей. Значение по умолчанию – наименьшее число, для которого можно вычислить обратную величину.

iopt(IO) = ?_options(?_lin_sol_svd_overwrite_input,?_dummy) – входные массивы a и b не будут сохраняться.

iopt(IO) = ?_options(?_lin_sol_svd_safe_reciprocal, *safe*) – задает значение *safe*, на которое в процессе вычисления обобщенного обращения замещается знаменатель, если он меньше *safe*. Значение по умолчанию – наименьшее число, для которого можно вычислить обратную величину.

iopt(IO) = ?_options(?_lin_sol_svd_scan_for_NaN, ?_dummy) – проверяет входные массивы a и b на наличие в них NaN. По умолчанию проверка не выполняется.

Описание алгоритма см. в [6, разд. 5.5.3].

Пример. Методом наименьших квадратов решается система $Ax \approx b$, где A – вещественная $m \times n$-матрица $(m > n)$. Правда, в этом случае более эффективен рассмотренный выше решатель LIN_SOL_LSQ. Это обусловлено тем, что входная матрица полноранговая. Применение LIN_SOL_SVD обосновано, если матрица A либо плохо обусловленная, либо не имеет полного ранга.

После вызова LIN_SOL_SVD решение получается еще дважды: сначала с применением процедуры DLSVRR библиотеки IMSL 77, затем – функций и операций IMSL 90. Последовательность получения решения средствами библиотеки IMSL 77 проиллюстрирована в разд. 9.1.4. Та же последовательность применяется и при употреблении функций и операций IMSL 90.

```
program sol_svd
 use Lin_sol_svd_int; use rand_gen_int
 use text_transfer                    ! Для вывода русского текста
 use Numerical_Libraries              ! Для вызова процедур библиотеки IMSL 77
```

```
use Linear_operators                    ! Для операции .ix.
implicit none
integer(4), parameter :: m = 128, n = 32
real(kind(1d0)), parameter :: one = 1d0
real(kind(1d0)) :: a(m, n), b(m, 1), x(n, 1), y(m * n), err
! Массивы применяются при работе с функциями и операциями IMSL 90
real(kind(1d0)) :: u(m, m), v(n, n), s(n), g(m), tol
integer(4) :: irank                     ! Параметр процедуры DLSVRR
call rand_gen(y)                        ! Генерируем матрицу A и вектор b
a = reshape(y, (/ m, n /))
call rand_gen(b(1:m, 1))
call Lin_sol_svd(a, b, x)               ! Решаем переопределенную систему Ax ≈ b
call check( )                           ! Проверяем решение
! Получим решение, применив процедуру DLSVRR библиотеки IMSL 77
! Вычисляем сингулярное разложение матрицы A
tol = 10 * epsilon(1d0)
call dlsvrr(m, n, a, m, 11, tol, irank, s, u, m, v, n)
g = matmul(transpose(u), b(:, 1))
where(s > tol) g(1:n) = g(1:n) / s      ! gi = gi / σi
x(:, 1) = matmul(v, g(1:n))             ! x = Vg
call check( )                           ! Проверяем решение
! Используем функции и операции IMSL 90
s = svd(a, u = u, v = v)                ! Решаем переопределенную систему Ax ≈ b
g = u .tx. b(:, 1)
x(:, 1) = v .x. diag(one / s) .x. g(1:n)
! Проверка результата
err = norm(a .tx. (b - (a .x. x))) / (norm(a) + norm(x))
if(err <= sqrt(epsilon(one))) then
 print *, trim(ru_doswin('Удачное завершение примера для LIN_SOL_SVD', .false.))
end if
contains
subroutine check( )                     ! Проверка решения
 ! Невязки должны быть ортогональны вектор-столбцам матрицы A
 err = sum(abs(matmul(transpose(a), b - matmul(a, x)))) / sum(abs(a))
 if(err <= sqrt(epsilon(one))) then
  print *, trim(ru_doswin('Удачное завершение примера для LIN_SOL_SVD', .false.))
 end if
end subroutine check
end program sol_svd
```

Замечание. По непонятным причинам результаты, возвращаемые процедурой DLSVRR библиотеки IMSL 77, отличаются от результатов, выдаваемых LIN_SOL_SVD и функцией SVD библиотеки IMSL 90.

Приложение 1. ВЫВОД РУССКОГО ТЕКСТА В DOS-ОКНО

Рассматриваемая в приложении функция *ru_doswin* не является процедурой IMSL. Однако она вполне уместна в пособии, поскольку позволяет выводить в консоль-проектах русские тексты в DOS-окно. Для ее вызова необходимо обеспечить доступ к модулю *text_transfer*. Также ее можно добавить и в LIB-файлы библиотеки IMSL.

Необходимость создания функции *ru_doswin* объясняется тем, что DOS- и Windows-коды букв русского алфавита различаются. Это обстоятельство надо учитывать при работе с консоль-приложениями FPS 4.0 и CVF, поскольку присутствующие в программе символьные данные имеют Windows-коды, а вывод текста, например оператором

```
print *, 'Сообщение на русском языке'
```

выполняется в DOS-окно.

Так как в Фортране (FPS 4.0 и CVF) не выполняется автоматического преобразования русского Windows-текста в DOS-текст, то об этом должен позаботиться сам пользователь.

Также необходимость преобразований возникает при вводе русского текста с клавиатуры или из DOS-файла: введенный текст имеет DOS-кодировку и для дальнейшей работы необходимо преобразовать его в Windows-текст.

В нижеприводимом модуле *text_transfer* функция *ru_doswin* преобразовывает текст DOS в текст Windows, если *dos_win* = .TRUE., и преобразовывает текст Windows в текст DOS, если *dos_win* = .FASLE..

```
module text_transfer                          ! ncresults - длина строк - результирующих
 integer(4), parameter :: ncresults = 250     ! переменных символьных функций

 contains

! Функция преобразовывает текст DOS в текст Windows, если dos_win = .TRUE.,
! и преобразовывает текст Windows в текст DOS, если dos_win = .FASLE.
 function ru_doswin(string, dos_win)          ! Длина строки string не должна превышать
  character(ncresults) :: ru_doswin           ! ncresults символов
  character(*), intent(in) :: string          ! Параметры string и dos_win имеют вид связи IN
  logical(4), intent(in) :: dos_win           ! и, следовательно, не должны меняться в ru_doswin
  integer(2) :: i, dos_win_code, dif          ! dif - величина, на которую при преобразовании
  ru_doswin = string                          ! изменяется код буквы
  do i = 1, len_trim(ru_doswin)
   dos_win_code = iachar(ru_doswin(i:i))      ! DOS- или Windows-код символа
   dif = 0                                    ! dif больше нуля, если символ - русская буква
   if(dos_win) then                           ! Если преобразование DOS – Windows
```

```
    select case(dos_win_code)          ! Найдем величину dif
    case(128 : 175)                    ! DOS-русские буквы от А до Я и от а до п
      dif = 64
    case(224 : 239)                    ! DOS-русские буквы от р до я
      dif = 16
    end select
   else                                ! Преобразование Windows – DOS
    select case(dos_win_code)
    case(192 : 239)                    ! Windows-русские буквы от А до Я и от а до п
      dif = -64
    case(240 : 255)                    ! Windows-русские буквы от р до я
      dif = -16
    end select
   end if
   ! Выполняем преобразование символа, если он является буквой русского алфавита
   if(dif /= 0) ru_doswin(i:i) = char(dos_win_code + dif)
  end do
 end function ru_doswin
end module text_transfer
```

Пример. Вывести в консоль-приложении заданный в программе русский текст (он имеет Windows-кодировку) в DOS-окно, а введенный с клавиатуры DOS-текст вывести в текстовой файл a.txt, выполнив предварительно преобразование DOS – Windows.

```
program text_go
 use text_transfer
 character(120) :: string
 ! Выводим на консоль DOS-текст Введите строку на русском языке,
 ! получаемый после преобразования Windows – DOS
 ! Встроенная функция TRIM выполняет отсечение концевых пробелов
 print *, trim(ru_doswin('Введите строку на русском языке', .false.))
 read (*, '(a)') string                ! Вводим с клавиатуры DOS-текст
                                       ! Введем: Текст на русском языке
 print *, string                       ! Выводим строку на консоль без преобразований
                                       ! Результат: Текст на русском языке
 open(10, file = 'a.txt')
 write(10, '(a)') string               ! Выводим строку в файл a.txt без преобразований
 ! Просмотрим файл a.txt в Блокноте (NotePad)
 ! Результат – нечитаемый текст: 'Ґбв - аґббЄ®¬ п§лЄҐ
 write(10, '(a)') ru_doswin(string, .true.)  ! Выводим после преобразований строку в файл
                                       ! Просмотрим файл a.txt в Блокноте
end program text_go                    ! Результат: Текст на русском языке
```

Приложение 2. СПИСОК ПРЕДСТАВЛЕННЫХ В ПОСОБИИ ПРОЦЕДУР МАТЕМАТИЧЕСКОЙ БИБЛИОТЕКИ IMSL

Многие процедуры основаны на известных процедурах других пакетов, например LINPACK. Список пакетов, из которых IMSL почерпнула прототипы для своих процедур, приведен в табл. П.-2.1. Также эта таблица включает ссылки на публикации, содержащие перечни базовых процедур линейной алгебры (Basic linear algebra subprograms – BLAS), на базе которых созданы BLAS-процедуры IMSL.

Таблица П.-2.1. Пакеты и источники для процедур IMSL

Пакет или источник	*Где описан*
LINPACK	34
TOEPLITZ	21
EISPACK, National Laboratory	69, 42
Уровень 1 BLAS (BLAS 1)	57
Уровень 2 BLAS (BLAS 2)	36, 35
MINPACK	65

Все процедуры IMSL следуют разработанной институтом NISTIR GAMS-классификации [28], основные категории которой приведены в табл. П.-2.2.

Таблица П.-2.2. GAMS-категории

Категория	*Содержание*
A	Арифметика, анализ ошибок
B	Теория чисел
C	Элементарные и специальные функции
D	Линейная алгебра
E	Интерполяция
F	Решение нелинейных уравнений
G	Оптимизация
H	Дифференцирование и интегрирование
I	Дифференциальные и интегральные уравнения
J	Интегральные преобразования
K	Аппроксимация
L	Статистика, вероятность
M	Моделирование, стохастическое моделирование
N	Обработка данных
O	Символические вычисления

P	Вычислительная геометрия
Q	Графика
R	Обслуживающие процедуры
S	Средства разработки программного обеспечения
Z	Другое

В табл. П.-2.3 дается в алфавитном порядке список рассмотренных в пособии процедур IMSL с указанием их назначения. Также для BLAS-процедур и процедур, подробное описание которых отсутствует в пособии, приводятся ссылки на пакет, процедуру-прототип и авторов, из работ которых почерпнуты алгоритмы, положенные в основу процедур. Подробную информацию об источнике можно найти по его номеру в списке литературы.

Как всегда, для группы процедур указывается имя лишь одной, работающей с типом REAL(4) или COMPLEX(4). Так, группу процедур LFCCG, DLFCCG, L2CCG и DL2CCG, вычисляющих *LU*-разложение комплексной матрицы общего вида и выполняющих оценку ее числа обусловленности, в таблице представляет процедура LFCCG.

В последней графе таблицы указан индекс процедуры в GAMS-классификации. Расшифровка индексов дана в табл. П.-2.4.

Таблица П.-2.3. Список представленных в пособии процедур IMSL

Процедура	*Что делает/источники*	*GAMS-индекс*
AMACH	Возвращает вещественные машинные константы одинарной точности	R1
ARBRB	Выполняет сложение ленточных матриц	D1b5
BLINF	Вычисляет билинейную и квадратичную формы	D1b4
CCONV	Вычисляет свертку двух комплексных векторов	J2
CHBCB	Преобразовывает верхнюю эрмитову ленточную матрицу в полную эрмитову ленточную матрицу	D1b9
CHBMV	Умножает эрмитову ленточную матрицу на вектор / BLAS 2 и 3	D1b4
CHEMM	Находит произведение с эрмитовой матрицей / То же	D1b6
CHEMV	Умножение эрмитовой матрицы на вектор / "	D1b4
CHER	Выполняет преобразования ранга 1 эрмитовой матрицы / "	D1b
CHER2	Выполняет преобразования ранга 2 эрмитовой матрицы / "	D1b
CHER2K	Выполняет преобразования ранга $2k$ эрмитовой матрицы / "	D1b
CHERK	Выполняет преобразования ранга k эрмитовой матрицы / "	D1b
CHFAC	Вычисляет R^TR-разложение Холецкого вещественной симметрической неотрицательно определенной матрицы / См. описание процедуры	D2b1b
CHFCG	Преобразовывает верхнюю эрмитову матрицу в полную эрмитову матрицу	D1b9

CONST	Возвращает математические и физические константы / См. описание процедуры	C19
CPSEC	Вычисляет процессорное время, прошедшее с момента первого вызова CPSEC	S3
CRBRB	Копирует ленточные матрицы	D1b8
CRBRG	Преобразовывает ленточную матрицу в матрицу общего вида	D1b9
CRGRB	Преобразовывает матрицу общего вида в ленточную матрицу	D1b9
CSBRB	Преобразовывает верхнюю симметрическую ленточную матрицу в полную симметрическую ленточную матрицу	D1b9
CSFRG	Преобразовывает верхнюю симметрическую матрицу в полную симметрическую матрицу	D1b9
CUNIT	Осуществляет перевод величин из одной системы единиц в другую	C19
CVTSI	Преобразовывает строку, состоящую из цифр, в соответствующее целое число	N3
DISL1	Вычисляет манхеттеново расстояние между точками	D1a3a
DISL2	Вычисляет евклидово расстояние между точками	D1a3b
DISLI	Вычисляет ∞-нормы расстояние между точками	D1a3c
DMACH	Возвращает вещественные машинные константы двойной точности	R1
DQADD	Добавляет в аккумулятор повышенной точности число *da*	A3c
DQINI	Заносит числа в аккумулятор *qacc* повышенной точности	A3c
DQMUL	Добавляет в аккумулятор повышенной точности произведение *qacc* = *qacc* + *da* * *db*	A3c
DQSTO	Выполняет присваивание *da* = *qacc*, где *qacc* – содержимое аккумулятора повышенной точности	A3c
DSDOT	Возвращает скалярное произведение векторов с аккумулятором двойной точности / BLAS 1	D1a4
EPICG	Вычисляет оценку погрешности решения проблемы собственных значений с комплексной неэрмитовой матрицей / EISPACK; [69, p. 124-125]	D4c
EPIHF	Вычисляет оценку погрешности решения проблемы собственных значений с комплексной эрмитовой матрицей / То же	D4c
EPIRG	Вычисляет оценку погрешности решения проблемы собственных значений с вещественной несимметрической матрицей / "	D4c
EPISB	Вычисляет оценку погрешности решения проблемы собственных значений с вещественной симметрической ленточной матрицей / "	D4c
EPISF	Вычисляет оценку погрешности решения проблемы собственных значений с вещественной симметрической матрицей / "	D4c
ERSET	Изменяет заданные по умолчанию действия, связанные с выводом сообщений и остановкой программы при возникновении ошибки заданного уровня / См. описание процедуры	R3
EVAHF	Вычисляет несколько наибольших или несколько наименьших собственных значений комплексной эрмитовой матрицы / EISPACK, HTRIDI, RATQR	D4a3

EVASB	Вычисляет несколько наибольших или несколько наименьших собственных значений вещественной симметрической ленточной матрицы / EISPACK, BANDR, RATQR	D4a6
EVASF	Вычисляет несколько наибольших или несколько наименьших собственных значений вещественной симметрической матрицы / EISPACK, TRED2; [66, p. 169]	D4a1
EVBHF	Вычисляет собственные значения комплексной эрмитовой матрицы, принадлежащие заданному интервалу / EISPACK, HTRIDI, BISECT	D4a3
EVBSB	Вычисляет собственные значения вещественной симметрической ленточной матрицы, принадлежащие заданному интервалу / EISPACK, BANDR, BISECT	D4a6
EVBSF	Вычисляет собственные значения вещественной симметрической матрицы, принадлежащие заданному интервалу / EISPACK, TRED1	D4a1
EVCCG	Вычисляет все собственные значения и собственные векторы комплексной неэрмитовой матрицы / См. описание процедуры	D4a4
EVCCH	Вычисляет все собственные значения и собственные векторы комплексной верхней матрицы Хессенберга / То же	D4c2b
EVCHF	Вычисляет все собственные значения и собственные векторы комплексной эрмитовой матрицы / EISPACK, HTRIDI, IMTQL2	D4a3
EVCRG	Вычисляет все собственные значения и собственные векторы вещественной несимметрической матрицы / См. описание процедуры	D4a2
EVCRH	Вычисляет все собственные значения и собственные векторы вещественной верхней матрицы Хессенберга / EISPACK, HQR2	D4c2b
EVCSB	Вычисляет все собственные значения и собственные векторы вещественной симметрической ленточной матрицы / EISPACK, BANDR, IMTQL2	D4a6
EVCSF	Вычисляет все собственные значения и собственные векторы вещественной симметрической матрицы / см. описание процедуры	D4a1
EVEHF	Вычисляет наибольшее или наименьшее собственное значение и соответствующий собственный вектор комплексной эрмитовой матрицы / EISPACK, HTRIDI, TINVIT, HTRIBK	D4a3
EVESB	Вычисляет наибольшее или наименьшее собственное значение и соответствующий собственный вектор вещественной симметрической ленточной матрицы / EISPACK, BANDR, RATQR, BANDV; [51]	D4a6
EVESF	Вычисляет наибольшее или наименьшее собственное значение и соответствующий собственный вектор вещественной симметрической матрицы / См. описание процедуры	D4a1
EVFHF	Вычисляет принадлежащие заданному числовому интервалу собственные значения и соответствующие собственные векторы комплексной эрмитовой матрицы / EISPACK, HTRIDI, BISECT, TINVIT, HTRIBK	D4a3

EVFSB	Вычисляет принадлежащие заданному числовому интервалу собственные значения и соответствующие собственные векторы вещественной симметрической ленточной матрицы / EISPACK, BANDR, BISECT, BANDV; [51]	D4a6
EVFSF	Вычисляет принадлежащие заданному числовому интервалу собственные значения и соответствующие собственные векторы вещественной симметрической матрицы / См. описание процедуры EVESF	D4a1
EVLCG	Вычисляет все собственные значения комплексной неэрмитовой матрицы / EISPACK, [66, p. 169]	D4a4
EVLCH	Вычисляет все собственные значения комплексной верхней матрицы Хессенберга / EISPACK, COMQR2	D4c2b
EVLHF	Вычисляет все собственные значения комплексной эрмитовой матрицы / EISPACK, HTRIDI, IMTQL1	D4a3
EVLRG	Вычисляет все собственные значения вещественной несимметрической матрицы / EISPACK, BALANC, ELMHES; [51]	D4a2
EVLRH	Вычисляет все собственные значения вещественной верхней матрицы Хессенберга / EISPACK, HQR	D4c2b
EVLSB	Вычисляет все собственные значения вещественной симметрической ленточной матрицы / EISPACK, BANDR, IMTQL1	D4a6
EVLSF	Вычисляет все собственные значения вещественной симметрической матрицы / EISPACK, TRED2; [66, p. 169; 51]	D4a1
GIRTS	Решает треугольную (возможно, вырожденную) систему линейных уравнений и вычисляет обобщенное обращение верхней треугольной матрицы / См. описание процедуры	D9c
GPICG	Вычисляет оценку погрешности решения обобщенной проблемы собственных значений с комплексной неэрмитовой матрицей / То же	D4c
GPIRG	Вычисляет оценку погрешности решения обобщенной проблемы собственных значений с вещественной несимметрической матрицей / EISPACK; [42, p. 77-79]	D4c
GPISP	Вычисляет оценку погрешности решения обобщенной проблемы собственных значений с вещественной симметрической матрицей / То же	D4c
GVCCG	Вычисляет все собственные значения и собственные векторы обобщенной собственной системы с комплексными неэрмитовыми матрицами A и B / [64; 51]	D4b4
GVCRG	Вычисляет все собственные значения и собственные векторы обобщенной собственной системы с вещественными несимметрическими матрицами A и B / См. описание процедуры	D4b2
GVCSP	Вычисляет все собственные значения и собственные векторы обобщенной собственной системы с вещественными симметрическими матрицей A и положительно определенной матрицей B / То же	D4b1
GVLCG	Вычисляет все собственные значения обобщенной собственной системы с комплексными неэрмитовыми матрицами A и B / [64; 51]	D4b4

GVLRG	Вычисляет все собственные значения обобщенной собственной системы с вещественными несимметрическими матрицами *A* и *B* / EISPACK, QZHES, QZIT и QZVAL; [64]	D4b2
GVLSP	Вычисляет все собственные значения обобщенной собственной системы с вещественными симметрическими матрицей *A* и положительно определенной матрицей *B* / [63, 51]	D4b1
HYPOT	Вычисляет SQRT(*a***2 + *b***2) без исчезновения порядка и переполнения / См. описание процедуры	C2
ICASE	Возвращает ASCII-значение символа без учета регистра	N3
IDYWK	Вычисляет день недели для заданной даты	R
IERCD	Возвращает код последней, возникшей при вызове процедур IMSL информационной ошибки / См. описание процедуры	R3
IFNAN	Проверяет, содержит ли переменная NaN (не число) / То же	R1
IICSR	Выполняет сравнение строк без учета регистра	N3
IIDEX	Возвращает позицию подстроки в строке без учета регистра	N3
IMACH	Возвращает целые машинные константы	R1
ISAMAX	Возвращает индекс максимального по модулю элемента вектора / BLAS 1	D1a2
ISAMIN	Возвращает индекс минимального по модулю элемента вектора / То же	D1a2
ISMAX	Возвращает индекс максимального элемента вектора / "	D1a2
ISMIN	Возвращает индекс минимального элемента вектора / "	D1a2
IUMAG	Задает и читает целочисленные параметры процедур / "	R
JCGRC	Решает систему линейных уравнений $Ax = b$ с симметрической положительно определенной разреженной матрицей *A* методом сопряженных градиентов с аппроксимацией Якоби и обратной связью / "	D2b4
LCLSQ	Решает методом наименьших квадратов проблему $Ax = b$ с линейными ограничениями / "	D9b1
LDNCH	Осуществляет пересчет разложения Холецкого / "	D7
LFCCB	Вычисляет *LU*-разложение и выполняет оценку числа обусловленности комплексной неэрмитовой ленточной матрицы / LINPACK, CGBCO; [31]	D2c2
LFCCG	Вычисляет *LU*-разложение и оценку числа обусловленности комплексной неэрмитовой матрицы общего вида / LINPACK, CGECO; [31]	D2c1
LFCCT	Выполняет оценку числа обусловленности матрицы / LINPACK, CTRCO	D2c3
LFCDH	Вычисляет $R^H R$-разложение комплексной положительно определенной эрмитовой матрицы и выполняет оценку ее числа обусловленности / LINPACK, CPOCO; [31]	D2d1b
LFCDS	Вычисляет $R^T R$-разложение вещественной симметрической положительно определенной матрицы и выполняет оценку ее числа обусловленности / LINPACK, SPOCO; [31]	D2b1b

LFCQH	Вычисляет $R^H R$-разложение комплексной эрмитовой ленточной положительно определенной матрицы и выполняет оценку ее числа обусловленности / LINPACK, CPBCO; [31]	D2d2
LFCQS	Вычисляет $R^T R$-разложение вещественной симметрической положительно определенной ленточной матрицы и выполняет оценку ее числа обусловленности / LINPACK, SPBCO; [31]	D2b2
LFCRB	Вычисляет LU-разложение и выполняет оценку числа обусловленности вещественной несимметрической ленточной матрицы / См. описание процедуры	D2a2
LFCRG	Вычисляет LU-разложение и выполняет оценку числа обусловленности вещественной несимметрической общего вида матрицы / То же	D2a1
LFCRT	Выполняет оценку числа обусловленности вещественной треугольной матрицы / "	D2a3
LFCSF	Вычисляет UDU^T-разложение вещественной симметрической матрицы и выполняет оценку ее числа обусловленности / LINPACK, SSICO; [31]	D2b1a
LFDCB	Вычисляет определитель комплексной неэрмитовой ленточной матрицы / LINPACK, CGBDI	D3c2
LFDCG	Вычисляет определитель комплексной неэрмитовой общего вида матрицы / LINPACK, CGEDI	D3c1
LFDDH	Вычисляет определитель комплексной эрмитовой положительно определенной матрицы / LINPACK, CPODI	D3d1b
LFDDS	Вычисляет определитель вещественной симметрической положительно определенной матрицы / LINPACK, SPODI	D3b1b
LFDHF	Вычисляет определитель комплексной эрмитовой неопределенной матрицы / LINPACK, CSIDI	D2d1a
LFDQH	Вычисляет определитель комплексной эрмитовой положительно определенной ленточной матрицы / LINPACK, CPBDI	D3d2
LFDQS	Вычисляет определитель вещественной симметрической положительно определенной ленточной матрицы / LINPACK, SPBDI	D2b2
LFDRB	Вычисляет определитель вещественной ленточной матрицы / LINPACK, CGBDI	D3a2
LFDRG	Вычисляет определитель вещественной матрицы общего назначения / LINPACK, SGEDI	D3a1
LFDRT	Вычисляет определитель вещественной треугольной матрицы / LINPACK, STRDI	D3a3
LFDRT	Вычисляет определитель комплексной неэрмитовой треугольной матрицы / То же	D3a3
LFDSF	Вычисляет определитель вещественной симметрической неопределенной матрицы / LINPACK, SSIDI	D3b1a
LFICB	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение комплексной неэрмитовой ленточной матрицы A, и выполняет итерационное уточнение решения	D2c2

LFICG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение комплексной неэрмитовой общего вида матрицы A, и выполняет итерационное уточнение решения	D2c1
LFIDH	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^HR-разложение комплексной положительно определенной эрмитовой матрицы A, и выполняет итерационное уточнение решения	D2d1b
LFIDS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^TR-разложение вещественной симметрической положительно определенной матрицы A, и выполняет итерационное уточнение решения	D2b1b
LFIQH	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^HR-разложение комплексной эрмитовой ленточной положительно определенной матрицы A, и выполняет итерационное уточнение решения	D2d2
LFIQS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^TR-разложение вещественной симметрической положительно определенной ленточной матрицы A, и выполняет итерационное уточнение решения	D2b2
LFIRB	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение вещественной несимметрической ленточной матрицы A, и выполняет итерационное уточнение решения	D2a1
LFIRG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение вещественной несимметрической общего вида матрицы A, и выполняет итерационное уточнение решения	D2a1
LFISF	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное UDU^T-разложение вещественной симметрической матрицы A, и выполняет итерационное уточнение решения	D2b1a
LFSCB	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение комплексной неэрмитовой ленточной матрицы A / LINPACK, CGBSL; [31]	D2c2
LFSCG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение комплексной неэрмитовой общего вида матрицы A / LINPACK, CGESL	D2c1
LFSDH	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^HR-разложение комплексной эрмитовой положительно определенной матрицы A / LINPACK, CPOSL	D2d1b
LFSDS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^TR-разложение вещественной симметрической положительно определенной матрицы A / LINPACK, SPOSL	D2b1b
LFSQH	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^HR-разложение комплексной эрмитовой ленточной положительно определенной матрицы A / LINPACK, CPBSL	D2d2
LFSQS	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное R^TR-разложение вещественной симметрической положительно определенной ленточной матрицы A / LINPACK, SPBSL	D2b2

LFSRB	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение вещественной несимметрической ленточной матрицы A / [37]	D2a2
LFSRG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение вещественной несимметрической общего вида матрицы A / LINPACK, SGESL	D2a1
LFSSF	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное UDU^T-разложение симметрической матрицы A / LINPACK, SSISL	D2b1a
LFSXD	Решает систему линейных уравнений с симметрической положительно определенной разреженной матрицей с использованием ранее найденного LL^T-разложения Холецкого / См. описание процедуры LSCXD	D2b4
LFSXG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение вещественной несимметрической разреженной матрицы A / См. описание процедуры LFTXG	D2a4
LFSZD	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LL^T-разложение Холецкого комплексной эрмитовой разреженной матрицы A / См. описание процедуры LSCXD	D2d4
LFSZG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение матрицы A / [33, 38]	D2c4
LFTCB	Вычисляет LU-разложение комплексной неэрмитовой ленточной матрицы общего вида без оценки ее числа обусловленности / LINPACK, CGBFA	D2c2
LFTCG	Вычисляет LU-разложение комплексной неэрмитовой матрицы общего вида без оценки ее числа обусловленности / LINPACK, CGEFA	D2c1
LFTDH	Вычисляет R^HR-разложение комплексной положительно определенной эрмитовой матрицы без оценки ее числа обусловленности / LINPACK, CPOFA	D2d1b
LFTDS	Вычисляет R^TR-разложение вещественной симметрической положительно определенной матрицы без оценки ее числа обусловленности / LINPACK, SPOFA	D2b1b
LFTQH	Вычисляет R^HR-разложение комплексной эрмитовой ленточной положительно определенной матрицы без оценки ее числа обусловленности / LINPACK, SPBFA	D2d2
LFTQS	Вычисляет R^HR-разложение эрмитовой положительно определенной ленточной матрицы без оценки ее числа обусловленности / LINPACK, CPBFA	D2b2
LFTRB	Вычисляет LU-разложение вещественной несимметрической ленточной матрицы без оценки ее числа обусловленности / [37]	D2a2
LFTRG	Вычисляет LU-разложение вещественной несимметрической общего вида матрицы без оценки ее числа обусловленности / См. описание процедуры	D2a1
LFTSF	Вычисляет UDU^T-разложение вещественной симметрической матрицы без оценки ее числа обусловленности / LINPACK, SSIFA	D2b1a

LFTXG	Вычисляет *LU*-разложение вещественной несимметрической разреженной матрицы / См. описание процедуры	D2a4
LFTZG	Вычисляет *LU*-разложение матрицы комплексной неэрмитовой разреженной матрицы / [33, 38]	D2c4
LIN_EIG_GEN	Вычисляет собственные значения и собственные векторы матрицы общего вида. Процедура библиотеки IMSL F90 / См. описание процедуры	D4a3 D4a4
LIN_EIG_SELF	Вычисляет собственные значения и собственные векторы самоприсоединенной матрицы. Процедура библиотеки IMSL F90 / То же	D4a1 D4a3
LIN_GEIG_GEN	Решает обобщенную собственную проблему. Процедура библиотеки IMSL F90 / "	D4b1 D4b2 D4b4
LIN_SOL_GEN	Решает линейную систему $Ax = b$ с матрицей общего вида. Вычисляет *LU*-разложение матрицы A, A^{-1}, det(A). Процедура библиотеки IMSL F90 / "	D2a1 D2c1
LIN_SOL_LSQ	Решает прямоугольную линейную систему методом наименьших квадратов. Процедура библиотеки IMSL F90 / "	D9a1 D9c
LIN_SOL_SELF	Решает линейную систему $Ax = b$ с самоприсоединенной матрицей. Вычисляет разложение матрицы A, A^{-1}, det(A). Процедура библиотеки IMSL F90 / "	D2b1a D2b1b D2d1a D2d1a
LIN_SOL_SVD	Решает прямоугольную линейную систему методом наименьших квадратов, используя сингулярное разложение. Процедура библиотеки IMSL F90 / "	D9a1 D6
LIN_SOL_TRI	Решает линейную систему $Ax = b$ с трехдиагональной матрицей. Процедура библиотеки IMSL F90 / "	D2a2a D2c2a
LIN_SVD	Вычисляет сингулярное разложение матрицы. Процедура библиотеки IMSL F90 / "	D6
LINCG	Вычисляет *LU*-разложение комплексной неэрмитовой общего вида матрицы A, а затем находит матрицу A^{-1}	D2c1
LINCT	Вычисляет матрицу, обратную комплексной неэрмитовой треугольной матрице	D2c3
LINDS	Вычисляет R^TR-разложение вещественной симметрической положительно определенной матрицы A, а затем находит матрицу A^{-1}	D2b1b
LINRG	Вычисляет *LU*-разложение вещественной несимметрической общего вида матрицы A, а затем находит матрицу A^{-1}	D2a1
LINRT	Вычисляет матрицу, обратную вещественной треугольной матрице	D2a3
LNFXD	Находит числовое разложение Холецкого симметрической положительно определенной разреженной матрицы / См. описание процедуры LSCXD	D2b4
LNFZD	Вычисляет LL^T-разложение Холецкого комплексной эрмитовой разреженной матрицы / То же	D2d4
LQERR	Восстанавливает ортогональную матрицу Q из QR-разложения	D5

LQRRR	Вычисляет QR-разложение вещественной матрицы / См. описание процедуры	D5
LQRRV	Решает переопределенные и недоопределенные системы линейных уравнений методом наименьших квадратов с использованием блочных преобразований Хаусхолдера / То же	D9a1
LQRSL	Вычисляет преобразования координат и завершает решение проблемы $Ax = b$ методом наименьших квадратов / "	D9a1
LSACB	Вычисляет LU-разложение комплексной неэрмитовой ленточной матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2c2
LSACG	Вычисляет LU-разложение комплексной неэрмитовой общего вида матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2c1
LSADH	Вычисляет R^TR-разложение комплексной положительно определенной эрмитовой матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2d1b
LSADS	Вычисляет R^TR-разложение вещественной симметрической положительно определенной матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2b1b
LSAQH	Вычисляет R^TR-разложение комплексной эрмитовой ленточной положительно определенной матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2d2
LSAQS	Вычисляет R^TR-разложение вещественной симметрической положительно определенной ленточной матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2b2
LSARB	Вычисляет LU-разложение вещественной несимметрической ленточной матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2a1
LSARG	Вычисляет LU-разложение вещественной несимметрической общего вида матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2a1
LSASF	Вычисляет UDU^T-разложение вещественной симметрической матрицы A, решает систему линейных уравнений $Ax = b$ и выполняет итерационное уточнение решения	D2b1a
LSBRR	Решает переопределенные и недоопределенные системы линейных уравнений методом наименьших квадратов и выполняет итерационное уточнение корней / См. описание процедуры	D9a1
LSCXD	Вычисляет структуру данных для представления нижней треугольной матрицы L из LL^T-разложения Холецкого симметрической положительно определенной разреженной матрицы / То же	D2b4
LSGRR	Вычисляет обобщенную обратную матрицу	D9
LSLCB	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение комплексной неэрмитовой ленточной матрицы A без выполнения итерационного уточнения решения	D2c2
LSLCC	Решает систему линейных уравнений с циклической матрицей	D2c.

LSLCG	Решает систему линейных уравнений $Ax = b$, применяя ранее найденное LU-разложение комплексной неэрмитовой общего вида матрицы A без выполнения итерационного уточнения решения	D2c1
LSLCQ	Вычисляет LDU-разложение комплексной неэрмитовой трехдиагональной матрицы / [55, 50]	D2c2a
LSLCR	Вычисляет LDU-разложения вещественной несимметрической трехдиагональной матрицы / [50]	D2a2a
LSLCT	Решает систему линейных уравнений $Ax = b$ с комплексной неэрмитовой треугольной матрицей / LINPACK, CTRSL	D2c3
LSLDH	Вычисляет R^HR-разложение комплексной положительно определенной эрмитовой матрицы A и решает систему линейных уравнений $Ax = b$	D2d1b
LSLDS	Вычисляет R^TR-разложение вещественной симметрической положительно определенной матрицы A и решает систему линейных уравнений $Ax = b$	D2b1b
LSLPB	Выполняет R^TDR-разложение Холецкого вещественной симметрической положительно определенной ленточной матрицы и решает систему $Ax = b$	D2b2
LSLQB	Вычисляет R^HDR-разложение Холецкого комплексной эрмитовой ленточной положительно определенной матрицы и решает систему $Ax = b$ / [49]	D2d2
LSLQH	Вычисляет R^HR-разложение комплексной эрмитовой ленточной положительно определенной матрицы A и решает систему линейных уравнений $Ax = b$	D2d2
LSLQS	Вычисляет R^TR-разложение вещественной симметрической положительно определенной ленточной матрицы A и решает систему линейных уравнений $Ax = b$	D2b2
LSLRB	Вычисляет LU-разложение вещественной несимметрической ленточной матрицы A и решает систему линейных уравнений $Ax = b$	D2a1
LSLRG	Вычисляет LU-разложение вещественной несимметрической общего вида матрицы A и решает систему линейных уравнений $Ax = b$	D2a1
LSLRT	Решает систему линейных уравнений $Ax = b$ с вещественной треугольной матрицей A / LINPACK, STRSL	D2a3
LSLSF	Вычисляет UDU^T-разложение вещественной симметрической матрицы A и решает систему линейных уравнений $Ax = b$	D2b1a
LSLTC	Решает систему линейных уравнений с комплексной неэрмитовой матрицей Теплица / TOEPLITZ, TSLC; [70; 45, p. 125-133]	D2c
LSLTO	Решает систему линейных уравнений с вещественной матрицей Теплица / См. описание процедуры	D2a
LSLTQ	Решает трехдиагональную систему линейных уравнений с комплексной неэрмитовой трехдиагональной матрицей / LINPACK, CGTSL	D2c2a
LSLTR	Решает трехдиагональную систему линейных уравнений с вещественной несимметрической трехдиагональной матрицей / См. описание процедуры	D2a2a

LSLXD	Решает систему линейных уравнений с симметрической положительно определенной разреженной матрицей методом Холецкого / См. описание процедуры LSCXD	D2b4
LSLXG	Решает систему линейных уравнений $Ax = b$ методом исключения Гаусса с вещественной несимметрической разреженной матрицей A / См. описание процедуры	D2a4
LSLZD	Решает систему линейных уравнений $Ax = b$ методом Холецкого с комплексной эрмитовой разреженной матрицей / См. описание процедуры LSCXD	D2d4
LSLZG	Решает систему линейных уравнений $Ax = b$ методом исключения Гаусса, выполняя итерационное улучшение решения / См. описание процедуры LSLXG	D2c4
LSQRR	Решает переопределенные и недоопределенные системы линейных уравнений методом наименьших квадратов без итерационного уточнения корней	D9a1
LSVCR	Находит сингулярное разложение комплексной матрицы / См. описание процедуры	D6
LSVRR	Находит сингулярное разложение вещественной матрицы / То же	D6
LUPCH	Осуществляет пересчет разложения Холецкого / "	D7
LUPQR	Осуществляет пересчет QR-разложения / "	D7
MCHOL	Вычисляет R^TR-разложение Холецкого вещественной симметрической общего вида матрицы с добавлением к ней диагональной матрицы / "	D2b1
MRRRR	Вычисляет произведение двух вещественных матриц	D1b6
MURBV	Умножает ленточную матрицу на вектор	D1b4
N1RTY	Возвращает уровень серьезности последней возникшей при вызове процедур IMSL ошибки / См. описание процедуры	R3
NDAYS	Вычисляет число дней от 1 января 1900 г. до заданной даты	R
NDYIN	Вычисляет дату, отстоящую от 1 января 1900 г. на заданное число дней	R
NEQBF	Решает систему нелинейных уравнений методом секущих, применяя конечно-разностную аппроксимацию для вычисления матрицы Якоби / См. описание процедуры	F2
NEQBJ	Решает систему нелинейных уравнений методом секущих, используя для вычисления матрицы Якоби пользовательскую подпрограмму / То же	F2
NEQNF	Решает систему нелинейных уравнений, применяя модифицированный гибридный алгоритм Пауэлла и конечно-разностную аппроксимацию для вычисления матрицы Якоби / "	F2
NEQNJ	Решает систему нелинейных уравнений, применяя модифицированный гибридный алгоритм Пауэлла; для вычисления матрицы Якоби используется пользовательская подпрограмма / "	F2
NR1RB	Вычисляет 1-норму ленточной матрицы	D1b2
NR1RR	Вычисляет 1-норму матрицы	D1b2
NR2RR	Вычисляет 2-норму матрицы	D1b2

NRIRR	Вычисляет ∞-норму матрицы	D1b2
PCGRC	Решает систему линейных уравнений $Ax = b$ с симметрической положительно определенной разреженной матрицей A методом сопряженных градиентов с аппроксимацией и обратной связью / См. описание процедуры	D2b4
PERMA	Выполняет перестановки строк или столбцов матрицы	N8
PERMU	Выполняет перестановки в векторе	N8
PGOPT	Устанавливает или определяет ширину и высоту страницы	N1
PLOTP	Выводит в DOS-окно до 10 наборов данных	Q
POLRG	Оценка матричного степенного ряда	D1b7
PRIME	Выполняет разложение целого числа на простые множители / См. описание процедуры	A6c
SADD	Прибавляет к вектору скаляр / BLAS 1	D1a11
SASUM	Вычисляет сумму абсолютных значений элементов вектора / То же	D1a3a
SAXPY	Выполняет присваивание вида $y = \alpha x + y$, где x и y – векторы, а α – скаляр / "	D1a7
SCOPY	Копирует вектор / "	D1a5
SDDOTA	Вычисляет скалярное произведение векторов с аккумулятором двойной точности, передаваемым в качестве параметра / "	D1a4
SDOT	Возвращает скалярное произведение векторов / "	D1a4
SDSDOT	Вычисляет сумму скаляра и скалярного произведения векторов с аккумулятором двойной точности / "	D1a4
SGBMV	Умножает ленточную матрицу на вектор / BLAS 2 и 3	D1b4
SGEMM	Находит произведение матриц общего вида / То же	D1b6
SGEMV	Умножает матрицу общего вида на вектор / "	D1b4
SGER	Выполняет преобразования ранга 1 матрицы общего вида / "	D1b
SHPROD	Возвращает произведение Адамара / BLAS 1	D1a11
SNRM2	Возвращает евклидову норму вектора / То же	D1a3b
SPRDCT	Возвращает произведение элементов вектора / "	D1a11
SROT	Применяет плоское вращения Гивенса / BLAS 1	D1a8
SROTG	Находит плоское вращение (преобразование) Гивенса / То же	D1a8
SROTM	Применяет модифицированное плоское вращение Гивенса / "	D1a8
SROTMG	Находит модифицированное плоское вращение Гивенса / "	D1a8
SSBMV	Умножает вещественную симметрическую ленточную матрицу на вектор / BLAS 2 и 3	D1b4
SSCAL	Масштабирует вектор / BLAS 1	D1a6
SSET	Присваивает массиву скаляр / То же	D1a1
SSUB	Вычитает из скаляра вектор / BLAS 1	D1a11
SSUM	Возвращает сумму элементов вектора / То же	D1a11
SSWAP	Выполняет обмен данными векторов / "	D1a5

SSYMM	Вычисляет произведение и сумму симметрических матриц / BLAS 2 и 3	D1b6
SSYMV	Умножает вещественную симметрическую матрицу на вектор / То же	D1b4
SSYR	Выполняет преобразования ранга 1 симметрической матрицы / "	D1b
SSYR2	Выполняет преобразования ранга 2 симметрической матрицы / "	D1b
SSYR2K	Выполняет преобразования ранга 2*k* симметрической матрицы / "	D1b
SSYRK	Выполняет преобразования ранга *k* симметрической матрицы / "	D1b
STBMV	Умножает треугольную ленточную матрицу на вектор / "	D1b4
STBSV	Умножает обратную треугольную ленточную матрицу на вектор / "	D1b
STRMM	Вычисляет произведение с треугольной матрицей / "	D1b6
STRMV	Умножает треугольную матрицу на вектор / "	D1b4
STRSM	Производит вычисления с треугольной матрицей / "	D1b
STRSV	Умножает обратную треугольную матрицу на вектор / "	D1b
SUMAG	Задает и читает параметры процедур типа REAL(4) / См. описание процедуры	R
SVCAL	Умножает вектор на скаляр / BLAS 1	D1a5
SXYZ	Возвращает сумму произведений трех векторов / BLAS 1	D1a11
TDATE	Вычисляет текущую дату, которую показывают встроенные часы	R
TIMDY	Вычисляет час, минуты и секунды, которые показывают встроенные часы	R
TRNRR	Транспонирует прямоугольную матрицу	D1b3
UMACH	Устанавливает или запрашивает номер устройства В/В или передачи данных об ошибках	R3b
VERML	Возвращает версию библиотеки IMSL, операционную систему и компилятор Фортрана, для которых приспособлена библиотека, и ее серийный номер	R
WRCRL	Выводит комплексную прямоугольную матрицу по заданному формату и с заданными именами ее строк и столбцов	N1
WRCRN	Выводит комплексную прямоугольную матрицу с нумерацией ее строк и столбцов	N1
WRIRL	Выводит целочисленную прямоугольную матрицу по заданному формату и с заданными именами ее строк и столбцов	N1
WRIRN	Выводит целочисленную прямоугольную матрицу с нумерацией ее строк и столбцов	N1
WROPT	Устанавливает или запрашивает параметры, используемые при выводе матрицы	N1
WRRRL	Выводит вещественную прямоугольную матрицу по заданному формату и с заданными именами ее строк и столбцов	N1
WRRRN	Выводит вещественную прямоугольную матрицу с нумерацией ее строк и столбцов	N1

Таблица П.-2.4. Расшифровка GAMS-индексов процедур

Индекс	*Что означает*
A	Арифметика, анализ ошибок
A3c	Вещественные объекты. Двойная точность
A6c	Изменение представления. Декомпозиция, конструирование
C	Элементарные и специальные функции
C2	Возведение в степень, извлечение корней, обратные числа
C19	Другие специальные функции
D1a	Линейная алгебра. Элементарные векторные операции
D1a1	Присваивание констант
D1a2	Минимальные и максимальные компоненты
D1a3	Норма вектора
D1a3a	1-норма
D1a3b	2-норма (евклидова)
D1a3c	∞-норма
D1a4	Скалярное произведение
D1a5	Копирование или обмен данными
D1a6	Умножение вектора на скаляр
D1a7	Триада – вычисление $\alpha x + y$
D1a8	Вращение Гивенса
D1a10	Свертки
D1a11	Другие операции над векторами
D1b	Элементарные матричные операции
D1b2	Норма матрицы
D1b3	Транспонирование
D1b4	Умножение на вектор
D1b5	Сложение, вычитание
D1b6	Умножение
D1b7	Матричные полиномы
D1b8	Копирование
D1b9	Преобразование представления
D2a	Решение систем линейных уравнений. Вещественная несимметрическая матрица
D2a1	Общего вида
D2a2	Ленточная
D2a2a	Трехдиагональная
D2a3	Треугольная
D2a4	Разреженная
D2b	Решение систем линейных уравнений. Вещественная симметрическая матрица

D2b1	Произвольная
D2b1a	Неопределенная
D2b1b	Положительно определенная
D2b2	Положительно определенная ленточная
D2b4	Разреженная
D2c	Решение систем линейных уравнений. Комплексная неэрмитова матрица
D2c1	Общего вида
D2c2	Ленточная
D2c2a	Трехдиагональная
D2c3	Треугольная
D2c4	Разреженная
D2c	Решение систем линейных уравнений. Комплексная эрмитова матрица
D2d1a	Неопределенная
D2d1b	Положительно определенная
D2d2	Положительно определенная ленточная
D2d4	Разреженная
D3a	Определители. Вещественная несимметрическая матрица
D3a1	Общего вида
D3a2	Ленточная
D3a3	Треугольная
D3b	Определители. Вещественная симметрическая матрица
D3b1a	Неопределенная
D3b1b	Положительно определенная
D3c	Определители. Комплексная неэрмитова матрица
D3c1	Общего вида
D3c2	Ленточная
D3d	Определители. Комплексная эрмитова матрица
D3d1b	Положительно определенная
D3d2	Положительно определенная ленточная
D4	Собственные значения и собственные векторы
D4a	Обычная проблема
D4a1	Вещественная симметрическая матрица
D4a2	Вещественная несимметрическая матрица
D4a3	Комплексная эрмитова матрица
D4a4	Комплексная неэрмитова матрица
D4a6	Ленточная
D4b	Обобщенная проблема
D4b1	Вещественная симметрическая матрица
D4b2	Вещественная общего вида матрица

D4b4	Комплексная общего вида матрица
D4c	Дополнительные операции
D4c2b	Вычисление собственных значений и собственных векторов матрицы Хессенберга
D5	*QR*-разложение. Ортогонализация Грама – Шмидта
D6	Сингулярное разложение
D7	Пересчет матричных разложений
D9	Вырожденные, переопределенные и недоопределенные системы линейных уравнений. Обобщенное обращение
D9a1	Решение линейных систем без ограничений методом наименьших квадратов
D9b1	Решение линейных систем с линейными ограничениями методом наименьших квадратов
D9c	Решение линейных систем с ограничениями методом наименьших квадратов
F1	Решение нелинейного уравнения с одним неизвестным
F1a1	Многочлены с вещественными коэффициентами
F1a2	Многочлены с комплексными коэффициентами
F1b	Трансцендентные уравнения
F2	Системы нелинейных уравнений
J2	Интегральные преобразования. Свертки
N1	Ввод/вывод
N3	Обработка символьных данных
N5c	Поиск ключа
N6a2a	Сортировка целочисленного массива
N6a2b	Сортировка вещественного массива
N8	Перестановки в векторах и матрицах
Q	Графика
R	Вспомогательные процедуры
R1	Машинно-зависимые константы
R3	Обработка ошибок
R3b	Задание номера устройства для вывода сообщений об ошибках
S3	Анализ динамических характеристик программы

ЛИТЕРАТУРА

1. Амосов А. А., Дубинский Ю. А., Копченова Н. В. Вычислительные методы для инженеров. – М.: Высш. шк., 1994. – 544 с.
2. Бартеньев О. В. Современный Фортран. – М.: Диалог-МИФИ, 1998. – 397 с.
3. Бартеньев О. В. Visual Fortran: Новые возможности. – М.: Диалог-МИФИ, 1999. – 288 с.
4. Бартеньев О. В. Графика OpenGL: программирование на Фортране. – М.: Диалог-МИФИ, 2000. – 368 с.
5. Воеводин В. В. Линейная алгебра. – М.: Наука, 1980. – 400 с.
6. Голуб Дж., Ван Лоан Ч. Матричные вычисления. – М: Мир, 1999. – 548 с.
7. Демидович Б. П., Марон И. А. Основы вычислительной математики. – М.: Наука, 1966. – 664 с.
8. Дэнис Дж., мл., Шнабель Р. Численные методы безусловной оптимизации и решения нелинейных уравнений. – М.: Мир, 1988. – 440 с.
9. Ильин В. А., Позняк Э. Г. Аналитическая геометрия. – М.: Наука, 1981. – 232 с.
10. Каханер Д., Моулер К., Нэш С. Численные методы и математическое обеспечение. – М.: Мир, 1988. – 575 с.
11. Курош А. Г. Курс высшей алгебры. – М.: Наука, 1968. – 432 с.
12. Потемкин В. Г. Система инженерных и научных расчетов MATLAB 5.x: В 2 т. – М.: Диалог-МИФИ, 1999.
13. Рыжиков Ю. И. Программирование на Фортране PowerStation для инженеров. Практическое руководство. – СПб.: КОРОНА принт, 1999. – 160 с.
14. Турчак Л. И. Основы численных методов. М.: Наука, 1987. – 320 с.
15. Уилкинсон Дж. Алгебраическая проблема собственных значений. – М.: Наука, 1970.
16. Уилкинсон Дж., Райнш К. Справочник алгоритмов на языке АЛГОЛ. Линейная алгебра. – М.: Машиностроение, 1976. – 389 с.
17. Форсайт Дж., Моллер К. Численное решение систем линейных алгебраических уравнений. – М.: Мир, 1969.
18. Фортран 90. Международный стандарт. – М.: Финансы и статистика, 1998. – 416 с.
19. Шикин А. В., Боресков А. В. Компьютерная графика. Полигональные модели. – М.: Диалог-МИФИ, 2000. – 464 с.
20. Aird T. J., Howell B. W. IMSL Technical Report 9103, IMSL, Houston, 1991.
21. Arushanian O. B., Samarin M. K., Voevodin V. V., Tyrtyshikov E. E., Garbow B. S., Boyle J. M., Cowell W. R., Dritz K. W. The TOEPLITZ Package Users' Guide, Argonne National Laboratory, Argonne, Illinois, 1983.
22. Ashcraft C. A vector implementation of the multifrontal method for large sparse, symmetric positive definite linear systems, Technical Report ETA-TR-51, Engineering Technology Applications Division, Boeing Computer Services, Seattle, Washington, 1987.
23. Ashcraft C., Grimes R., Lewis J., Peyton B., Simon H. Progress in sparse matrix methods for large linear systems on vector supercomputers. Intern. J. Supercomputer Applic., 1(4), 1987. – P. 10-29.
24. Atchison M. A., Hanson R. J. An Options Manager for the IMSL Fortran 77 Libraries, Technical Report 9101, IMSL, Houston, 1991.

25. Barrett J. C., Healy M. J. R. A remark on Algorithm AS 6: Triangular decomposition of a symmetric matrix, Applied Statistics, 27, 1978. – P. 379-380.
26. Bjorck A. Iterative refinement of linear least squares solutions I, BIT, 7, 1967. – P. 322-337.
27. Bjorck A. Iterative refinement of linear least squares solutions II, BIT, 8, 1968. – P. 8-30.
28. Boisvert R. F., Howe S. E., Kahaner D. K., Springmann J. L. Guide to Available Mathematical Software, National Institute of Standards and Technology NISTIR 90-4237. 1990.
29. Brenner N. Algorithm 467: Matrix transposition in place [F1], Communication of ACM, 16, 1973. – P. 692-694.
30. Bunch J. R., Parlett B. N. Direct methods for solving symmetric indefinite systems of linear equations, SIAM J. Num. Anal. 8, 1972. – P. 639-655.
31. Cline A. K., Moler C. B., Stewart G. W., Wilkinson J. H. An estimate for the condition number of a matrix, SIAM Journal of Numerical Analysis, 16, 1979. – P. 368-375.
32. Cohen E. R., Taylor B. N. The 1986 Adjustment of the Fundamental Physical Constants, Codata Bulletin, Pergamon Press, New York, 1986.
33. Crowe K., Yuan-An Fan, Jing Li, Dale N., Phil S. A direct sparse linear equation solver using linked list storage, IMSL Technical Report 9006, IMSL, Houston, 1990.
34. Dongarra J. J., Bunch J. R., Moler C. B., Stewart G. W. LINPACK Users' Guide, SIAM, Philadelphia, 1979.
35. Dongarra J. J., Du Croz J., Hammarling S., Duff I. A set of level 3 basic linear algebra subprograms, ACM Transactions on Mathematical Software, 16, 1990. – P. 1-17.
36. Dongarra J. J., Du Croz J., Hammarling S., Hanson R. J. An Extended Set of Fortran basic linear algebra subprograms, ACM Transactions on Mathematical Software, 14, 1988. – P. 1-17.
37. Du Croz J., Mayes P. G., Radicati. Factorization of band matrices using Level-3 BLAS, Proceedings of CONPAR 90 VAPP IV, Lecture Notes in Computer Science, Springer, Berlin, 1990. – P. 222.
38. Duff I. S., Erisman A. M., Reid J. K. Direct Methods for Sparse Matrices, Clarendon Press, Oxford, 1986.
39. Duff I. S., Reid J. K. The multifrontal solution of indefinite sparse symmetric linear equations. ACM Transactions on Mathematical Software, 9, 1983. – P. 302-325.
40. Duff I. S., Reid J. K. The multifrontal solution of unsymmetric sets of linear equations. SIAM Journal on Scientific and Statistical Computing, 5, 1984. – P. 633-641.
41. Farebrother R. W., Berry G. A remark on Algorithm AS 6: Triangular decomposition of a symmetric matrix, Applied Statistics, 23, 1974. – P. 477.
42. Garbow B. S., Boyle J. M., Dongarra J. J., Moler C. B. Matrix eigensystem Routines: EISPACK Guide Extension, Springer-Verlag, New York, 1972.
43. George A., Liu J. W. H. Computer Solution of Large Sparse Positive-definite Systems, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
44. Gill P. E., Murray W., Wright M. Practical Optimization, Academic Press, New York, 1981.
45. Golub G. H., Van Loan C. F. Matrix Computations, Johns Hopkins University Press, Baltimore, Maryland, 1983.
46. Gregory R., Karney D. A Collection of Matrices for Testing Computational Algorithms, Wiley-Interscience, John Wiley & Sons, New York, 1969.

47. Griffin R., Redish K. A. Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, Communications of the ACM, 13, 1970. – P. 54.
48. Hageman L. A., Young D. M. Applied iterative methods, Academic Press, New York, 1981.
49. Hanson R. J. Least squares with bounds and linear constraints, SIAM Journal Sci. Stat. Computing, 7, 3, 1986.
50. Hanson R. J. A cyclic reduction solver for the IMSL Mathematics Library, IMSL Technical Report 9002, IMSL, Houston, 1990.
51. Hanson R. J., Lehoucq R., Stolle J., Belmonte A. Improved performance of certain matrix eigenvalue computations for the IMSL/MATH Library, IMSL Technical Report 9007, IMSL, Houston, 1990.
52. Healy M. J. R. Algorithm AS 6: Triangular decomposition of a symmetric matrix, Applied Statistics, 17, 1968. – P. 195-197.
53. Higham J. H. Analysis of the Cholesky decomposition of a semi-definite matrix. Reliable Numeric Computation, Oxford University Press, 1989.
54. Isnan. Appendix. IEEE 1985.
55. Kershaw D. Solution of tridiagonal linear systems and vectorization of the ICCG algorithm on the Cray-1, Parallel Computations, Academic Press, Inc., 1982. – P. 85-99.
56. Liepman D. S. Mathematical constants, in Handbook of Mathematical Functions, Dover Publications, New York, 1964.
57. Lawson C. L., Hanson R. J., Kincaid D. R. Krogh F. T. Basic linear algebra subprograms for Fortran usage, ACM Transactions on Mathematical Software, 5, 1979. – P. 308- 323.
58. Liu J. W. H. On the storage requirement in the out-of-core multifrontal method for sparse factorization. ACM Transactions on Mathematical Software, 12, 1986. – P. 249-264.
59. Liu J. W. H. A collection of routines for an implementation of the multifrontal method, Technical Report CS-87-10, Department of Computer Science, York University, North York, Ontario, Canada, 1987.
60. Liu J. W. H. The multifrontal method and paging in sparse Cholesky factorization. ACM Transactions on Mathematical Software, 15, 1989. – p. 310-325.
61. Liu J. W. H. The multifrontal method for sparse matrix solution: theory and practice, Technical Report CS-90-04, Department of Computer Science, York University, North York, Ontario, Canada, 1990.
62. Maindonald J. H. Statistical Computation, John Wiley & Sons, New York, 1984.
63. Martin R. S., Wilkinson J.W. Reduction of the symmetric eigenproblem $Ax = \lambda Bx$ and related problems to standard form, Numerische Mathematik, 11, 1968. – P. 99-119.
64. Moler C., Stewart G. W. An algorithm for generalized matrix eigenvalue problems, SIAM Journal on Numerical Analysis, 10, 1973. – P. 241-256.
65. More J., Burton G., Kenneth H. User guide for MINPACK-1, Argonne National Labs Report ANL-80-74, Argonne, Illinois, 1980.
66. Parlett B. N. The Symmetric Eigenvalue Problem, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1980.
67. Petro R. Remark on Algorithm 347: An efficient algorithm for sorting with minimal storage, Communications of the ACM, 13, 1970. – P. 624.

68. Sallas W. M., Lionti A. M. Some useful computing formulas for the nonfull rank linear model with linear equality restrictions, IMSL Technical Report 8805, IMSL, Houston, 1988.
69. Smith B. T., Boyle J. M., Dongarra J. J., Garbow B. S., Ikebe Y., Klema V. C., Moler C. B. Matrix Eigensystem Routines – EISPACK Guide, Springer-Verlag, New York, 1976.
70. Trench W. F. An algorithm for the inversion of finite Toeplitz matrices, Journal of the Society for Industrial and Applied Mathematics, 12, 1964. – P. 515-522.
71. Wilkinson J. H. A Priori Error Analyses of Algebraic Processes. – Pro. International Congress Math., 1968. – P. 629-639.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

L

LU-разложение
вычисление · 150
применение · 152

N

NaN · *См.* Не число

Q

QR-алгоритм · 325
со сдвигом · 327
QR-разложение
в вырожденной задаче наименьших квадратов · 358
в задаче наименьших квадратов · 357
восстановление матрицы *Q* · 292, 296
пересчет · 298, 302
прямоугольной матрицы · 294
с выбором главного столбца · 358
симметрической трехдиагональной матрицы · 290

А

Аккумулятор повышенной точности · 101, 143
Алгоритм корректный · 6

Б

Билинейная форма · 135

В

Ведущее уравнение · 147
Ведущий элемент · 147
полный выбор · 157
частичный выбор · 148, 153
Вектор
Гаусса · 150
единичный, нормированный · 92
координаты · 90
масштабирование · 106
перестановок · 155
столбец · 76
строка · 77
Хаусхолдера · 275
Векторы
внешнее произведение · 79
линейная комбинация · 89
линейно зависимые · 90
линейно независимые · 90
скалярное произведение · 78, 108
Вспомогательная процедура
ru_doswin · 413
Вспомогательный модуль
text_transfer · 413

Д

Двоичный порядок · 21

З

Задача· 16

И

Исчезновение порядка · 15, 70
Итерационное уточнение решения · 165, 385

К

Квадратичная форма · 83, 136
Коэффициент роста · 159, 167

Л

Лента матрицы
верхняя, нижняя · 84
ширина · 84

М

Мантисса · 21
Матрица
g3-обратная · 205
LU-разложение · 149
QR-разложение · 285
Адамара · 194
балансировка · 158
блочная · 87
блочно-столбцовая · 77
блочно-строковая · 77
верхняя треугольная · 84
виды разложений · 274
вращения · 88
вырожденная, невырожденная · 83
Гивенса · 89, 279
Гильберта · 42
главные диагональные миноры · 81

главные подматрицы · 81
диагональная · 82
диагонально доминирующая · 158
единичная, тождественная · 82
квадратичной формы · 83
квазидиагональная · 325
квазитреугольная · 325
кодиагональ · 84
ленточная · 84, 95
ленточная симметрическая · 95, 97
ленточная треугольная · 97
ленточная эрмитова · 96, 98
масштабирование · 158
минор · 81
неопределенная · 176
неотрицательно определенная · 171
нижняя треугольная · 84
обобщенное обращение · 204, 317
обратная · 82
обращение · 83
общего вида · 82, 188
определитель · 80
определитель произведения матриц · 82
ортогональная · 83
отражения · 88
отрицательно определенная · 168
персимметрическая · 254
плохо обусловленная · 161
подматрица · 81
подобная · 322
полная · 94
полный столбцовый ранг · 91
положительно определенная · 167
преобразования Гаусса · 150
преобразования подобия · 322
произведение · 80, 85, 126
простой структуры · 323
прямоугольная · 82
разложение Холецкого · 168
разреженная · 84, 99
ранг · 90
расщепление · 327
самоприсоединенная · 378
симметрическая · 83
сингулярное разложение · 274
след · 323
собственное значение · 319
собственный вектор · 319
степенной ряд · 136
Теплица · 254
трапециевидная · 85
треугольная · 201
умножение на вектор · 86, 119, 137
унитарная · 84
унитреугольная · 85
характеристическая · 319
характеристический определитель · 319
характеристическое уравнение · 319
Хаусхолдера · 89, 275
Хессенберга · 324
хорошо обусловленная · 161
циклическая · 257
число обусловленности · 161, 189, 311
элементарные преобразования · 80
эрмитова · 83
Якоби · 89
Машинная бесконечность · 20
отрицательная, положительная · 38
Машинная точность · 24
Метод
наименьших квадратов · 355
обратных итераций · 328
отражений · 276
с диагональным выбором · 176
сопряженных градиентов · 227
степенной · 323
Хаусхолдера · 289
Метод исключений Гаусса · 146
вычисление обратной матрицы · 159
вычисление определителя · 160
обратный ход · 146
прямой ход · 146
устойчивость · 159
Множитель *k*-го шага · 150
Модель · 355
параметры · 355

Н

Не число · 22
  сигнальный NaN · 37
  совершенный NaN · 37
Невязка · 147, 165
Норма вектора
  1-, 2-, ∞-норма · 91
  2-норма, евклидова · 15, 91, 112
  p-норма · 91
Норма матрицы
  1-, 2-, ∞-норма · 92, 140
  p-норма · 92
  Фробениуса · 92
Нуль-вектор · 90

О

Обобщенное имя {G} · 102
Обусловленность вычислительной задачи · 16
Ошибка
  завершающая · 55
  информационная · 55
  округления · 14
  тип или уровень · 55
  уровень серьезности · 57
Ошибка округления · 27

П

Параметры *diag*, *side*, *trans*, *uplo* · 117
Параметры *incx*, *incy* · 102
Переполнение · 15, 70
Подстановки
  обратные · 152
  прямые · 152
Предотвращение конфликтов памяти · 193
Преобразования
  Гивенса · 113, 279
  матриц · 129
  ортогональные · 93
  ранга 1 · 122
  ранга 1, 2, *k* и 2*k* · 125
Преобразования Хаусхолдера · 275
  блочные · 368
Проблема $Ax \approx b$
  без ограничений · 363
  с линейными ограничениями · 372
Пространство
  базис · 90
  нормированное · 92
Процедура IMSL
  второго уровня · 10
  главная · 10

Р

Разложение неопределенной матрицы · 176
Разложение Холецкого
  вычисление · 169
  неотрицательно определенной матрицы · 172
  пересчет · 302

С

Симметричный выбор · 173
Сингулярное разложение · 310
  в задаче наименьших квадратов · 361
Сингулярные векторы · 310
Сингулярные значения · 310
Система линейных уравнений· 356, 145
  однородных · 145
Собственные значения
  вычисление · 324
  диагональной матрицы · 323
  итерационные методы вычислений · 322
  обобщенная проблема · 321
  обычная проблема · 321
  оценка точности вычисления · 352
  полная проблема · 321
  прямые методы вычислений · 322
  свойства · 323
  частичная проблема · 321
Стандарт IEEE · 20, 37
Схема
  Горнера · 12, 137
  частичного выбора · 148

Т

Треугольники Холецкого · 168

Ф

Флоп · 11

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ 3

1. ХАРАКТЕРИСТИКИ ПРОЦЕДУР IMSL 5

1.1. Состав библиотеки 5
1.2. Вызов процедур IMSL 6
1.3. Выделение памяти 10
1.4. Соглашения об именах 10
1.5. Вычислительная сложность алгоритмов 11
1.6. Оптимизация кода 12
1.7. Учет особенностей машинной арифметики 13
1.7.1. Разрывы между вещественными числами 13
1.7.2. Ошибки округления 14
1.7.3. Устранение переполнения и исчезновения порядка 15
1.7.4. Оценка точности результата 16
1.8. Обработка ошибок 18
1.9. Соглашения, действующие при описании процедур библиотеки IMSL 18
1.10. Вспомогательная функция *ru_doswin* 19

2. ВЫЧИСЛЕНИЯ С ПЛАВАЮЩЕЙ ТОЧКОЙ 20

2.1. Двоичные представления чисел 20
2.1.1. Представление целых чисел 20
2.1.2. Вещественные числа с плавающей точкой 20
2.1.3. Виды вещественных чисел с плавающей точкой 22
2.1.4. характеристики модели представления чисел 23
2.1.5. Абсолютная и относительная ошибки 26
2.2. Исключения 27
2.3. Управление вычислениями с плавающей точкой 28
2.3.1. Значения статуса и управляющего слова 28
2.3.2. Чтение статуса 29
2.3.3. Изменение управляющего слова 30
2.4. Обработка исключений 31
2.4.1. Обработка исключений с плавающей точкой 31
2.4.2. Обработка математических исключений 33

3. ВСПОМОГАТЕЛЬНЫЕ ПРОЦЕДУРЫ 36

3.1. Машинные константы 36

3.1.1. Целые машинные константы *36*
3.1.2. Вещественные машинные константы *37*
3.1.3. Проверка на NAN *38*
3.2. Вывод результатов 39
3.2.1. Список, вызовы и параметры процедур *39*
3.2.2. Вывод прямоугольной матрицы с нумерацией строк и столбцов *41*
3.2.3. Вывод прямоугольной матрицы по заданному формату и с заданными именами строк и столбцов *43*
3.2.4. Управление выводом *46*
3.2.5. Управление размером страницы *53*
3.2.6. Задание номера устройства В/В *54*
3.3. Обработка ошибок 55
3.3.1. Виды ошибок IMSL *55*
3.3.2. Примеры ошибок IMSL *56*
3.3.3. Изменение характера реагирования на информационные ошибки *57*
3.3.4. Использование системы обработки ошибок *59*
3.4. Процедуры даты и времени 60
3.5. Задание и чтение настроек процедур IMSL 64
3.6. Некоторые специальные процедуры 67
3.6.1. Разложение числа на простые множители *67*
3.6.2. Дополнительные символьные функции *68*
*3.6.3. Вычисление SQRT(a**2 + b**2) без исчезновения порядка и переполнения* *70*
3.6.4. Математические и физические константы *71*
3.6.5. Перевод величин из одной системы единиц в другую *73*
3.7. Версия библиотеки IMSL 75

4. БАЗОВЫЕ ПРОЦЕДУРЫ ЛИНЕЙНОЙ АЛГЕБРЫ 76

4.1. Введение 76
4.2. Некоторые сведения о векторах и матрицах 76
4.2.1. Обозначение векторов и матриц *76*
4.2.2. Операции над векторами *78*
4.2.3. Операции над матрицами *79*
4.2.4. Элементарные преобразования *80*
4.2.5. Определитель и миноры матрицы *80*
4.2.6. Виды матриц *82*
4.2.7. Умножение матриц *85*
4.2.8. Умножение матрицы на вектор *86*

4.2.9. Блочные матрицы *87*
4.2.10. Двумерные матрицы вращения и отражения *88*
4.2.11. Линейная независимость и базис *89*
4.2.12. Ортогональность векторов *90*
4.2.13. Ранг матрицы *90*
4.2.14. Векторные нормы *91*
4.2.15. Матричные нормы *92*
4.2.16. Нормы и ортогональные преобразования *93*
4.3. Представление матриц в программах 93
4.3.1. Представление всех элементов матрицы *94*
4.3.2. Ленточные матрицы *95*
4.3.3. Ленточные симметрические матрицы *95*
4.3.4. Ленточные эрмитовы матрицы *96*
4.3.5. Ленточные треугольные матрицы *97*
4.3.6. Кодиагональное представление ленточных симметрических матриц *97*
4.3.7. Кодиагональное представление ленточных эрмитовых матриц *98*
4.3.8. Разреженные матрицы *99*
4.4. Базовые процедуры линейной алгебры уровня 1 101
4.4.1. Замечания для программиста *101*
4.4.2. Список процедур уровня 1 ..... *103*
4.4.3. Присваивание вектору скаляра *104*
4.4.4. Копирование вектора *105*
4.4.5. Масштабирование вектора *106*
4.4.6. Умножение вектора на скаляр *106*
4.4.7. Сумма вектора и скаляра *106*
4.4.8. Вычитание из скаляра вектора *107*
4.4.9. Присваивание вида $y = \alpha x + y$ *107*
4.4.10. Обмен данными векторов *107*
4.4.11. Скалярное произведение векторов *108*
4.4.12. Скалярное произведение векторов с аккумулятором двойной точности *108*
4.4.13. Сумма скаляра и скалярного произведения векторов с аккумулятором двойной точности *109*
4.4.14. Скалярное произведение векторов с аккумулятором двойной точности, передаваемым в качестве параметра *109*
4.4.15. Произведение Адамара *110*
4.4.16. Сумма произведений трех векторов *110*
4.4.17. Сумма элементов вектора *111*

4.4.18. Сумма абсолютных значений элементов вектора ... *111*
4.4.19. Евклидова, или 2-норма вектора ... *112*
4.4.20. Произведение элементов вектора ... *112*
4.4.21. Индекс минимального элемента вектора ... *112*
4.4.22. Индекс максимального элемента вектора ... *112*
4.4.23. Индекс минимального по модулю элемента вектора ... *112*
4.4.24. Индекс максимального по модулю элемента вектора ... *113*
4.4.25. Построение плоского вращения Гивенса ... *113*
4.4.26. Применение плоского вращения Гивенса ... *114*
4.4.27. Построение быстрого вращения Гивенса ... *115*
4.4.28. Применение быстрого вращения Гивенса ... *115*
4.5. Базовые процедуры линейной алгебры уровней 2 и 3 ... 116
4.5.1. Замечания для программиста ... *116*
4.5.2. Список процедур уровней 2 и 3 ... *118*
4.5.3. Умножение матрицы общего вида на вектор ... *119*
4.5.4. Умножение ленточной матрицы на вектор ... *120*
4.5.5. Умножение эрмитовой матрицы на вектор ... *121*
4.5.6. Умножение эрмитовой ленточной матрицы на вектор ... *121*
4.5.7. Умножение вещественной симметрической матрицы на вектор ... *121*
4.5.8. Умножение симметрической ленточной матрицы на вектор ... *121*
4.5.9. Умножение треугольной матрицы на вектор ... *121*
4.5.10. Умножение треугольной ленточной матрицы на вектор ... *122*
4.5.11. Умножение обратной треугольной матрицы на вектор ... *122*
4.5.12. Умножение обратной треугольной ленточной матрицы на вектор ... *122*
4.5.13. Преобразования ранга 1 матрицы общего вида ... *122*
4.5.14. Преобразования ранга 1, 2, k и 2k симметрической матрицы ... *123*
4.5.15. Преобразования ранга 1, 2, k и 2k эрмитовой матрицы ... *125*
4.5.16. Произведение матриц общего вида ... *126*
4.5.17. Произведение с симметрической матрицей ... *127*
4.5.18. Произведение с эрмитовой матрицей ... *127*
4.5.19. Произведение с треугольной матрицей ... *127*
4.5.20. Вычисления с треугольной матрицей ... *128*
4.6. Дополнительные операции над векторами и матрицами ... 128
4.6.1. Процедуры, которые могут быть заменены встроенными возможностями Фортрана ... *128*
4.6.2. Преобразование матриц ... *129*

4.6.3. Билинейная и квадратичная формы *135*
4.6.4. Оценка матричного степенного ряда *136*
4.6.5. Умножение ленточной матрицы на вектор *137*
4.6.6. Сложение ленточных матриц *138*
4.6.7. Вычисление нормы матрицы *140*
4.6.8. Вычисление расстояния между точками *142*
4.6.9. Вычисления с повышенной точностью *143*

5. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ 145

5.1. Постановка задачи 145
5.2. Метод исключения Гаусса 146
5.2.1. Идея метода исключения *146*
5.2.2. LU-разложение матрицы *149*
5.2.3. Повышение качества разложения *153*
5.2.4. Устойчивость метода исключений Гаусса *159*
5.2.5. Некоторые приложения метода исключений Гаусса *159*
5.2.6. Число обусловленности матрицы *161*
5.2.7. Оценка точности и уточнение решения *165*
5.2.8. Системы с разреженными матрицами *166*
5.3. Линейные системы с симметрической матрицей 167
5.3.1. Положительно определенные системы *167*
5.3.2. Получение разложения Холецкого *168*
5.3.3. Неотрицательно определенные системы *171*
5.3.4. Симметрические неопределенные системы *176*

6. ПРОЦЕДУРЫ IMSL ДЛЯ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ И ВЫЧИСЛЕНИЯ ОПРЕДЕЛИТЕЛЕЙ 181

6.1. Введение 181
6.2. Системы линейных уравнений с вещественной несимметрической матрицей 182
6.2.1. Список процедур с вещественной несимметрической матрицей *182*
6.2.2. Возможные ошибки процедур *185*
6.2.3. Параметры процедур с вещественными несимметрическими и комплексными неэрмитовыми матрицами *186*
6.2.4. Процедуры с вещественной несимметрической общего вида матрицей *188*
6.2.5. Пример описания процедуры с вещественной несимметрической ленточной матрицей *197*

6.2.6. Процедуры с вещественной несимметрической трехдиагональной матрицей *198*
6.2.7. Процедуры с вещественной треугольной матрицей *201*
6.2.8. Процедуры с вещественной несимметрической разреженной матрицей *206*
6.3. Процедуры с вещественной симметрической матрицей 211
6.3.1. Список процедур *211*
6.3.2. Возможные ошибки процедур с вещественной симметрической матрицей *215*
6.3.3. Параметры процедур с вещественной симметрической и комплексной эрмитовой матрицами *216*
6.3.4. Процедуры с вещественной симметрической положительно определенной матрицей *216*
6.3.5. Процедуры с вещественной симметрической положительно определенной ленточной матрицей *221*
6.3.6. Процедуры с вещественной симметрической положительно определенной разреженной матрицей *227*
6.3.7. Разложение Холецкого симметрической неотрицательно определенной матрицы *241*
6.3.8. Разложение Холецкого симметрической неотрицательно определенной матрицы с применением симметричного выбора *243*
6.3.9. Процедуры с вещественной симметрической неопределенной матрицей *247*
6.3.10. Разложение симметрической матрицы с добавлением к ней диагональной матрицы *251*
6.3.11. Комментарий *254*
6.4. Системы линейных уравнений с вещественной матрицей Теплица 254
6.5. Системы линейных уравнений с комплексной матрицей 255
6.5.1. Процедуры с комплексной неэрмитовой матрицей *256*
6.5.2. Процедуры с комплексной эрмитовой матрицей *263*
6.6. Вычисление определителей 269
6.6.1. Список процедур *269*
6.6.2. Параметры процедур *271*
6.6.3. Описание подпрограммы LFDRG *271*
6.6.4. Примеры вычисления определителей *272*
7. МАТРИЧНЫЕ РАЗЛОЖЕНИЯ И ИХ ПЕРЕСЧЕТ 274
7.1. Введение 274
7.2. Преобразования Хаусхолдера и Гивенса 275

7.2.1. Преобразование Хаусхолдера *275*
7.2.2. Преобразование Гивенса *279*
7.3. Методы вычисления *QR*-разложения 285
7.3.1. QR-разложение прямоугольной матрицы с применением преобразования Хаусхолдера *285*
7.3.2. QR-разложение симметрической матрицы *288*
7.4. *QR*-разложение процедурами IMSL 294
7.4.1. QR-разложение прямоугольной матрицы *294*
7.4.2. Восстановление ортогональной матрицы Q изQR-разложения прямоугольной матрицы *296*
7.5. Пересчет матричных разложений 297
7.5.1. Алгоритм пересчета QR-разложения *297*
7.5.2. Перечень процедур *302*
7.5.3. Пересчет разложения Холецкого *303*
7.5.4. Пересчет QR-разложения *308*
7.6. Вычисление сингулярного разложения 310
7.6.1. Понятие сингулярного разложения *310*
7.6.2. Сингулярное разложение комплексной матрицы *312*
7.6.3. Сингулярное разложение вещественной матрицы *315*
7.6.4. Вычисление обобщенной обратной матрицы *316*
8. СОБСТВЕННЫЕ ЗНАЧЕНИЯ И СОБСТВЕННЫЕ ВЕКТОРЫ 319
8.1. Подходы к решению проблемы собственных значений 319
8.1.1. Понятия собственного значения и собственного вектора *319*
8.1.2. Преобразования подобия *322*
8.1.3. Некоторые свойства собственных значений и собственных векторов *323*
8.1.4. Собственные значения и собственные векторы вещественной симметрической и комплексной эрмитовой матрицы *324*
8.2. Процедуры IMSL для вычисления собственных значений ... 330
8.2.1. Организация материала *330*
8.2.2. Обычная проблема собственных значений *330*
8.2.3. Обобщенная проблема собственных значений *346*
8.3. Оценка точности решения проблемы собственных значений 352
8.3.1. Список, вызовы, параметры и ошибки процедур *352*

9. РЕШЕНИЕ ПРЯМОУГОЛЬНЫХ СИСТЕМ $Ax \approx b$ с ОГРАНИЧЕНИЯМИ И БЕЗ НИХ 355
9.1. Метод наименьших квадратов 355
9.1.1. Постановка задачи *355*
9.1.2. Применение QR-разложения в методе наименьших квадратов *356*
9.1.3. Вырожденная задача наименьших квадратов *358*
9.1.4. Решение задачи наименьших квадратов с помощью сингулярного разложения *361*
9.2. Подпрограммы IMSL для решения переопределенной и недоопределенной проблемы $Ax \approx b$ 362
9.2.1. Перечень подпрограмм *362*
9.2.2. Проблема $Ax \approx b$ без ограничений *363*
9.3. Проблема $Ax \approx b$ с линейными ограничениями *372*
10. ПРОЦЕДУРЫ И ОПЕРАЦИИ БИБЛИОТЕКИ IMSL FORTRAN 90 MP 375
10.1. Введение 375
10.2. Операции библиотеки IMSL 90 378
10.3. Функции библиотеки IMSL 90 379
10.4. Решение систем линейных уравнений 381
10.4.1. Процедура LIN_SOL_GEN для линейных систем с матрицей общего вида *381*
10.4.2. Процедура LIN_SOL_SELF для линейных систем с самоприсоединенной матрицей *387*
10.4.3. Процедура LIN_SOL_TRI для линейных систем с трехдиагональной матрицей *391*
10.5. Сингулярные и собственные значения 394
10.5.1. Процедура LIN_SVD для вычисления сингулярного разложения *394*
10.5.2. Процедура LIN_EIG_SELF для вычисления собственных значений самоприсоединенной матрицы *397*
10.5.3. Процедура LIN_EIG_GEN для вычисления собственных значений матрицы общего вида *399*
10.5.4. Процедура LIN_GEIG_GEN для обобщенной проблемы собственных значений *403*
10.6. Решение линейных систем методом наименьших квадратов 406

10.6.1. Процедура LIN_SOL_LSQ для решения прямоугольных линейных систем методом наименьших квадратов *406*
10.6.2. Процедура LIN_SOL_SVD, использующая сингулярное разложение для решения прямоугольных линейных систем *410*

ПРИЛОЖЕНИЕ 1. Вывод русского текста в DOS-окно 413

ПРИЛОЖЕНИЕ 2. Список представленных в пособии процедур математической библиотеки IMSL 415

ЛИТЕРАТУРА 433

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 437