Д.К. Андрейченко, Ю.В. Чурсова, В.В. Кононов, Д.В. Супрун

ОСНОВЫ РАБОТЫ В СРЕДЕ MATLAB

Саратов 2012

ПРЕДИСЛОВИЕ

При выполнении научных и инженерных расчетов требуется обрабатывать достаточно большие объемы числовой информации в форме действительных или комплексных чисел с плавающей точкой и применять к ним разнообразные численные методы вычислительной математики. В этом случае традиционно используется ряд программных средств, основанных на расширенном представлении и применении матричных операций над числовыми матрицами. К числу подобных систем компьютерной математики относятся, прежде всего, пакеты MATLAB, Scilab, и Octave. Разработанный фирмой MathWork Inc пакет MATLAB (MATrix LABoratory – «матричная лаборатория»), является коммерческим, обладает наибольшими возможностями и стал фактически стандартом для использования в образовательных и научно – исследовательских учреждениях. Пакеты Octave (GNU) и Scilab (INRIA) являются свободно распространяемыми. Все указанные системы используют сходный интерпретирующий входной язык, основными типами данных для которого являются матрицы и векторы.

MATLAB содержит обширный набор средств для решения следующих типов задач:

- Применение численных методов линейной алгебры над плотно заполненными матрицами: численное решение нелинейных уравнений, факторизация и обращение матриц, нахождение собственных значений и векторов.
- Численные методы линейной алгебры для работы с разреженными матрицами: численное решение линейных уравнений, факторизация разреженных матриц, решение частичной проблемы собственных значений и прочие.
- Манипуляции с алгебраическими полиномами.
- Решение нелинейных уравнений, систем нелинейных уравнений и разнообразных задач оптимизации.
- Интерполяция и аппроксимация функций одной или нескольких действительных переменных, а также аппроксимация экспериментальных данных.
- Численное интегрирование.
- Быстрое преобразование Фурье.
- Численное решение задач Коши и краевых задач для систем обыкновенных дифференциальных уравнений, численное решение обыкновенных дифференциальных уравнений с запаздывающим аргументом, дифференциально–алгебраических уравнений и разностных уравнений.

Стандартные средства системы MATLAB позволяют легко выполнять построение разнообразных графиков, диаграмм, и обеспечивают ви-

зуализацию результатов вычислений. Кроме того, многочисленные пакеты расширений обеспечивают решение следующих задач:

- Математическое моделирование краевых задач математической физики для уравнений в частных производных. В частности, на основе метода конечных элементов.
- Решение задач математической статистики.
- Математическое моделирование и анализ систем автоматического управления.
- Моделирование и идентификация динамических систем.
- Выполнение символьных вычислений и преобразований.
- Решение задач, связанных с системами искусственного интеллекта на основе аппарата нейронных сетей, нечетких множеств и т.д.
- Обработка сигналов и изображений.

Различные версии MATLAB функционируют под управлением всех современных операционных систем: UNIX / Linux, Windows, Mac OS.

К числу достоинств MATLAB можно отнести достаточно высокую производительность старших версий (версии 2006 и выше), которая, в частности, достигается за счет технологии прекомпиляции интерпретатором исходного кода. Кроме того, в MATLAB применяются оптимизированные математические библиотеки, которые используют технологии параллельных вычислений, что позволяет получить значительный прирост производительности на многоядерных и многопроцессорных рабочих станциях.

Также система обладает тщательно продуманными и хорошо документированными так называемыми «внешними интерфейсами» для подключения к ней разделяемых библиотек (динамически загружаемые библиотеки) для импорта функций, разработанных на языках C/C++/Fortran. С другой стороны, пакет MATLAB позволяет с помощью внутренних средст генерировать интерфейсы для вызова функций, разработанных в среде MATLAB. Таким образом, помещенные в разделяемые библиотеки функции MATLAB можно вызывать из других языков программирования (C/C++/C#).

При наличии всех описаных возможностей среда MATLAB является одним из самых мощных и универсальных инструментов компьютерной математики, находящих применение в самых широких областях науки.

1. Элементы языка MATLAB

1.1. Основные типы данных

Типы данных в MATLAB называются классами. Базовым классом данных является многомерный массив (гиперматрица). Ее частными случаями являются скаляр, обычная двумерная матрица и одномерный массив – строчная или столбцевая матрица. Нумерация индексов всегда начинается с единицы. При этом матрица размером 1×1 формально трактуется как скаляр, столбцевой вектор размерности N трактуется как матрица размером $N \times 1$, а строчный вектор размерности N трактуется как матрица размерности $1 \times N$. Иерархия типов данных MATLAB представлена на рис.1.

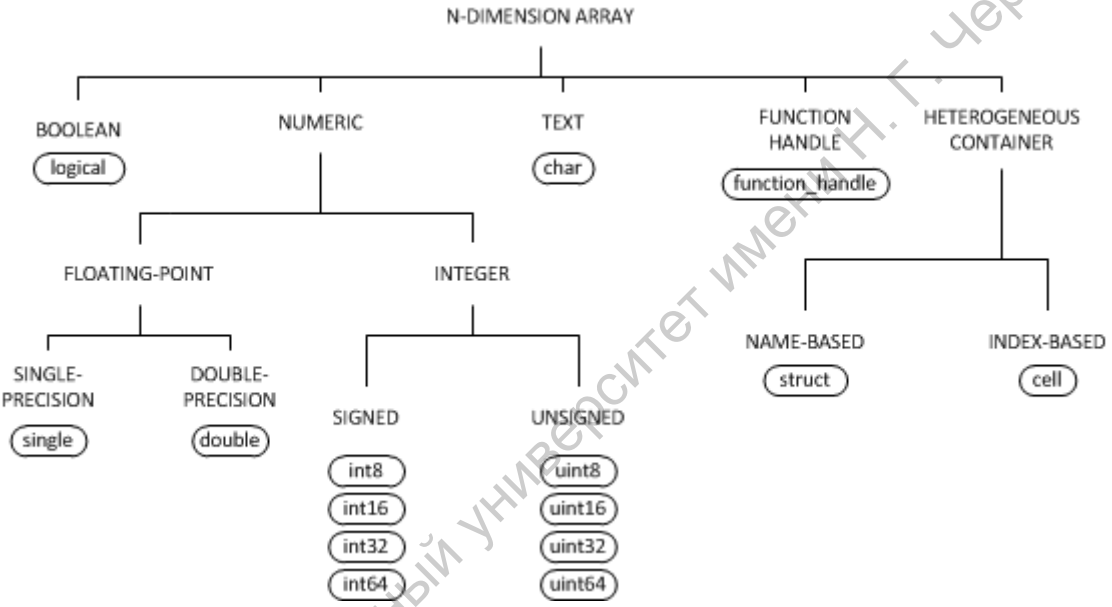


Рис.1.

Целые:

int8, int16, int32, int64 – знаковые
uint8, uint16, uint32, uint64 – беззнаковые.

Вещественные:

single, double – одинарной и двойной точности.

Для создания переменных любого из этих типов можно:

- Преобразовать скалярное значение к нужному типу и поместить в переменную

```
x=int64(7);
y=single(3.33);
```

- Воспользоваться функцией zeros, указав ей в качестве последнего аргумента строку символов с именем типа (класса)

```
A=zeros(2,2,'int64');
B=zeros(2,2,'single');
X=zeros(2,2,'double');
```

Заметим, что стандартная функция class(A) возвращает строку символов, содержащую наименование типа (класса) переменной A.

По правилам, принятым в системе MATLAB, действительные и мнимые части числовых матриц хранятся в отдельных числовых массивах. Поэтому после выполнения операции

```
B(1,1)=7+3i;
```

к матрице B будет добавлен массив размерности 2x2 из действительных чисел одинарной точности для хранения мнимых частей матрицы B. Аналогично, после выполнения операции

```
X(1,1)=7+3i;
```

к матрице X будет добавлен массив размерности 2x2 из действительных чисел одинарной точности для хранения мнимых частей матрицы X.
Комманда

```
whos X
```

распечатывает детальную информацию о переменной X.

1.2. Нечисловые типы данных

Логические значения.

Возвращаются логическими операциями и некоторыми функциями MATLAB. В соответствующих элементах матриц, в частности, в скалярах, логические TRUE и FALSE кодируются посредством 0 и 1.

Символы, строки и массивы символов.

Например, отдельно взятый символ ‘h’ можно поместить в переменную hChar в рабочем пространстве MATLAB следующим образом:

```
hChar='h';
```

Заметим, что MATLAB трактует сивольный тип (класс с именем char) как целочисленное двухбайтное значение, эквивалентное символу UNICODE.

Строка

```
str='Hello';
```

трактуется как строчный вектор (матрица 1x5) из 5 двухбайтных символов.

Переменная

```
Animals= ['Dog      ';…
          'Cat      ';…
          'Bull     ';…
          'Elephant'];
```

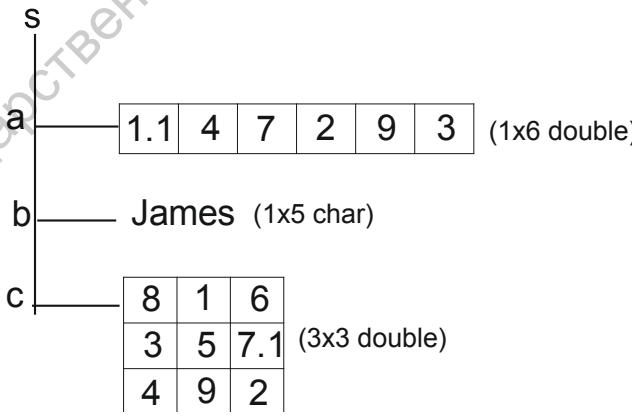
будет толковаться как массив символов (матрица) размером 4x8; при этом все строки должны иметь одинаковую длину. Тот же результат легко получается при помощи стандартной функции char:

```
Animals=char('Dog','Cat','Bull','Elephant');
```

Доступ к элементам подобных массивов реализуется обычным образом.

1.3. Структуры

С точки зрения MATLAB, структура состоит из некоторого количества контейнеров данных, называемых полями, и каждое поле содержит в себе массив данных некоторого типа, свойственного системе MATLAB. При создании структуры ее полям присваиваются имена. На рисунке ниже показана структура с тремя полями: a, b, c.



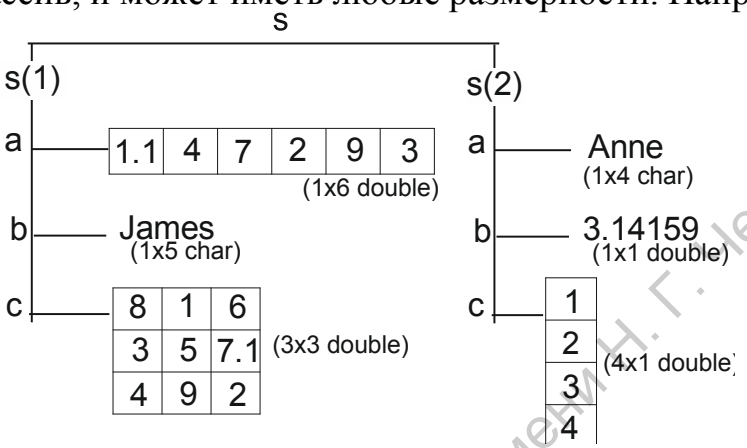
Создать структуру можно следующим образом:

```
S.a=[1 4 7 2 9 3];
S.b='James';
S.c=[8,1,6;3,5,7;4,9,2];
```

Либо при помощи встроенной функции Struct:

```
S=struct('a', [1 4 7 2 9 3], 'b', 'James', 'c',
     [8,1,6;3,5,7;4,9,2]);
```

Подобно всем остальным типам данных MATLAB, структура представляет собой массив, и может иметь любые размерности. Например:



Создать подобную структуру можно следующим образом:

```
S(1).a=[1 4 7 2 9 3];
S(1).b='James';
S(1).c=[8,1,6;3,5,7;4,9,2];
S(1).a='Anne';
S(1).b=3.1416;
S(1).c=1:7';
```

Заметим, что одноименным полям разных элементов массива с типом элементов «структура» могут соответствовать разные типы данных. Заметим также, что подобную структуру данных можно создать и следующим образом:

```
s=struct('a',{[1 4 7 2 9 3], 'Anne'},…
         'b',{'James',3.1416},…
        'c', {[8,1,6;3,5,7;4,9,2], 1:7'});
```

При помощи фигурных скобок в MATLAB конструируются так называемые массивы ячеек, которые зачастую передаются в качестве параметров функциям (массивы ячеек мы рассмотрим ниже). Поялми элементов массива структур также могут быть структуры.

Доступ к элементам структур организуется следующим образом:

`s(1).a` – доступ к полю a элемента массива типа структура;

`s.a` – доступ ко всем полям a массива структур.

1.4. Массивы ячеек (cell arrays)

Массивы ячеек представляют собой набор контейнеров, называемых ячейками, в которых могут храниться данные различного типа.

Для создания массива ячеек обычно используют фигурные скобки:

```
A={[1,4,3;0,5,8;7,2,9],'AnneSmith';3+7i,1:3}
```

Тот же самый результат может быть достигнут в результате операции последовательного присваивания значений элементам массива:

```
A(1,1)={[1,4,3;0,5,8;7,2,9]};
A(1,2)={'Anne Smith'};
A(2,1)={3+7i};
A(2,2)={1:3};
```

Для доступа к данным, хранимым в массиве ячеек, обычно используется список индексов, заключенных в фигурные скобки:

```
A{2,1}
ans=3.0000 + 7.0000i
```

2. Операции над данными

2.1. Операции над числовыми скалярами

Оператор присваивания в MATLAB имеет традиционный вид.

< переменная > = < выражение >

При этом тип переменной динамически изменяется в зависимости от типа выражения в правой части оператора присваивания.

Рассмотрим базовые операции над скалярами в MATLAB.

```
c = a+b;  % сложение
c = a-b;  % вычитание
c = a*b;  % умножение
c = a/b;  % деление
c = a^2;  % возведение переменной в квадрат
c = a^0.5; % извлечение квадратного корня
```

Кроме действительных, MATLAB также поддерживает работу с комплексными скалярами.

```
c = 3+i; % комплексный скаляр
```

Здесь i – зарезервированное имя мнимой единицы. В MATLABв качестве зарезервированного имени мнимой единицы также используется буква j.

Комплексный скаляр также можно задать следующим образом.

```
c = complex(3,i); % аналогично записи c = 3+i
```

При работе с комплексными числами используются специальные функции:

real(x) – взятие действительной части комплексного числа x;
imag(x) – взятие мнимой части комплексного числа x;
abs(x) – вычисление абсолютного значения комплексного числа x;
conj(x) – вычисление комплексно-сопряженного числа x;
angle(x) – вычисление аргумента комплексного числа x;
isreal(x) –определяет, является ли скаляр действительным.

Основные математические функции MATLABпредставлены в таблице.

sqrt(x)	вычисление квадратного корня
exp(x)	возведение в степень числа e
pow2(x)	возведение в степень числа 2
log(x)	вычисление натурального логарифма
log10(x)	вычисление десятичного логарифма
log2(x)	вычисление логарифма по основанию 2
sin(x)	синус угла x, заданного в радианах
cos(x)	косинус угла x, заданного в радианах
tan(x)	тангенс угла x, заданного в радианах
cot(x)	котангенс угла x, заданного в радианах
asin(x)	арксинус
acos(x)	арккосинус
atan(x)	арктангенс
round(x)	округление до ближайшего целого
fix(x)	усечение дробной части числа
floor(x)	округление до меньшего целого
ceil(x)	округление до большего целого
mod(x)	остаток от деления с учётом знака
sign(x)	знак числа
factor(x)	разложение числа на простые множители
isprime(x)	истинно, если число простое
rand	генерация псевдослучайного числа с равномерным законом распределения
randn	генерация псевдослучайного числа с

	нормальным законом распределения
abs(x)	вычисление модуля числа

Некоторые системные переменные:

i или j– мнимая единица;
pi – число π;
eps – погрешность операций над числами с плавающей точкой ($\approx 2^{-52}$)
realmin – наименьшее число с плавающей точкой ($\approx 2^{-1022}$)
realmax – наибольшее число с плавающей точкой ($\approx 2^{1024}-1$)
inf – значение машинной бесконечности
ans – переменная, хранящая результат последней операции и обычно вызывающая его отображение на экране дисплея
NaN – указание на нечисловой тип данных.

Вышеперечисленные переменные в момент запуска MATLAB принимают указанные значения, однако они могут быть изменены в ходе выполнения программы.

2.2. Основные команды среды MATLAB

Значения переменных сохраняются интерпртатором в течение сессии работы с MATLAB. Для очистки памяти и уничтожения созданных переменных используется команда

Edit ->ClearWorkspace

Для того, чтобы очистить командное окно используется команда

Edit ->ClearCommandWindow

Стойжецелью служат следующие комманды:

Clear – очистка всего рабочего пространства

ClearXYZ–очистка переменных X, Y, Z

Для записи значений переменных в дисковые файлы служат комманды:

Savefname – рабочая область всех переменных записывается бинарного формата fname.mat

SavefnameX - запись в файл лишь значений переменной X

SavefnameXYZ – запись в файл лишь значений переменных X, Y, Z

Аналогично, комманда

Loadfname – считывает все переменные из файла fname.mat

LoadXYZ – считывает из файла только переменные X, Y, Z

Заметим, что все указанные комманды по правилам интерпретатора MATLAB можно выполнить и в функциональной форме. Например:

Save(‘fname’, ‘X’, ‘Y’, ‘Z’)

Здесь аргументами вызываемой функции являются строки символов, соответствующих именам дисковых файлов и переменных MATLAB.

2.3. Скрипты системы MATLAB

Программы (скрипты) системы MATLAB представляют собой последовательность команд, помещенных в обычные текстовые файлы с расширением *.m. Интерпретатор MATLAB читает файл *.m и поочередно выполняет содержащиеся в нем команды. Чтобы выполнить скрипт, достаточно набрать его имя (т.е. имя соответствующего *.m файла) в командном окне MATLAB. Для выбора каталога (папки, директории), где будет происходить поиск исполняемых скриптов, достаточно в меню MATLAB воспользоваться полем ввода CurrentFolder или добавить нужную папку к числу тех, где выполняется поиск скриптов, при помощи меню File ->SetPath.

2.4. Формирование числовых матриц

В системе MATLAB матрицы могут быть сформированы следующим образом:

- посредством явного ввода элементов матрицы. Например:
```
A = [10, 2, 3; 4, 10, 6; 7, 8, 10] % элементы
% строки отделяются запятой, а сами строки
% символом ;
B = [1 37] % строковй вектор
```
- посредством чтения элементов матриц из дисковых файлов;
- посредством вызова встроенных функций;
- посредством вызова пользовательских функций.

Обращение к отдельным элементам матрицы осуществляется указанием в круглых скобках списка индексов, разделенных запятой. Для векторов достаточно указать один индекс, а для многомерных массивов (гиперматриц) потребуется указать несколько индексов. При извлечении элементов из матриц выход индекса за пределы изменения диапазона генерирует ошибку.

Пример:

```
x = A(1, 1) + A(3, 2) * A(2, 3);
A(2, 2) = x ^ 3;
t = A(4, 1) % ошибка
```

В системе MATLAB матрицы хранятся по столбцам, т.е. с ростом адреса оперативной памяти индекс столюца изменяется быстрее, чем индекс строки. С учетом этого, к элементу A(2, 2) матрицы Аразмерности 3x3 можно обратиться посредством одного индекса

```
A(5) = x ^ 3;
```

При присваивании значений элементам матрицы выход индекса за пределы максимального размера автоматически увеличит размер матрицы:

```
A(4, 4) = 7; % изменит размер на 4x4
```

При формировании матриц и векторов часто используется оператор:

```
<целое 1> = <целое 2>
```

Оператор такого рода формирует строчный вектор, элементы которого принимают значения от <целого 1> до <целого 2> с шагом 1.

В более общей форме, оператор

```
a : b : c
```

приведет к формированию числового вектора, первый элемент которого равен a, второй a+b, третий a+2b и т.д., а последний – не больше c при b>0 и не меньше c при b<0.

Операция сцепления (конкатенации) матриц. Пусть матрицы A и B имеют одинаковое число строк, а число столбцов в них соответственно M и N. Операция

```
[AB] или [A, B]
```

создает матрицу, в которой первые M столбцов совпадают со столбцами матрицы A, а оставшиеся N столбцов совпадают со столбцами матрицы B.

Пусть матрицы A и B имеют одинаковое число столбцов, а число строк в них соответственно M и N. Операция

```
[A;B]
```

создает матрицу, в которой первые M строк совпадают со строками матрицы A, а оставшиеся N строк совпадают со строками матрицы B.

Выбор всех эементов столбца или строки матрицы

```
A(:, <номер столбца>)
```

указывает на все элементы соответствующего столбца, а

```
A(<номер строки>, :)
```

указывает на все элементы соответствующей строки.
Например,

```
A(:, 1) = 0; % обнулит все элементы первого столбца
A(1, :) = 0; % обнулит все элементы первой строки
```

Специальное значение [] означает пустой объект.

```
A(:, 1) = []; % удаляет из матрицы элементы первого
              % столбца и уменьшает ее размерность
```

```
A(1, :) = []; % удаляет из матрицы элементы первой
              % строки и уменьшает ее размерность
A(1 : 2: 9) = []; % приведет к удалению элементов
                  % A(1, 1), A(3, 1), A(2, 2), A(3,3)
                  % вматрице 3x3 и приведет к ее
                  % преобразованию в одномерный вектор
```

Специальное значение end кодирует наибольшее значение соответствующего индекса матрицы. Например,

```
A(end, 1) = 0; %последний элемент первого столбца
A(1, end) = 0; %последний элемент первой строки
A(end + 1, 1) = 7; %вызовет увеличение числа строк
                   % матрицы на 1
A(1, end + 1) = 3; %вызовет увеличение
                   % числа столбцов матрицы на 1
```

Если A – прямоугольная матрица, а I, J–вектора соответствующих размерностей с целочисленными элементами, то запись

```
A(I, J)
```

указывает на следующее подмножество элементов матрицы A:

```
A(I(1), J(1))     A(I(1), J(2))     …   A(I(1), J(end))
A(I(2), J(1))     A(I(2), J(2))     …   A(I(2), J(end))
. . . . . . . . . . . . . . . . . . . . . . . . . . . . .
A(I(end), J(1))   A(I(end), J(2))   …   A(I(end), J(end))
```

При формировании матриц часто используются следующие функции:

```
zeros(M, N);
zeros(M, N, …);
zeros([M, N, …])
```

Выделяет память под матрицу MxN и заполняет ее нулями;
Вызов функции:

```
eye(M, N)
```

Создает матрицу размерности MxN, на главной диагонали которой расположены единицы, а остальные элементы обнуляются;
Функция:

```
ones(M, N)
```

Создает матрицу размерности MxN, состоящую из единиц;
Вызов:

```
size(A)
```

Возвращает вектор, содержащий размеры матрицы Апо каждому ее измерению;

2.5. Арифметические операции над матрицами и массивами

Операции над матрицами определены в соответствии с правилами линейной алгебры, а операции над массивами выполняются поэлементно. Чтобы различать эти операции, в операциях над массивами присутствует точка. Операции сложения и вычитания над матрицами и массивами дают одинаковый результат.

- `A + B` – матрицы должны быть одинакового размера за исключением случая, когда одна из них – скаляр; скаляр добавляется ко всем элементам второго операнда;
- `A – B` – матрицы должны быть одинакового размера за исключением случая, когда одна из них – скаляр; скаляр вычитается из всех элементов второго операнда либо все элементы второго операнда вычитаются из скаляра;
- `A * B` – умножение матриц; число столбцов первого сомножителя должно быть равно числу строк второго; на скаляр умножаются все элементы сомножителя;
- `A. * B` – поэлементное перемножение двух массивов одинакового размера; на скаляр умножаются все элементы массива;
- решение системы линейных уравнений
  `AX = B`
  возвращается операцией левого деления
  `X = A \ B` (для квадратных невырожденных матриц $A^{-1}B$)
  Решение системы линейных уравнений
  `YA = B`
  возвращается операцией правого деления
  `Y = A/B` (для квадратных невырожденных матриц AB^{-1})

Если A – квадратная невырожденная матрица, система линейных уравнений решается методом Гаусса. Если A – прямоугольная матрица, то находится X, обеспечивающий минимум невязки ||AX – B|| ->min либо ||YA – B|| ->min на основе соответствующего варианта метода наименьших квадратов.

- “Левое” деление массивов:
  `A.\ B`

возвращает матрицу с элементами B(k,j)/A(k,j); матрицы должны быть одинаковых размеров за исключением случая, когда одна из них превращается в скаляр. Тот же результат обеспечивается операцией:

- "Правое" деление массивов
  `B./ A`
- Степень матрицы
  `A^p`
  Если p – целое положительное число, то степень матрицы вычисляется посредством ее перемножения на себя; если p – целое отрицательное число, то в перемножении участвует обратная матрица. Для других значений p степень матрицы, подобно другим функциям от матриц, вычисляется на основе анализа спектра собственных значений и принадлежащих им собственных векторов.
- Степень массива
  `A.^B`
  Результатом является матрица с элементами A(k, j)^B(k, j). Матрицы должны быть одинакового размера за исключением случаев, когда одна из них вырождается в скаляр.
- `A'` – транспонирование действительных матриц и эрлитово сопряжение комплексных матриц. При эрлитовом сопряжении транспонирование дополняется комплексным сопряжением.
- `A.'` – транспонирование массива. Для комплексных матриц комплексное сопряжение их элементов не выполняется.

При выполнении операций A+B, A-B, A*B, A', A.' используются оптимизированные версии свободно распространяемого пакета BLAS. При выполнении операций A\B и B/A используются оптимизированные версии свободно распространяемого пакета LAPACK, если A и B – плотно заполненные матрицы. Если в операции участвуют разреженные матрицы, то используются пакеты BLAS, LAPAC, UMFPACK, CMOLMOD, MAS7, AMD, COLAMD. В частности, использование оптимизированных версий BLAS/LAPACK обеспечивает распараллеливание на основе многопоточности.

2.6. Логические операции

По правилам MATLAB, значение 0 соответствует булевому значению FALSE (ложь), а любое ненулевое значение – булевому значению TRUE (истина).

Логические операции возвращают 1 в качестве TRUE и 0 в качестве FALSE и выполняются над массивами поэлементно.

```
A<B      A>B      A<=B     A>=B     A==B     A~=B
A&B      A|B  ~   A
a&&ba||b
```

Логические операции имееют низкий приоритет по сравнению с операциями отношения и арифметическими операциями.

Приоритет выполнения операций:

1. ()
2. .’ .^ ‘ ^
3. унарные операции + - ~
4. .* ./ .\ * / \
5. бинарные операции + -
6. :
7. <<= >>= == ~=
8. &
9. |
10. &&
11. ||

3. Функции и управляющие конструкции MATLAB

3.1. Управляющие конструкции языка MATLAB

Условный оператор IF имеет вид:

```
if <условие>
    <операторы 1>
. . . . . . . . . . . . . . . . . . . . .
else                  % эта часть
    <операторы 2>         % может отсутствовать
. . . . . . . . . . . . . . . . . . . . .
end
```

Заметим, что в качестве второй группы операторов может использоваться оператор разветвления IF.

```
if <условие 1>
    <операторы 1>
else if <условие 2>
        <операторы 2>
     else
        <операторы 3>
end
```

Оператор множественного ветвления имеет вид

```
switch <выражение> % скалярного
                   % или строкового типа
    case <значение 1>
        <операторы 1>
        % выполняется, если выражение принимает
        % значение 1
    case <значение 2>
        <операторы 2>
        % выполняется, если выражение принимает
        % значение 2
. . . . . .
    otherwise            % эта часть
        <операторы>      % может отсутствовать
        % выполняется, если выражение не совпало
        % ни с одним из значений
. . . . . .
end
```

В отличие от аналогичного оператора C/C++, оператор switch не «проваливается вниз», т.е., если значением выражения является значение 1, будут выполнены операторы 1, а операторы, соответствующие другим ключевым словам case и слову otherwise, не будут выполняться.

Оператор цикла с заранее известным числом повторений имеет вид:

```
for <индекс>=<начальное значение>:<приращение>:
                <конечное значение>
    <операторы>
end
```

По умолчанию индекс изменяется с шагом 1. Для положительных индексов цикл завершается, когда значение индекса превышает конечное значение; для отрицательных значений выполнение прекращается, когда индекс становится меньше конечного значения.

Пример. Решить систему линейных уравнений:

$$\sum_{l=1}^{N} a_{kl}x_l = 1 + \frac{1}{k^2}, k = 1,2,\dots,N;$$

$$a_{kl} = \begin{cases} 1, k = l; \\ \dfrac{1}{2k^2 + 3l^2}, k \neq l. \end{cases}$$

Пусть A – матрица коэффициентов системы линейных уравнений, B– вектор правых частей, X – вектор, содержащий искомое решение. Примем N=10. Решение задачи имеет вид:

```
N=10; % размерность системы уравнений
A = zeros(N); % выделение памяти под матрицу A
B = zeros(N, 1); % выделение памяти под
                 % столбцевой вектор правых частей
For k=1:N
    % заполнение матрицы системы
    forl=1:N
        if k==l
            A(k, l) = 1;
        else
            A(k, l) = 1/(2*k^2+3*l^2);
        end
    end
    % заполнение правых частей
    B(k) = 1 + 1/k^2;
end
% нахождение решения и его печать
X = A\B
% печатьпогрешности
disp('погрешность');
disp(norm(A*X-B));
```

Оператор цикла с предысловием имеет вид:

```
  while<условие> % условие представляет собой
                 % логическое выражение
    <операторы>
  end
```

Оператор выполняется, пока условие принимает значение 1 (TRUE).

Оператор **break** завершает выполнение объемлющего цикла for или while. Управление передается оператору, непосредственно следующему за оператором цикла.

Оператор **continue** передает управление следующему шагу цикла for или while, в котором он размещается. При этом пропускаются операторы, следующие за оператором continueв теле объемлющего цикла.

Обработка ошибочных (исключительных) ситуаций:

```
  try
    <операторы 1>
```

```
catch exObj
  <операторы 2>
end
```

В данном фрагменте, операторы 1 в блоке try выполняются подобно остальному обычному коду программы. Если возникает ошибка в группе операторов 1, то выполняется группа операторов 2, а изучение содержимого переменной `exObj`, характеризующейся типом (классом) MException, позволяет установить причину ошибочной ситуации. Если возникает ошибка в группе операторов 2, интерпретатор MATLAB завершает выполнение программы вне зависимости от наличия других блоков try-catch.

Оператор **return**. Выполнение этого оператора непосредственно в программе приводит к немедленному прекращению работы программы. Выполнение этого оператора в некоторой функции приводит к немедленному выходу из функции и возврату управления в вызывающую функцию или программу.

3.2. Пользовательские функции MATLAB

Функции MATLAB представляют собой подпрограммы, обычно реализованные как М-файлы. Схематически, они имеют вид:

```
function [out1, out2, …]=funname(in1, in2, …)
    <операторы>
end  % это ключевое слово может отсутствовать
```

Представим предыдущий пример в виде функции.

```
function X=My_Func(N)
    A=zeros(N);
    B=zeros(N, 1);
    for k=1:N
        for l=1:N
            if k==l
                A(k, l) = 1;
            else
                A(k, l) = 1/(2*k^2+3*l^2);
            end
        end
        B(k)=1+1/k^2;
```

```
    end
    X=A\B;
End
```

Заметим, что каждая функция оперирует со своим рабочим пространством, отличным от базового рабочего пространства MATLAB, и, соответственно, локальные переменные функции находятся в ее рабочем пространстве. Если требуется, чтобы некоторая переменная, которая появилась внутри функции, хранилась в базовом рабочем пространстве MATLAB, ее нужно объявить глобальной, т.е. видимой во всех функциях:

```
GLOBAL<имя переменной 1><имя переменной 2>
% аналог квалификатора extern C/C++
```

С другой стороны, если требуется, чтобы значения некоторой переменной (локальной переменной) сохранялось в памяти и после выхода из функции, и могло быть использовано при повторном входе в функцию, то

```
PERSISTENT <имя переменной 1><имя переменной 2>
% аналог квалификатора static C/C++
```

Необязательные параметры. Интерпретатор MATLAB позволяет использовать при вызове функций необязательные параметры – как входные, так и выходные.

Например, стандартная функция

```
[B, Idx]=Sort(A, dim)
```

выполняет сортировку элементов матрицы A по измерению dim (по столбцам, если dim=1 и по строкам, если dim=2). Отсортированная матрица возвращается в параметре B, а соответствующие индексы элементов строк или столбцов – в параметре Idx.

Возможные варианты вызова:

```
[B, Idx]=Sort(A, 2)   % сортировка по строкам
[B, Idx]=Sort(A)      % второй параметр опущен;
                      % по умолчанию – сортировка
                      % по столбцам
B=sort(A)     %второй выходной параметр опущен;
              % индексы строк не нужны
```

Иногда требуется проигнорировать один или несколько параметров в числе входных и выходных параметров; в этом случае используется сим-

вол «~». Например, если требуется лишь индексы элементов столбцов, но сама матрица не нужна, то

```
[~, Idx]=sort(A);
```

Первичные функции.

Первая по порядку из расположенных в М-файле функций называется первичной функцией. За ней может следовать несколько других функций, называемых субфункциями, которые служат подрограммами для первичной функции. Именно первичную функцию можно вызвать из другого М-файла, в частности, из основной программы.

Рекомендуется, чтобы имя первичной функции совпадало с именем М-файла. Если имя первичной функции не совпадает с именем файла, то для ее вызова требуется использоваться имя файла.

Заметим, что когда управление передается внутрь М-файла (т.е. когда он выполняется), интерпретатор MATLAB ищет для выполнения именно субфункции в первую очередь по сравнению с остальными одноименными функциями.

Вложенные функции.

Интерпретатор MATLAB допускает наличие так называемых вложенных функций (nestedfunctions) различной глубины вложенности. Как обычно, переменные, которые появляются во вложенных функциях, видны лишь в той функции, для которой они являются локальными. Заметим также, что вложенные функции имеют доступ к рабочим пространствам тех функций, в которые они вложены, и, следовательно, к локальным переменным тех функций, в которые они вложены.

```
function F1(x, y)
F2(x, y);
F4(x, y);
    function F2(x, y)
    F3(x);
    F4(x);
        function F3(x)
        F4(x);
        end
    end
    function F4(x)
    F5(x5);
        function F5(x)
        …
        end
```

```
    end
end
```

Вложенные функции могут быть вызваны:

- на уровне непосредственно выше вложенной функции;
- во вложенной функции непосредственно на том же самом уровне;
- во вложенных функциях на любом другом более глубоком уровне вложения.

Приватные функции.

Приватные функции – это функции, для которых М-файлы (или MEX-файлы) располагаются в суюдиректориях (подкаталогах) со специальным именем private. Эти функции называются приватными, поскольку они видны лишь в М-файлах (скриптах и функциях), которые удовлетворяют следующим условиям:

- функция, которая вызывает приватную функцию, определена в М-файле, распложенном в директории непосредственно на 1 уровень выше, чем директория приватной функции;
- скрипт, который вызывает приватную функцию, сам вызывается из функции, которая имеет доступ к приватной функции по правилу, указанному выше.

3.3. Передача произвольного числа входных и выходных параметров

Интерпретатор MATLAB позволяет использовать функции с произвольным, заранее не фиксированным, числом параметров. При этом для передачи входных параметров используется массив ячеек с предопределенным именем varargin, а для передачи выходных параметров используется массив ячеек с предопределенным именем varargout. Например:

```
function [varargout] = funname(varargin)
    <операторы>
end
```

В пределах текущей функции встроенные функции MATLAB `nargin` и `nargout`, вызванные без аргументов, возвращают число входных и выходных параметров текущей функции. После того, как число входных и выходных параметров становится известно, извлечение k-го параметра выполняется при помощи конструкции `varargin{k}`, а упаковка k-го выходного параметра при помощи конструкции `varargout{k}=…` . Заметим также, что функции `nargin(fun)` и `nargout(fun)`, где fun– имя функции, позволяют определить число входных и выходных параметров требуемой функции.

3.4. Дескрипторы функций

В некотром смысле дескрипторы функций подобны указателям на функции C/C++. Они представляют собой данные специального типа, которые позволяют вызвать функцию в любом месте программы, независимо от области ее видимости. В основном, дескрипторы функций используются для передачи функций как параметров другим функциям.

Дескриптор функции конструируется посредством символа@, за которым следует имя функции:

```
h=@functionname
```

Заметим, что при создании дескриптора функции MATLAB используют те же самые правила для поиска M-файлов (или MEX-файлов), как и при запуске функций на выполнение.

Для запуска функции на выполнение достаточно снабдить дескриптор функции списком аргументов в круглых скобках:

```
h(arg1,arg2,…,argn)
```

Пример:

Найти $\int_1^5 e^{-x^2} dx$

```
function Untitled
quad(@My_Func, 1, 5)
end
function y=My_Func(x)
y=exp(-x.*x)
end
ans=0.1394
```

3.5. Внешние интерфейсы (External Interfaces)

Внешние интерфейсы помогают подключить к матричным системам (MATLAB, Octave, SciLab) бинарные модули с соответствующими подпрограммами (функциями в терминологии C/C++) в формате разделяемых (динамически загружаемых) библиотек, разработанных при помощи языков C/C++/Fortran.

Заметим, что компиляторы C/C++/Fortran как правило, используют один и тот же компоновщик (редактор связей, линкер). Например, в ОС Windows комплекты компиляторов Microsoft C/C++, IntelC/C++/Fortran используют компоновщик (линкер) Microsoftlink.exe, а в ОС Linuxкомплекты компиляторов GNU/IntelC/C++/Fortran используют компоновщик (линкер) GNUld. По указанной причине, достаточно рассмотреть внешние интерфейсы с программами, разработанными на C/C++.

Информация ниже является специфичной для системы MATLAB.

Внешние интерфейсы на уровне функций MATLAB реализуются при помощи MEX-файлов (MATLABExecutable).MEX-файл представляет собой разделяемую (динамически загружаемую) библиотеку с некоторым именем (например, My_Func) и определенным расширением (например, mexw32 в ОС WindowsIA32 и mexw64 в ОС Windowsx64). При вызове функции My_Func интерпретатор MATLAB загружает разделяемую библиотеку My_Func (My_Func.mexw32 или My_Func.mexw64), находит функцию (точка входа) со стандартным именем mexFunctionи выполняет ее. Прототип данной функции, являющейся переходником между интерпретатором MATLAB и остальным кодом C/C++, определен в заголовочном файле mex.h и имеет примерный вид:

```
#ifdef __cplusplus
    extern "C" {
#endif
void mexFunction (intnlhs, // числовыходных
                        // параметров
                  mxArray *plhs[], // массив
                      // указателей на выходные
                      // параметры
                  int nrhs, // число входных
                            // параметров
                  const mxArray *prhs[]
                      // массив указателей на
                      // входные параметры
                   );
#ifdef __cplusplus
    }
#endif
```

Заметим, что, если в одной и той же папке (каталоге, директории) расположены одноименные M-файл и MEX-файл, то интерпретатор MATLAB исполняет MEX-файл, отдавая ему предпочтение перед M-файлом.

Перепишем приведенный ранее пример M-функции в форме исходного кода C++ для MEX-файла (файл My_Func_.cpp). Скрипт компиляции

и сборки при помощи MSVisualC/C++ имеет примерный вид (файл My_Build.cmd):

```
REM Сборка MEX-файлов при помощи MS Visual C/C++
REM использование:
REM Открыть командное окно среды нужной версии компилятора MS Visual
C/C++
REM Visual Studio Command Prompt либо
REM Visual Studio x64 Win64 Command Prompt либо
REM Visual Studio x64 Cross Tools Command Prompt и т.д.
REM Перейти в данную папку (каталог) и выполнить
REM Build_Mex_VC
REM Исходники
set SRC=My_Func_.cpp
REM Имя MEX-файла
set MEXNAME=My_Func_VC
REM Дополнительные статические библиотеки и объектные файлы
set ADDIT_LIB=libmwlapack.lib
REM Имя платформы: win32 для Windows IA32, win64 для Windows x64
set PLATF=win64
REM Расширение mex-файла: mexw32 для Windows IA32, mexw64 для Win-
dows x64
set MEXEXT=mexw64
REM Путь к папке (каталогу), в котором установлен MATLAB
set MATLAB_DIR="C:\Program Files\MATLAB\R2011b"
REM Здесь располагаются заголовочные файлы MATLAB'a
set MATLAB_INCL=%MATLAB_DIR%\extern\include
REM Параметры командной строки для запуска компилятора MS Visual
C/C++ (cl.exe)
set CFLAGS=/I%MATLAB_INCL% /c /Zp8 /GR /W3 /EHs /nologo /MD
/D_CRT_SECURE_NO_DEPRECATE
set CFLAGS=%CFLAGS%/D_SCL_SECURE_NO_DEPRECATE /D_SECURE_SCL=0
/DMATLAB_MEX_FILE
set OPTIMFLAGS=/O2 /DNDEBUG
REM Параметры командной строки для запуска компоновщика Microsoft
(link.exe)
set LINKFLAGS=/dll /export:mexFunction
/LIBPATH:%MATLAB_DIR%\extern\lib\%PLATF%\microsoft
REM Компиляция
cl %CFLAGS% %OPTIMFLAGS% %SRC%
REM Компоновка (линкование)
link %LINKFLAGS% /OUT:%MEXNAME%.%MEXEXT% *.obj libmx.lib libmex.lib
libmat.lib %ADDIT_LIB%
REM Внедрение в MEX-файл сведвний об используемой версии C Run-Time
Library (CRT)
mt -outputresource:%1.%MEXEXT%;2 -manifest %1.%MEXEXT%.manifest
REM Удаление лишних файлов
del *.obj
del *.exp
del *.manifest
del *.lib
```

Код файла My_Func_.cpp:

```
#include <mex.h> //Заголовок для подключения к MATLAB
#include <lapack.h> //Дополнительный заголовок для подключения
                    //функций LAPACK
```

```
//Вспомогательные макросы для доступа к элементам матриц
//и векторов в стиле MATLAB
#define AA(k,j) APtr[(N)*((j)-1)+(k)-1]
#define XX(k) XPtr[(k)-1]

//Переходник между интерпретатором (прекомпилятором)
//MATLAB и остальным кодом C/C++
//Прототип определен в заголовочном файле mex.h
void mexFunction (int nlhs, //Число выходных параметров в вызываемой
функции
                  mxArray *plhs[], //Набор выходных параметров
вызываемой функции
                                   //Тип mxArray - это внутрен-
ний формат
                                         //хранения объектов
MATLAB
                                         //Он определен в заго-
ловке mex.h
                  int nrhs, //Число входных параметров в вызываемой
функции
                  const mxArray *prhs[]) //Набор входных пара-
метров вызываемой функции
{//Проверка числа входных параметров
 if (nrhs!=1) mexErrMsgTxt("My_Func_: Число входных аргументов от-
лично от 1");
 //Проверка числа выходных параметров
 if (nlhs!=1) mexErrMsgTxt("My_Func_: Число выходных параметров от-
лично от 1");
 //Извлечение первого входного параметра
 const mxArray *In1=prhs[0];
 //Является ли входной параметр числовым?
 if (!mxIsNumeric(In1))
      mexErrMsgTxt("My_Func_: Входной аргумент не является число-
вым");
 //Извлечение числа из скалярного входного аргумента
 ptrdiff_t N=mxGetScalar(In1);
 //Захват памяти под матрицу системы
 double *APtr=new double[N*N];
 //Заполнение матрицы системы
 for (int k=1; k<=N; k++)
      for (int j=1; j<=N; j++)
          if (k==j) AA(k,k)=1;
          else AA(k,j)=1.0/(2*k*k+3*j*j);
 //Создание столбцевого вектора MATLAB,
 //т.е. матрицы размером Nx1 из действительных чисел
 //В нем сначала будут помещены правые части лин. ур-ий,
 //а после вызова функции DGESV из LAPACK - искомое решение
 mxArray *R=mxCreateDoubleMatrix(N, 1, mxREAL);
 //Запись указателя на созданный столбцевой вектор в первый элемент
 //набора выходных параметров
 plhs[0]=R;
 //Указатель на собственно массив действительных частей
 //матрицы MATLAB
 double *XPtr=mxGetPr(R);
 //Заполнение правых частей
 for (int k=1; k<=N; k++)
      XX(k)=1+1.0/(k*k);
```

```
 //Число столбцов в наборе правых частей
 ptrdiff_t N_R=1;
 //Массив из целых чисел, в котором сохраняются индексы перестановок
 //при факторизации матрицы системы
 ptrdiff_t *I_PIV=new ptrdiff_t[N];
 //Информационная переменная
 ptrdiff_t INFO;
 //Собственно решение системы линейных уравнений с квадратной
 //матрицей общего вида штатными средствами LAPACK
 dgesv(
    &N, //Число уравнений
    &N_R, //Число правых частей
    APtr, //Адрес начала матрицы системы. Она не сохраняется
    &N, //Число строк в матрице системы
    I_PIV, //Адрес массива индексов перестановок
    XPtr, //Адрес начала вектора правых частей.
           //На выходе он будет содержать искомое решение
    &N,   //Число строк в массиве правых частей
    &INFO //Информация об успешности решения сист. лин ур-ий
);
//Очистка памяти
 delete [] I_PIV;
 delete [] APtr;
 //Анализ успешности решения сист. лин ур-ий
 if (INFO > 0) mexErrMsgTxt("My_Func_: Матрица системы вырождена");
 if (INFO < 0) {mexPrintf("Ошибочное значение %i-го параметра ф-ии
dgesv", -INFO);
                mexErrMsgTxt("My_Func_: Неправильный параметр при
вызове ф-ии dgesv");
     }
}
```

Для компиляции и сборки достаточно выполнить комманду:

```
Build_MexMy_Func_
```

Также, для компиляции и сборки MEX-файлов можно использовать скрипт mex, вызываемый из коммандного окна MATLAB. Например, сначала его можно сконфигурировать коммандой

```
mex –setup
```

а затем выполнить компиляцию и сборку приведенного выше MEX-файла командой:

```
mex My_Func_.cpp libmwlapack.lib
```

4. Объектно-ориентированное программирование в MATLAB

Поддержка объектно-ориентированного программирования заметно усилена в MATLABначиная с версии R2008a. Заметим, что интерпретатор поддерживает старый синтаксис описания классов, их полей (свойств в терминологии MATLAB), методов, переопределения операций и т.д.

Уместно рассмотреть в качестве примера класс из документации MATLAB, инкапсулирующий манипуляции с полиномами. В частности

```
y = polyval(p, x)
```

вычисляет значение полинома $y = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$, коэффициенты которого задаются строчным вектором p, причем хможет быть вектором или матрицей – в таком случае значения полинома вычисляются для всех компонент вектора или матрицы;

```
w=conv(u, v)
```

находит коэффициенты полинома, являющегося произведением полиномов, заданных векотрами коэффициентов uи v;

```
q=polyder(p);
```

```
q=polyder(p1,p2)
```

возвращает строчный вектор коэффициентов полиномов $Q(x) = aP(x)/dx$ или $Q(x) = \frac{d}{dx}(P_1(x)P_2(x))$;

```
r=roots(p)
```

возвращает столбцевой вектор корней полинома, коэффициенты которого заданы строчным вектором p;

```
p=poly(r)
```

возвращает коэффициенты полинома p по вектору его корней r;

```
p=poly(A)
```

возвращает коэффициенты характеристического полинома $\det(\lambda E - A)$ квадратной матрицы A.

В документации к пакету MATLAB приведен пример класса, который инкапсулирует подобные операции

```
classdef DocPolynom
   % Documentation example
   % A value class that implements a data type for polynonials
   % See Implementing a Class for Polynomials in the
   % MATLAB documentation for more information.

   properties
      coef
   end

   % Class methods
   methods
```

```
function obj = DocPolynom(c)
 % Construct a DocPolynom object using the coefficients supplied
   if isa(c,'DocPolynom')
      obj.coef = c.coef;
   else
      obj.coef = c(:).';
   end
end % DocPolynom
function obj = set.coef(obj,val)
   if ~isa(val,'double')
      error('Coefficients must be of class double')
   end
   ind = find(val(:).'~=0);
   if ~isempty(ind);
      obj.coef = val(ind(1):end);
   else
      obj.coef = val;
   end
end % set.coef

function c = double(obj)
   c = obj.coef;
end % double

function str = char(obj)
   % Created a formated display of the polynom
   % as powers of x
   if all(obj.coef == 0)
      s = '0';
   else
      d = length(obj.coef)-1;
      s = cell(1,d);
      ind = 1;
      for a = obj.coef;
         if a ~= 0;
            if ind ~= 1
               if a > 0
                  s(ind) = {' + '};
                  ind = ind + 1;
               else
                  s(ind) = {' - '};
                  a = -a; %#ok<FXSET>
                  ind = ind + 1;
               end
            end
            if a ~= 1 || d == 0
               if a == -1
                  s(ind) = {'-'};
                  ind = ind + 1;
               else
                  s(ind) = {num2str(a)};
                  ind = ind + 1;
                  if d > 0
                     s(ind) = {'*'};
                     ind = ind + 1;
                  end
               end
            end
            if d >= 2
               s(ind) = {['x^' int2str(d)]};
```

```
                ind = ind + 1;
            elseif d == 1
                s(ind) = {'x'};
                ind = ind + 1;
            end
         end
         d = d - 1;
      end
   end
   str = [s{:}];
end % char

function disp(obj)
   % DISP Display object in MATLAB syntax
   c = char(obj);
   if iscell(c)
      disp(['     ' c{:}])
   else
      disp(c)
   end
end % disp

function b = subsref(a,s)
   % SUBSREF Implementing the following syntax:
   % obj([1 ...])
   % obj.coef
   % obj.plot
   % out = obj.method(args)
   % out = obj.method
   switch s(1).type
      case '()'
         ind = s.subs{:};
         b = a.polyval(ind);
      case '.'
         switch s(1).subs
            case 'coef'
               b = a.coef;
            case 'plot'
               a.plot;
            otherwise
               if length(s)>1
                  b = a.(s(1).subs)(s(2).subs{:});
               else
                  b = a.(s.subs);
               end
         end
      otherwise
         error('Specify value for x as obj(x)')
   end
end % subsref

function r = plus(obj1,obj2)
   % PLUS  Implement obj1 + obj2 for DocPolynom
   obj1 = DocPolynom(obj1);
   obj2 = DocPolynom(obj2);
   k = length(obj2.coef) - length(obj1.coef);
   r=DocPolynom([zeros(1,k) obj1.coef]+[zeros(1,-k) obj2.coef]);
end % plus

function r = minus(obj1,obj2)
```

```
         % MINUS Implement obj1 - obj2 for DocPolynoms.
         obj1 = DocPolynom(obj1);
         obj2 = DocPolynom(obj2);
         k = length(obj2.coef) - length(obj1.coef);
         r = DocPolynom([zeros(1,k) obj1.coef] - [zeros(1,-k)
obj2.coef]);
      end % minus

      function r = mtimes(obj1,obj2)
         % MTIMES   Implement obj1 * obj2 for DocPolynoms.
         obj1 = DocPolynom(obj1);
         obj2 = DocPolynom(obj2);
         r = DocPolynom(conv(obj1.coef,obj2.coef));
      end % mtimes

      function r = roots(obj)
         % ROOTS.  ROOTS(obj) is a vector containing the roots of obj.
         r = roots(obj.coef);
      end % roots

      function y = polyval(obj,x)
         % POLYVAL  POLYVAL(obj,x) evaluates obj at the points x.
         y = polyval(obj.coef,x);
      end % polyval

      function q = diff(obj)
         % DIFF  DIFF(obj) is the derivative of the polynom obj.
         c = obj.coef;
         d = length(c) - 1;  % degree
         q = DocPolynom(obj.coef(1:d).*(d:-1:1));
      end % diff

      function plot(obj)
         % PLOT  PLOT(obj) plots the polynom obj
         r = max(abs(roots(obj)));
         x = (-1.1:0.01:1.1)*r;
         y = polyval(obj,x);
         plot(x,y);
         c = char(obj);
         title(['y = ' c{:}])
         xlabel('X')
         ylabel('Y','Rotation',0)
         grid on
      end % plot
   end % methods
end % classdef
```

Описание данных, хранимых в экземплярах класса (в терминологии интерпретатора MATLABони назыаются свойствами – properties), и методов, т.е. функций, предназначенных для обработки хранимых в экземплярах класса данных, может располагаться в одном или нескольких M-файлах.

Например, определение класса ClassNameA и всех его методов может находиться в одном и том же M-файле – в таком случае соответствующий файл должен иметь имя ClassNameA.m и может быть расположен где угодно.

Однако возможна ситуация, когда определение класса ClassNameAрасполагается в файле ClassNameA.m, а его методы – в файлах ClassAMethod1.m, ClassAMethod2.mи т.д. В таком случае, все данные файлы должны располагаться в папке (подкаталоге) с именем @ClassNameA(т.е. имени класса предшествует @).

Схематически, объявление класса имеет вид:

```
classdef (Attribute1=Value1,…) ClassNameA<BaseClass
    properties (Attribute1=Value1,…)
        PropertyName1
        PropertyName2=pi/4;
    end
    methods (Attribute1=Value1,…)
        functionobj=ClassNameA(arg1,arg2,…)
            obj.PropertyName1=arg1;
            …
        end
        function y=ClassMethod1(obj,x)
            y=obj.PropertyName1+x;
        end
        y=ClassAMethod2(obj,x)
        …
    end
…
end
```

Здесь подразумевается, что реализация метода ClassAMethod2 располагается в отдельном M-файле, в подкаталоге @ClassNameA. Если реализация метода располагается в отдельном M-файле (например, ClassAMethod2.m), и требуется указать специфичные атрибуты для метода, то в описании класса приводится лишь сигнатура метода. Если указание специфичных атрибутов не требуется, то сигнатура метода в описании класса может вообще отсутствовать. В самом M-файле метод реализуется как обычная функция, без блока methods-end.

```
% ClassAMethod2.m
function y = ClassAMethod2(obj,x)
    y=obj.PropertyName2-x;
end
```

Заметим также, что если объект obj представляет собой объект класса ClassNameA, то, с точки зрения конструктора MATLAB, следующие вызовы эквивалентны:

```
obj.ClassAMethod1(x)
```

и

```
ClassAMethod1(obj,x)
```

Предположим также, что класс

```
Classdef ClassNameB < ClassNameA
…
end
```

производный от класса ClassNameA. В методах производного класса имеется возможность вызова методов базового класса при помощи конструкции:

```
Имя_Метода@Имя_Базового_Класса
```

Например:

```
obj.ClassMetod1@ClassNameA(x);
```

Конструктор – это специальный метод, который создает экземпляр класса. Как правило, конструктор воспринимает входные данные, чтобы выполнить их присваивание хранимым в классе данным (properties) и всегда возвращает инициализированный объект. При этом:

- конструктор должен иметь имя, совпадающее с именем класса;
- конструктор может иметь только один выходной параметр – собственно конструируемый объект;
- при создании производного класса MATLAB вызывается конструктор для каждого базового типа. Неявные вызовы конструкторов базового типа выполняются в случае, если конструктору базового класса не требуется передавать каких-либо аргументов. Если конструктор базового класса требует передачи ему аргументов, он должен быть явным образом вызван из конструктора производного класса;
- класс не нуждается в определении конструктора, если только он не является производным классом базового класса, в котором конструктор требует входные аргументы. В таком случае, потребуется явный вызов конструктора базового класса с передачей ему соответствующих аргументов;
- если класс не определяет конструктор, то генерируется конструктор по умолчанию, который возвращает скалярный объект, у которого свойства инициализированны пустыми значениями или значениями по умолчанию, указанными в секции properties-end;
- если класс загружается в рабочее пространство MATLAB из дискового файла или создается массив MATLAB из объектов класса, то

необходим конструктор без входных аргументов (т.е. конструктор по умолчанию);

- конструктор всегда должен возвращать объект своего класса;
- вызов конструктора базового класса не зависеть от какого-либо условия – т.е. он не может находиться внутри условного оператора или оператора цикла;
- доступ к конструктору, как и к любому методу, можно ограничить при помощи соответствующих атрибутов.

Методы доступа к свойствам класса позволяют выполнить некоторый код всякий раз, когда требуется получить информацию о свойствах класса или присвоить свойствам класса новые значения. Заметим, что внутри методов доступа нельзя вызвать другую функцию, которая пытается получить или изменить значения свойства. Методы доступа к свойствам класса имеют вид:

```
……
methods
      function Value=get.PropertyName(obj)
                  % для возврата значения, которое
               Value=……; %не хранится в классе
      end
      function obj=set.PropertyName(obj,Value)
               ……
               obj.PropertyName=Value;
               ……
      end
……
end
……
```

Статические методы (атрибут **Static**) ассоциируются с классом как с таковым, но не с конкретным экземпляром класса. Следовательно, статические методы не требуют в качестве первого аргумента экземпляра своего класса. Например, они могут быть полезны, если потребуется выполнить некоторый специфичный код без создания экземпляра класса.

MATLAB поддерживает два типа классов: классы типа «значение» и классы типа «дескриптор». Классы типа «значение», к которым относится большинство классов MATLAB, представляют собой объекты, от которых не требуется быть уникальными. Например – числовые значения. Классы типа «дескриптор» используется в том случае, когда требуется создать ссылку на данные в объекте какого-либо класса, и нежелательно, чтобы копирование объекта приводило к копированию хранимых в нем данных. Все классы типа «дескриптор» создаются, как производные встроенного класса **handle**.

Для класса типа «дескриптор» возможно создать деструктор, который вызывается при завершении жизненного цикла каждого экземпляра класса «дескриптор», например, для очистки некоторой информации или освобождения некоторых ресурсов. При этом поочередно будет вызываться деструктор **delete** для каждого из базовых классов.

```
.....
methods
    .....
    function delete(obj)
    .....
    end
.....
end
```

Заметим, что интерпретатор MATLAB уничтожает объект в рабочем пространстве текущей функции, если:

a) соответствующей переменной присваивается новое значение;
b) переменная нигде далее не используется в оставшейся части функции;
c) завершается выполнение функции.

Некоторые методы класса требуют, чтобы их реализация находилась в пределах блока **classdef-end**, в частности:

a) конструктор класса;
b) деструктор **delete**;
c) любая функция, которая содержит символ «.» в своем имени. Например, это методы доступа к свойствам класса.

5. Графические возможности MATLAB

Перечислим вкратце стандартные функции MATLAB, которые служат для отображения графиков, диаграмм и другого графического отображения данных.

Функция

```
plot(X1,Y1,X2,Y2,…,Xn,Yn);
plot(X1,Y1,LineSpec1,X2,Y2,LineSpec2,…);
```

отвечает за построение графиков n линий в одних и тех же координатных осях по точкам X1, Y1, X2, Y2, …, Xn, Yn, если все параметры представляют собой строчные или столбцевые векторы соответствующих размерностей.

Если параметр X является столбцевым или строчным вектором размерности N1, а параметр Y является матрицей размерности N1xN2, то вместо одной линии параметрам X и Y соответствует N2 линий, абсциссы точек которых хранятся в векторе X, а ординаты в матрице Y.

Если и параметр X, и параметр Y являются матрицами размерности N1xN2, то вместо одной линии параметрам X и Y соответствует N2 линий по N1 точек каждая.

Если матрица Y является комплексной, то вызов

```
plot(Y)
```

эквивалентен вызову

```
plot(real(Y),img(Y))
```

Функция

```
plot3(X1,Y1,Z1,X2,Y2,Z2,…,Xn,Yn,Zn);
plot3(X1,Y1,Z1,LineSpec1,X2,Y2,Z2,LineSpec2,…);
```

выполняет построение n кривых в трехмерном пространстве по аналогичным правилам.

Функция

```
loglog(X1,Y1,X2,Y2,…,Xn,Yn);
loglog(X1,Y1,LineSpec1,X2,Y2,LineSpec2,…);
```

выполняет построение графиков в логарифмическом масштабе по обеим осям.

Функция

```
semilogx(X1,Y1,X2,Y2,…,Xn,Yn);
semilogx(X1,Y1,LineSpec1,X2,Y2,LineSpec2,…);
```

построение графиков в полулогарифмическом масштабе. Для оси x используется логарифмический маштаб, для оси y – линейный.

Функция

```
semilogy(X1,Y1,X2,Y2,…,Xn,Yn);
semilogy(X1,Y1,LineSpec1,X2,Y2,LineSpec2,…);
```

построение графиков в полулогарифмическом масштабе. Для оси y используется логарифмический маштаб, для оси x – линейный.

Функция

```
plotyy(X1,Y1,X2,Y2);
```

выполняет построение пары графиков с двумя шкалами ординат.

Функция

```
bar(Y); bar3(Y);
```

обеспечивает построение двуммерных и трехмерных гистограмм соответственно.

Функция

```
pie(X); pie3(X);
```

отвечает за построение круговых диаграмм, двумерных и трехмерных соответственно.

Функции

```
mesh(X,Y,Z,…);
surf(X,Y,Z,…);
```

обеспечивают построение двумерных поверхностей в трехмерном пространстве. Матрицы X, Y, Z имееют одинаковый размер и содержат координаты поверхностей.

6. Поддержка в MATLAB разреженных матриц

В вычислительной математике, при решении задач математической физики, часто возникают матрицы чрезвычайно большого размера, у которых лишь незначительная часть элементов являются ненулевыми. Разреженный формат хранения матриц предусматривает хранение лишь ненулевых элементов подобных матриц, что позволяет легко их обрабатывать уже на персональных компьютерах и ноутбуках.

6.1. Внутренний формат хранения разреженных матриц

В основном, используется координатный, сжатый по строкам или по столбцам формат хранения разреженных матриц. Пакет MATLAB использует сжатый по столбцам (column compressed) формат.

Пусть разреженная матрица A имеет размерность $m \times n$, содержит $N \ll mn$ ненулевых элементов и представляется в виде набора: $A = \{\mathbf{A}^{(r)}, \mathbf{A}^{(i)}, \mathbf{I}, \mathbf{J}\}$, где:

- $\mathbf{A}^{(r)} = (A_0^{(r)}, A_1^{(r)}, ..., A_{N-1}^{(r)})^T$ содержит действительные части ненулевых элементов разреженной матрицы A, упорядоченных по столбцам;
- $\mathbf{A}^{(i)} = (A_0^{(i)}, A_1^{(i)}, ..., A_{N-1}^{(i)})^T$ содержит мнимые части ненулевых элементов разреженной матрицы A (отсутствует для матриц с действительными элементами), упорядоченных по столбцам;

- $\mathbf{I} = (I_0, I_1, ..., I_{N-1})^T$ содержит номера строк ненулевых элементов разреженной матрицы A, упорядоченных по столбцам;
- $\mathbf{J} = (0, J_1, ..., J_{N-1}, N)^T$ кодирует общее число ненулевых элементов во всех столбцах, предшествующих j-му столбцу.

Например, разреженная матрица

$$A = \begin{bmatrix} 2.5 & 3.7 & 0 & 0 & 0 \\ 3+2i & 0 & 4.3 & 0 & 6+7i \\ 0 & -1.2 & -3 & 2+3i & 0 \\ 0 & 0 & 1+0.5i & 0 & 0 \\ 0 & 4-i & 2.3 & 0 & 1.3 \end{bmatrix}$$

представляется в виде

$$\mathbf{A}^{(r)} = (2.5, 3, 3.7, -1.2, 4, 4.3, -3, 1, 2.3, 2, 6, 1.3)^T$$

$$\mathbf{A}^{(i)} = (0, 2, 0, 0, -1, 0, 0, 0.5, 0, 3, 7, 0)^T$$

$$\mathbf{I} = (0, 1, 0, 2, 4, 1, 2, 3, 4, 2, 1, 4)^T$$

$$\mathbf{J} = (0, 2, 5, 9, 10, 12)^T$$

Заметим, что разреженные матрицы изоморфны ориентированным взвешенным графам, дугам которых приписаны действительные либо комплексные числа.

При манипуляциях с разреженными матрицами фактически под вспомогательные массивы может выделяться несколько больше памяти, чем требуется для хранения N и n+1 элементов, что позволяет избежать перераспределения памяти при вставке очередного элемента. Тем не менее, вставка элементов в первые столбцы может приводить к достаточно большому числу операций копирования содержимого оперативной памяти.

Не все функции MATLAB поддерживают работу с разреженными матрицами. Для того, чтобы получить полный список функций, поддерживающих работу с разреженными матрицами в текущей версии MATLAB, достаточно выполнить команду **help sparfun**.

6.2. Создание разреженных матриц

Функция

```
S=sparse(m,n)
```

создает пустую (не содержащую элементов) разреженную матрицу размерности $m \times n$

```
S=sparse(Ir,Jc,Ari,m,n,nzmax)
```

создает разреженную матрицу размерности $m \times n$, в которой зарезервировано место для хранения nzmax ненулевых элементов. Здесь Ir,Jc,Ari – векторы одинаковой длины, содержащие номера строк (начинаются с 1), номера столбцов (начинаются с 1) и значения ненулевых элементов.

```
S=sparse(Ir,Jc,Ari,m,n)
```

то же самое, но nzmax определяется длиной векторов Ir,Jc,Ari.

```
S=sparse(Ir,Jc,Ari)
```

то же самое, но величины m и n определяются максимальными значениями векторов Ir и Jc.

```
S=sparse(A)
```

преобразует плотно заполненную матрицу A в разреженную, удаляя из нее нулевые элементы.

```
A=full(S)
```

выполняет преобразование разреженной матрицы в плотно заполненную

```
S=spalloc(m,n,nzmax)
```

выполняет создание разреженной матрицы размерности $m \times n$, в которой зарезервирована область памяти для хранения nzmax ненулевых элементов

```
S=speye(m,n)
```

выполняет создание разреженной матрицы размерности $m \times n$ с единицей на главной диагонали

```
S=spdiags(B,d,m,n)
```

создает ленточную разреженную матрицу размерности $m \times n$. При этом столбцы вектора **B** заполняют диагонали, номера которых указаны в векторе **d**

6.3. Функции для работы с разреженными матрицами

Функция

```
issparse(S)
```

возвращает true(1), если матрица является разреженной, и false(0) в противном случае.

```
nnz(S)
```

возвращает число ненулевых элементов в том числе и для разреженных матриц.

```
a=nonzeros(S)
```

возвращает плотно заполненный столбцевой вектор из ненулевых элементов матрицы S в том числе и для разреженных матриц.

```
F=spfun(fun,S)
```

выборочно применяет функцию fun (первый параметр – дескриптор функции) к ненулевым элементам матрицы S.

```
spy(S)
```

графически отображает заполненность разреженной матрицы S ненулевыми элементами.

6.4. Операции над разреженными матрицами

При выполнении операций над разреженными матрицами важную роль играют алгоритмы переупорядочивания их элементов.

Функция

```
P = colamd(S)
```

генерирует такой строчный вектор перестановки столбцов p, что LU - разложение матрицы S(, p) будет содержать не больше (а фактически меньше) ненулевых элементов, чем LU – разложение исходной квадратной разреженной матрицы S достаточно большого размера.

Функция

```
P = symamd(S)
```

генерирует такой строчный вектор p симметричных перестановок строк и столбцов исходной симметричной либо Эрмитово положительно определенной матрицы S, что разложение Холецкого матрицы S(p, p) после перестановок будет содержать не больше (а фактически меньше) ненулевых элементов, чем разложение Холецкого исходной матрицы S.

7. Факторизация матриц, решение систем линейных уравнений и задач «метода наименьших квадратов»

7.1. Методы факторизации матриц

Решение систем линейных уравнений требует разложения матриц на множители, называемого также факторизацией матриц.

8.1.1. Если $A = \{a_{kj}\}$, $k, j = 1, 2, ..., n$ – квадратная матрица общего вида, то (без учета возможных перестановок строк и столбцов при выборе главного элемента) численное решение по методу Гаусса системы линейных алгебраических уравнений с матрицей A приводит к LU-разложению

$$A = LU,$$

где L – нижняя треугольная матрица с единицей на главной диагонали, U – верхняя треугольная матрица.

Матрица

$$\begin{bmatrix} 1 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 1 & 0 & 0 & 0 & 0 \\ 0 & \ldots & s_{k+1,k} & 1 & 0 & 0 & 0 \\ 0 & \ldots & s_{k+2,k} & 0 & 1 & \ldots & 0 \\ 0 & \ldots & s_{n,k} & 0 & 0 & 0 & 1 \end{bmatrix} = E + S_k$$

где матрица S_k содержит ненулевые элементы лишь в k-м столбце ниже главной диагонали, называется матрица Гаусса. Нетрудно проверить, что для матриц Гаусса справедливо

$(E + S_k)(E + S_j) = (E + S_k + S_j)$ и, следовательно, $(E + S_k)^{-1} = E - S_k$

Решение системы линейных алгебраических уравнений с матрицей A по методу Гаусса сводится к последовательным преобразованиям с умножением слева на следующие матрицы Гаусса:

$$\begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ -a_{21} / a_{11} & 1 & 0 & \ldots & 0 \\ -a_{31} / a_{11} & 0 & 1 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ -a_{n1} / a_{11} & 0 & 0 & \ldots & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ a_{31} & a_{32} & a_{34} & \ldots & a_{3n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & a_{32} & \ldots & a_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \ldots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \ldots & a_{3n}^{(1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \ldots & a_{nn}^{(1)} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ 0 & -a_{32}^{(1)} / a_{22}^{(1)} & 1 & ... & 0 \\ ..... & ..... & ..... & ..... & ..... \\ 0 & -a_{32}^{(1)} / a_{22}^{(1)} & 0 & ... & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & ... & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & ... & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & ... & a_{3n}^{(1)} \\ ..... & ..... & ..... & ..... & ..... \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & ... & a_{nn}^{(1)} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{11} & a_{12} & a_{13} & ... & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & ... & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & ... & a_{3n}^{(2)} \\ ..... & ..... & ..... & ..... & ..... \\ 0 & 0 & a_{n3}^{(2)} & ... & a_{nn}^{(2)} \end{bmatrix}$$

и т.д. Отсюда

$$L = \begin{bmatrix} 1 & 0 & 0 & ... & 0 \\ a_{21} / a_{11} & 1 & 0 & ... & 0 \\ a_{31} / a_{11} & a_{32}^{(1)} / a_{22}^{(1)} & 1 & ... & 0 \\ ..... & ..... & ..... & ... & 0 \\ a_{n1} / a_{11} & a_{n2}^{(1)} / a_{22}^{(1)} & a_{n3}^{(2)} / a_{33}^{(2)} & ... & 1 \end{bmatrix}, \quad U = \begin{bmatrix} a_{11} & a_{12} & a_{13} & .... \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & ... \\ 0 & 0 & a_{33}^{(2)} & ... \\ .... & ..... & ..... & ..... \\ 0 & 0 & 0 & ... \end{bmatrix}$$

8.1.2. Если $A = \{a_{kj}\}$, $k, j = 1,2,...,n$ – симметричная, либо Эрмитова положительно определенная матрица, то можно полагать, что в LU-разложении $U = R$, $L = R^H$, где $R = \{r_{kj}\}$ – верхняя треугольная матрица, и, следовательно,

$$A = R^H R$$

– разложение Холецкого, аналогичное решению системы линейных алгебраических уравнений по методу квадратного корня. Элементы верхней треугольной матрицы R находятся из системы уравнений

$$a_{ik} = \sum_{j=1}^{\min(i,k)} \overline{r}_{ji} r_{jk}$$

квадратичной по диагональным элементам матрицы R и линейной по элементам матрицы R, лежащим выше главной диагонали.

8.1.3. Если A – симметричная либо Эрмитова неопределенная матрица, то можно полагать, что в LU-разложении $U = DL^H$, где D – диагональная матрица. Следовательно, в данном случае имеет место $LDLH$-разложение

$$A = LDL^H$$

Для улучшения свойств обусловленности при выполнении $LDLH$-

разложения выполняется строк/столбцов так, что D может быть блочно-диагональной симметричной либо Эрмитовой матрицей с блоками размера 1×1 и 2×2.

8.1.4. Если A – прямоугольная матрица размерости $m\times n$, то

при $m \le n$: $A = LQ$, L – нижняя треугольная матрица, $Q^{-1} = Q^H$ – унитарная матрица (*LQ*-разложение)

при $m \ge n$: $A = QR$, R –верхняя треугольная матрица, $Q^{-1} = Q^H$ – унитарная матрица (*QR*-разложение)

Достаточно пояснить, как реализуется *QR*-разложение, поскольку *LQ*-разложение выполняется аналогично. Пусть матрицы Q_k, $k = 1,2,...N$ унитарны (в частном случае, когда их элементы действительны – ортогональны), т. е. $Q_k^{-1} = Q_k^H$. Поскольку произведение унитарных матриц также унитарно, то

$$Q = \prod_{k=1}^{N} Q_k \quad \Rightarrow \quad Q^{-1} = Q^H$$

Следовательно, построение *QR*-разложения сводится к домножению слева исходной матрицы A на некоторые элементарные унитарные матрицы с целью последовательного обнуления поддиагональных элементов.

При $c,s \in \mathbb{C}$, $\arg c = \arg s$, $|c|^2 + |s|^2 = 1$ элементарная матрица вращений

$$G = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

позволяет обнулить нужный поддиагональный элемент в матрице GA.

Пусть $\mathbf{a} = (a_{11}, a_{21}, ..., a_{m1})^T$ – первый столбец матрицы A, $\mathbf{v} = (v_1, v_2, ..., v_m)^T \in \mathbb{C}^m$, $|\mathbf{v}| = 1$. Тогда, если $\mathbf{a}/|\mathbf{a}| = -2v_1\mathbf{v} + (1,0,...,0)^T$, то матрица отражения $H = E - 2\mathbf{v}\mathbf{v}^H$, определяемая вектором $\mathbf{v}$ (и Эрмитова, и унитарная одновременно: $H^{-1} = H^H = H$), обнуляет все поддиагональные элементы первого столбца матрицы $HA = A - 2\mathbf{v}\mathbf{v}^H A$.

Обнуление всех поддиагональных элементов второго и последующих столбцов выполняется аналогично.

Таким образом, выполнение *QR*-разложения сводится к домножению слева исходной матрицы A на некоторые элементарные унитарные матрицы отражений и вращений.

7.2. LU-разложение

Следующие вызовы для факторизации плотно заполненных матриц используют LAPACK/BLAS, а для факторизации разреженных матриц – UMFPACK/BLAS

Функция

```
[L,U,P]=lu(A)
```

возвращает верхнюю треугольную матрицу U, нижнюю треугольную матрицу L с единицей на главной диагонали и матрицу перестановок P такую, что $LU = PA$

```
Y=lu(A)
```

возвращает в матрице Y верхнюю треугольную матрицу U и поддиагональную часть нижней треугольной матрицы L такой, что $LU = PA$

```
[L,U]=lu(A)
```

возвращает верхнюю треугольную матрицу U и результата перестановки строк в нижней треугольной матрице L с единицей на главной диагонали так, что $LU = PA$

Следующие два вызова предназначены для LU-факторизации разреженных матриц и используют UMFPACK/BLAS

```
[L,U,P,Q,R]=lu(A)
```

возвращает нижнюю треугольную матрицу L с единицей на главной диагонали, верхнюю треугольную матрицу U, матрицу перестановки строк P, матрицу перенумерации столбцов Q и диагональную масштабирующую матрицу R, такие, что $PR^{-1}AQ = LU$

```
[L,U,P,Q]=lu(A)
```

то же самое, но $PAQ = LU$

Для того, чтобы сохранить информацию о перестановке строк и столбцов не в квадратных матрицах, а в строчных векторах, используются вызовы

```
[L,U,p]=lu(A,'vector')
[L,U,p,q]=lu(A,'vector')
[L,U,p,q,R]=lu(A,'vector')
```

7.3. Разложение Холецкого

Разложение Холецкого для симметричных, либо Эрмитовых, положительно определенной матрицы с точностью до симметричных перестановок имеет вид:

$R^H R = A$, $R -$ верхняя треугольная;

$LL^H = A$, $L = R^H$ – нижняя треугольная

```
R=chol(A);
L=chol(A,'lower')
```

Если исходная матрица является плотно заполненной, то используется LAPACK/BLAS, а если она является разреженной, то используется CHOLMOD/LAPACK/BLAS.

```
[R,p]=chol(A);
[L,p]=chol(A,'lower')
```

разложение Холецкого максимально возможного положительно определенного левого верхнего минора матрицы A размерности $(p-1)$. Если $p = 0$, то матрица A положительно определена.

Вызовы

```
[R,p,S]=chol(A);
[L,p,S]=chol(A,'lower')
```

дополнительно генерирует матрицу S симметричных перестановок строк и столбцов

Вызовы

```
[R,p,s]=chol(A,'vector');
[L,p,s]=chol(A,'lower', 'vector')
```

матрицы симметричных перестановок кодируются строчным вектором s

7.4. LDLH-разложение

Следующие вызовы для входных симметричных и Эрмитовых неопределенных плотно заполненных матриц используют LAPACK/BLAS, а для симметричных действительных неопределенных разреженных матриц используют MA57/ BLAS.

```
[L,D,P]=ldl(A);
```

```
[L,D,p]=ldl(A,'vector')
```

$P^H AP = LDL^H$, L – нижняя треугольная матрица с единицей на главной диагонали; D – блочно-диагональная симметричная, либо Эрмитова, матрица с блоками размера 1×1 и 2×2; P, **p** – матрица либо строчный вектор перестановок

```
[L,D,P]=ldl(A,'upper')
[L,D,p]=ldl(A,'upper','vector')
```

$P^H AP = U^H DU$, $U = L^H$ – верхняя треугольная матрица

Следующий вызов предназначен для факторизации лишь разреженных действительных симметричных матриц, генерирует дополнительно масштабирующую диагональную матрицу S, такую, что $P^T SASP = U^T DU$ и использует MA57/ BLAS

```
[L,D,P,S]=ldl(A);
[L,D,p,S]=ldl(A,'vector')
```

7.5. QR-факторизация

Следующие вызовы для прямоугольнольных плотно заполненных матриц используют LAPACK/BLAS, а (с MATLAB 2009a) для разреженных матриц используют SuiteSparseQR/CHOLMOD/LAPACK/BLAS

```
[Q,R,P]=qr(A);
[Q,R,p]=qr(A,'vector');
```

$AP = QR$, $Q^{-1} = Q^H$, R – верхняя треугольная, P, **p** – матрица либо вектор, кодирующие перестановку столбцов

```
[Q,R]=qr(A)
```

$A = QR$

```
[Q,R,p]=qr(A,0);
[Q,R]=qr(A,0)
```

– «экономичная» *QR*-факторизация

```
[C,R,P]=qr(A,B);
[C,R,p]=qr(A,B,'vector');
[C,R]=qr(A,B)
```

```
[C,R,p]=qr(A,B,0);
[C,R]=qr(A,B,0)
```

– вместо унитарной матрицы Q возвращается матрица $C = Q^H B$, что далее используется при численном решении недоопределенных либо переопределенных СЛАУ $AX = B$ по методу наименьш. квадратов

7.6. Решение систем линейных алгебраических уравнений

Решение системы линейных уравнений

AX=B, т.е. $X=A^{-1}B$,

где A, B, X – матрицы соответствующей размерности, обеспечивается оператором левого деления

```
X=A\B, т.е.X=mldivide(A,B).
```

Аналогично, решение системы линейных уравнений

XA=B, т.е. $X=BA^{-1}$,

где A, B, X – матрицы соответствующей размерности, обеспечивается оператором правого деления

```
X=B/A, т.е.X=mrdivide(A,B).
```

С решением систем линейных уравнений тесно связаны еще две стандартные функции MATLAB: вычисление определителей и нахождение обратной матрицы.

Функция

```
Z = det(A)
```

возвращает определитель квадратной матрицы.

Однако, если не нормировать должным образом строки или столбцы матрицы достаточно большого размера, вычисление определителя приводит к чрезвычайно большим числовым величинам и может вызвать переполнение порядка.

Функция

```
B = inv(A)
```

возвращает обратную матрицу для квадратной невырожденной матрицы A. Функция бессмысленна для разреженных матриц, т.к. фактически обратная матрица может быть плотно заполненной.

7.7. Алгоритм работы решателей систем линейных алгебраических уравнений

1) Если матрица A является разреженной и диагональной, то решение $X = A^{-1}B$ (или $X = BA^{-1}$) находится делением на диагональные элементы матрицы A
2) Если матрица A является разреженной, квадратной и ленточной:
   a) Если A – действительна и трехдиагональна, B – столбцевой вектор, то используется метод прогонки, при условии, что не требуется выполнять перестановку строк/столбцов.
   b) В противном случае используются оптимизированные «ленточные» решатели LAPACK/BLAS
3) Если A является верхней или нижней треугольной, то выполняется прямое или обратное исключение компонент искомого решения. Для плотно заполненных матриц используется LAPACK/BLAS.
4) Если A есть результат перестановки строк или столбцов треугольной матрицы, то этот случай распознается и сводится к предыдущему.
5) Если, с точностью до одной диагонали, матрица является треугольной (такие матрицы называются матрицами Хессенберга), то последовательным исключением элементов матрица СЛАУ приводится к треугольной форме.
6) Если A симметрична, либо Эрмитова, имеет действительные положительные элементы на главной диагонали и является разреженной, то предпринимается попытка выполнить разложение Холецкого

   $$P^T AP = R^H R,$$

   R – верхняя треугольная матрица, P – матрица перестановок.
   Для плотно заполненных матриц используется LAPACK/BLAS
   Разреженные матрицы: используется CHOLMOD/LAPACK/BLAS
7) Если A плотно заполнена, симметрична либо Эрмитова, либо A разреженная, действительная и симметричная, и при этом разложение Холецкого невозможно, то она факторизуется след. образом

   $P^T AP = LDL^T$, P – матрица перестановок, L – нижняя треугольная матрица, D – блочно-диагональная симметричная либо Эрмитова матрица с блоками размера 1×1 и 2×2
   Для плотно заполненных матриц используется LAPACK/BLAS
   Для разреженных матриц используется MA57/BLAS
8) Если матрица A является квадратной и не удовлетворяет предшествующим критериям, то выполняется *LU*-факторизация вида:
9) Для плотно заполненных матриц при помощи LAPACK/BLAS:

   $A = PLU$, P – матрица перестановок строк, L – нижняя треугольная матрица, U – верхняя треугольная матрица
10) Для разреженных матриц при помощи UMFPACK/BLAS:

   $PD^{-1}AQ = LU$, P – матрица перестановок строк, Q – матрица пе-

рестановок столбцов, D – диагональная матрица, L – нижняя треугольная матрица, U – верхняя треугольная матрица

11) Если матрица не является квадратной, то реализуется т.н. метод наименьших квадратов:

- Если система линейных алгебраических уравнений является недоопределенной, то реализуется «метод наименьших квадратов». С точностью до перестановки, выполняется LQ-факторизация

  $A = LQ$, $Q^{-1} = Q^H$, L – нижняя треугольная матрица,

  и минимизируется величина

  $$\left\|X\right\|_2 = \sqrt{\sum_{k=1}^{N_1}\sum_{j=1}^{N_2} |\, x_{kj}\,|^2} \to \min$$

- Если система линейных алгебраических уравнений является переопределенной, то также реализуется «метод наименьших квадратов». С точностью до перестановки, выполняется QR-факторизация

  $A = QR$, $Q^{-1} = Q^H$, R – верхняя треугольная матрица,

  и минимизируется величина

  $$\left\|AX - B\right\|_2 \to \min$$

- для плотно заполненных матриц используется LAPACK/BLAS
- для разреженных – SuiteSparseQR/CHOLMOD/LAPACK/BLAS

7.8. Пояснение к методу наименьших квадратов

Рассмотрим для простоты систему линейных уравнений

$$A\mathbf{x} = \mathbf{b}$$

с единственной правой частью-столбцом $\mathbf{b}$ и прямоугольной матрицей A имеет размерности $N_1 \times N_2$. Полагаем, что $\mathrm{rank}(A) = \min(N_1, N_2)$.

Если квадратная матрица A является вырожденной, то, в зависимости от того, является система линейных алгебраических уравнений недоопределенной или переопределенной, рассмотрение сводится к одному из представленных ниже случаев.

1) Пусть $N_1 < N_2$, и система недоопределена. Из всех возможных решений недоопределенной системы линейных алгебраических уравнений требуется выбрать такое решение, что $|\,\mathbf{x}\,| \to \min$. Выполняем LQ-разложение

   $A = LQ$, $Q^{-1} = Q^H$, L – нижняя треугольная матрица.

   и переходим к новым искомым переменным

   $$\mathbf{y} = Q\mathbf{x},$$

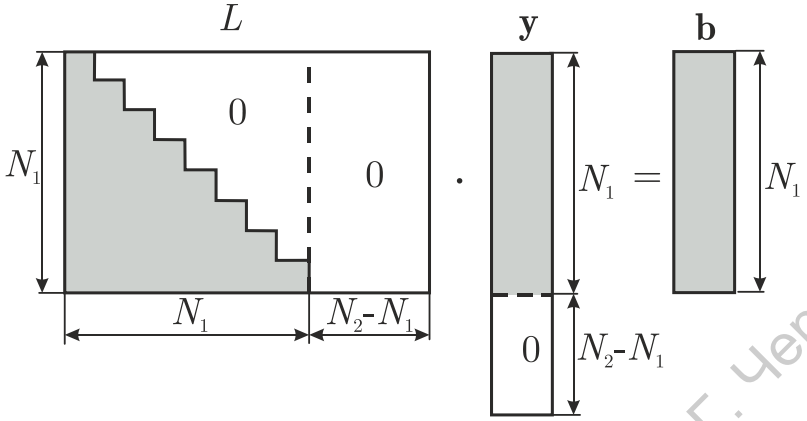
   причем $|\,\mathbf{y}\,| = |\,\mathbf{x}\,|$, и после нахождения $\mathbf{y}$ исходные переменные вычис-

ляются следующим образом:

$$\mathbf{x} = Q^H \mathbf{y}$$

Исходная задача преобразуется к следующей форме

$$L\mathbf{y} = \mathbf{b}, \quad |\,\mathbf{y}\,| \to \min$$



Минимизация величины $|\,\mathbf{y}\,|$ приводит к тому, что $y_{N_1+1} = ... = y_{N_2} = 0$, а $y_1,...,y_{N_1}$ находятся посредством решения системы линейных алгебраических уравнений с нижней треугольной матрицей.

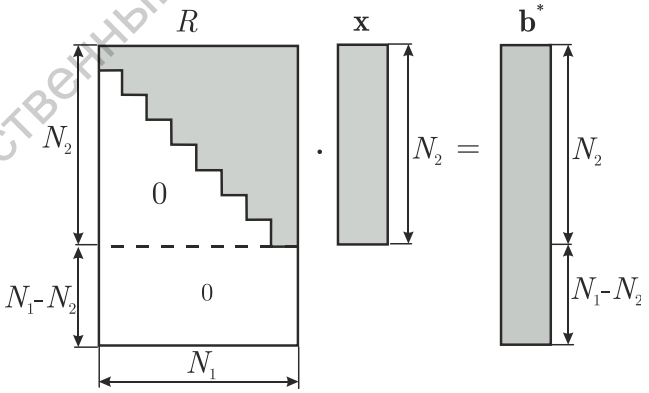
2) $N_1 > N_2$, т.е. система переопределена, не имеет решения, и $\mathbf{x}$ необходимо выбрать так, чтобы $|\,A\mathbf{x} - \mathbf{b}\,| \to \min$

Выполняя QR-факторизацию, находим

$A = QR$, $Q^{-1} = Q^H$, R – верхняя треугольная матрица

Т.к. унитарное преобразование сохраняет длины векторов, то

$$|\,\mathbf{d}\,| \to \min, \quad \text{где } \mathbf{d} = Q^H(A\mathbf{x} - \mathbf{b}) = R\mathbf{x} - \mathbf{b}^*,\ \mathbf{b}^* = Q^H\mathbf{b}$$



Мимнимизация величины $|\,\mathbf{d}\,|$ обеспечивается выполнением требования $d_1 = d_2 = ... = d_{N_2} = 0$, чтоприводит к решению системы линейных алгебраических уравнений относительно x_1, x_2,..., x_{N_2} с верхней треугольной матрицей.

7.9. Дополнительные функции, реализующие метод наименьших квадратов

Пусть $\mathbf{w} = (w_1, w_2, ..., w_m)^T \in \mathbb{R}^m$, $w_k > 0$ – т.н. весовой вектор

$\mathbf{a} = (a_1, a_2, ..., a_m)^T \in \mathbb{C}^m$, $\|\mathbf{a}\|_{\mathbf{w}} = \sum_{k=1}^{m} w_k \mid a_k \mid^2 \Big)^{1/2}$ – норма по отн. к $\mathbf{w}$

$V = \{v_{kj}\} \in \mathbb{C}^{m\times m}$, $k, j = 1, 2, ..., m$ – симметричная, либо Эрмитова матрица; $\|\mathbf{a}\|_V = (\mathbf{a} \cdot V\mathbf{a})^{1/2} = \sum_{k=1}^{m} \sum_{j=1}^{m} v_{kj} a_j \Big)^{1/2}$

Функция

```
x=lscov(A,B,w)
```

находит вектор $\mathbf{x} \in \mathbb{C}^n$, $n \le m$, такой, что $\|A\mathbf{x} - \mathbf{b}\|_{\mathbf{w}} \to \min$, где $A \in \mathbb{C}^{m\times n}$ – прямоугольная матрица; $\mathbf{b} \in \mathbb{C}^m$ – столбцевой вектор;

```
x=lscov(A,B,V)
```

находит вектор $\mathbf{x} \in \mathbb{C}^n$, $n \le m$, такой, что, где $A \in \mathbb{C}^{m\times n}$ – прямоугольная матрица; $\mathbf{b} \in \mathbb{C}^m$ – столбцевой вектор

Пояснение.

1) $\tilde{\mathbf{a}} = (a_1\sqrt{w_1}, a_2\sqrt{w_2}, ..., a_n\sqrt{w_n})^T \Rightarrow \|\mathbf{a}\|_{\mathbf{w}} = |\tilde{\mathbf{a}}|$, и масштабированием строк матрицы A и элементов вектора $\mathbf{b}$ задача сводится к обычной форме метода наименьших квад$\|A\mathbf{x} - \mathbf{b}\|_{V^{-1}} \to \min$ратов.

2) Т.к.

необходимо $A^H V^{-1} A\mathbf{x} = A^H V^{-1}\mathbf{b}$

Пусть $V = TT^H$, T – нижняя треугольная матрица (например, разложение Холецкого). Обозначая $A_* = T^{-1}A$, $\mathbf{b}_* = T^{-1}\mathbf{b}$, находим:

$A_* A_*^H \mathbf{x} = A_*^H \mathbf{b}_*$

Выполняя QR-разложение $A_* = QR$, $Q^{-1} = Q^H$, R – верхняя треугольная матрица,

$\|A\mathbf{x} - \mathbf{b}\|_{V^{-1}}^2 = \mathbf{x} \cdot A^H V^{-1} A\mathbf{x} - 2\mathbf{x} \cdot A^H V^{-1}\mathbf{b} + \mathbf{b} \cdot V^{-1}\mathbf{b} \to \min$ $c = Q^H \mathbf{b}_*$,

приходим к системе линейных уравнений $R^H R\mathbf{x} = R^H \mathbf{c}$. Если R_* – левый верхний минор размерности $n \times n$ матрицы R, $\mathbf{c}_* = (c_1, c_2, ...c_n)^T$, то $R_*^H R_* \mathbf{x} = R_*^H \mathbf{c}_*$, и даже если R_* вырождена и решение не является единственным, можно полагать

$$R_* \mathbf{x} = \mathbf{c}_*$$

8. Решение задач на собственные значения и другие функции линейной алгебры

8.1. Собственные векторы и собственные значения

Пусть $A \in \mathbb{C}^{n \times n}$, $\mathbf{x} = (x_1, x_2, ..., x_n)^T$, $\lambda \in \mathbb{C}$

Собственные значения $\lambda_1, \lambda_2, ..., \lambda_n$ и принадлежащие им собственные векторы $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ матрицы A удовлетворяют условию

$$A\mathbf{x} = \lambda \mathbf{x}$$

откуда следует $\det(\lambda E - A) = 0$

Известно, что различным собственным значениям соответствуют линейно-независимые собственные векторы. Пусть

$$D = \mathrm{diag}\{\lambda_1, \lambda_2, ..., \lambda_n\}$$

диагональная матрица из собственных значений

Всегда из собственных векторов (вектор-столбцов) $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ можно составить матрицу X размерности $n \times n$, так, что

$$AX = XD$$

- Если все собственные значения различны, то матрица X будет невырожденная
- Если матрица A симметрична, то матрица X будет ортогональна
- Если матрица A Эрмитова, то матрица X будет унитарна

8.2. QR-алгоритм для нахождения спектра плотно заполненных матриц (LAPACK)

Пусть $A \in \mathbb{C}^{n \times n}$. Выполняя процесс ортогонализации Грама-Шмидта собственных векторов матрицы A и, возможно, присоединенных к ним векторов Жордана, приходим к теореме Шура:

Любую квадратную матрицу $A \in \mathbb{C}^{n \times n}$ можно представить в виде

$$A = QUQ^H$$

где $U = \{u_{kj}\}$, $1 \le k, j \le n$, $u_{kj} \ne 0$ при $j \ge k$ – верхняя треугольная матрица, а $Q^{-1} = Q^H$ – унитарная матрица. Так как

$$\det(\lambda E - A) = \det(\lambda E - U) = \prod_{k=1}^{n} (\lambda - u_{kk})$$

собственные значения матрицы A совпадают с диагональными элементами матрицы U. Разложение Шура обычно находится при помощи следующего итерационного метода, называемого QR-алгоритмом. Полагаем

$A_0 = A,$

и далее для $k = 0,1,2,...$ выполняем QR-разложение

$A_k = Q_k R_k$

а затем находим $A_{k+1} = R_k Q_k$.

Если все собственные значения матрицы A различны, то

$U = \lim\limits_{k\to\infty} A_k,\ \ Q = \prod_{k=0}^{\infty} Q_k$

QR-алгоритм гарантированно сходится к верхней треугольной матрице U, если все собственные значения матрицы A различны по абсолютной величине. Если матрица A имеет равные по модулю комплексно сопряженные собств. значения, то в матрице $\lim\limits_{k\to\infty} A_k$ могут возникать диагональные блоки размером 2×2.

8.3. Ускорение QR-алгоритма при помощи приведения исходной матрицы к верхней квазитреугольной форме

Матрица B имеет верхнюю квазитреугольную форму, если отличны от нуля лишь элементы на главной диагонали, выше нее и на диагонали, лежащей ниже главной, например

$$B = \begin{bmatrix} \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} \\ \text{x} & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} \\ 0 & \text{x} & \text{x} & \text{x} & \text{x} & \text{x} \\ 0 & 0 & \text{x} & \text{x} & \text{x} & \text{x} \\ 0 & 0 & 0 & \text{x} & \text{x} & \text{x} \\ 0 & 0 & 0 & 0 & \text{x} & \text{x} \end{bmatrix}$$

Известно, что любую матрицу некоторым унитарным преобразованием можно привести к верхней квазитреугольной форме, т.е.

$$A = Q_* B Q_*^H$$

B – верхняя квазитреугольная матрица; $Q_*^{-1} = Q^H$ – унитарная матрица. В самом деле, если $\mathbf{a} = (0, a_{21}, a_{31}, ..., a_{n1})^T$ – поддиагональная часть первого столбца матрицы, а нормированный вектор $\mathbf{v} = (0, v_2, ..., v_n)^T \in \mathbb{C}^n$, $|\ \mathbf{v}\ | = 1$ удовлетворяет условию $\mathbf{a}/\ |\ \mathbf{a}\ | = -2v_2\mathbf{v} + (0,1,...,0)^T$, то матрица отражения $H = E - 2\mathbf{v}\mathbf{v}^H$ позволяет обнулить все элпементы первого столбца матрицы $HAH^H = HAH$, расположенные ниже второго элемента. Аналогично для столбцов $2,3,..n$, и унитарная матрица Q_* представляет собой произведение соответствующих $n-2$ элементарных матриц отра-

жений.

Таким образом, QR-алгоритм модифицируется следующим образом:

$$A = Q_* A_0 Q_*^H$$

где A_0 – верхняя квазитреугольная матрица; $Q_*^{-1} = Q_*^H$ – унитарная матрица;

$$A_k = Q_k R_k,\ A_{k+1} = R_k Q_k,\ k = 0,1,2,...$$

$$U = \lim_{k\to\infty} A_k,\ \ Q = Q_* \prod_{k=0}^{\infty} Q_k$$

Матрицы A_k являются верхними квазитреугольными, и для их QR-разложения требуется обнулить не более чем $n-1$ поддиагональных элементов посредством $n-1$ элементарных преобразований вращения.

- Матрицы A_k при $k \gg 1$ являются приближенно являются треугольными, а их диагональные элементы приближенно равны собственным значениям. Для ускорения сходимости в QR-алгоритм вводят сдвиги на величины σ_k, приближенно равные какому-либо из собственных значений:

  $$A_k - \sigma_k E = Q_k R_k,\ A_{k+1} = R_k Q_k + \sigma_k E,\ k = 0,1,2,...$$

  Например, $\sigma_k = a_{nn}^{(k)}$ – самый нижний элемент главной диагонали матрицы A_k.

- Процедура исчерпывания. Если в матрице $A_k - a_{nn}^{(k)} E$ обнуляется с требуемой степенью точности последняя строка, то $a_{nn}^{(k)}$ – с требуемой степенью точности искомое собственное значение. В матрице A_k отбрасывается последняя строка и последний столбец, и далее QR-алгоритм применяется к ее верхней левой квазитреугольной подматрице размерности $n-1$.
- В случае комплексно-сопряженных собственных значений на четном шаге в качестве σ_k выбирают приближенное значение одного собственного значения, а на нечетном – сопряженную к нему величину.

8.4. Обобщенная задача на собственные значения

Пусть $A, B \in \mathbb{C}^{n\times n}$, $\mathbf{x} = (x_1, x_2, ..., x_n)^T$, $\lambda \in \mathbb{C}$

Обобщенная задача нахождения собственных значений $\lambda_1, \lambda_2, ..., \lambda_n$ и принадлежащих им собственных векторов $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ имеет вид

$$A\mathbf{x} = \lambda B\mathbf{x}$$

откуда следует $\det(\lambda B - A) = 0$

Если матрица B невырождена, то формально обобщенная задача на собственные значения сводится к задаче на собственные значения для мат-

рицы $B^{-1}A$. Если матрица B вырождена, то формально некоторые собственные значения оказываются бесконечно большими.

Аналогично, из собственных векторов (вектор-столбцов) $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ можно составить матрицу X размерности $n \times n$, так, что

$$AX = BXD$$

где D – диагональная матрица, составленная из собственных значений.

Численное решение обобщенной задачи на собственные значение обеспечивается на основе QZ-алгоритма, родственного QR-алгоритму.

8.5. Функции MATLAB для решения задач на собственные значения

Для плотно заполненных матриц функция **eig** (SciLAB - **spec**) обеспечивает нахождение спектра. Она используют пакет LAPACK.
Функция

```
d=eig(A)
```

возвращает вектор длины n, составленный из собственных значений квадратной матрицы A
Вызов функции:

```
d=eig(A,B)
```

возвращает вектор длины n, составленный из обобщенных собственных значений квадратной матриц A, B
Вызов:

```
[X,D]=eig(A)
```

возвращает матрицу из собственных векторов X и диагональную матрицу из собственных значений D задачи на собственные значения, удовлетворяющие соответствующему уравнению
Вызов функции:

```
[X,D]=eig(A,B)
```

возвращает матрицу из собственных векторов X и диагональную матрицу из собственных значений D задачи на собственные значения, удовлетворяющие соответствующему уравнению.

8.6. Частичная задача на собственные значения и спектральные преобразования

Для разреженных матриц весьма большой размерности Требуется найти некоторое подмножество собственных значений задач

$$A\mathbf{x} = \lambda\mathbf{x} \tag{3}$$

либо

$$A\mathbf{x} = \lambda B\mathbf{x} \tag{4}$$

Здесь $N \gg 1$, $A \in \mathbb{C}^{N\times N}$, $B \in \mathbb{C}^{N\times N}$, $\mathbf{x} = (x_1, x_2, ..., x_N)^T \in \mathbb{C}^N$

- Нахождение ближайшего к λ_0 собственного значения λ обобщенной задачи (4) при помощи преобразования

$$A\mathbf{x} = [(\lambda - \lambda_0) + \lambda_0]B\mathbf{x} \Rightarrow$$

$$(A - \lambda_0 B)^{-1} B\mathbf{x} = \sigma\mathbf{x},\ \sigma = \frac{1}{\lambda - \lambda_0},\ \lambda = \lambda_0 + \frac{1}{\sigma}$$

сводится к нахождению наибольшего собственного значения σ задачи (5).

- Пусть

$$P(t) = \sum_{k=0}^{n} a_k t^k,\ a_k \in \mathbb{C},\ t \in \mathbb{C}$$

– некоторый полином. Тогда

$$A\mathbf{x} = \lambda\mathbf{x} \ \Rightarrow\ P(A)\mathbf{x} = P(\lambda)\mathbf{x}$$

8.7. Приближенное определение $n \ll N$ максимальных собственных значений

Пусть $A \in \mathbb{C}^{N\times N}$ – разреженная матрица размера $N \gg 1$, $\mathbf{v} = (v_1, v_2, ..., v_N) \in \mathbb{C}^N$ – некоторый вектор.

Линейное подпространство Крылова размерности $n \ll N$ суть

$$\mathcal{K}_n(A, \mathbf{v}) = \mathrm{span}\{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, ..., A^{n-1}\mathbf{v}\}$$

Упорядоченные по убыванию абсолютной величины собственные значения λ_1, λ_2,..., λ_n исходной матрицы A приближенно равны собственным значениям $\hat{\lambda}_1$, $\hat{\lambda}_2$,..., $\hat{\lambda}_n$ матрицы $\hat{A} \in \mathbb{C}^{n\times n}$, являющейся проекцией связанного с матрицей A линейного оператора на подпространство Крылова. Собственные векторы $\mathbf{x}_1$, $\mathbf{x}_2$,..., $\mathbf{x}_n$ матрицы A линейно выражаются через базис пространства Крылова и компоненты собственных векторов $\hat{\mathbf{x}}_1$, $\hat{\mathbf{x}}_2$,..., $\hat{\mathbf{x}}_n$ матрицы $\hat{A}$. Проектирование линейного оператора на подпространство Крылова обычно выполняется на основе процедуры Грама-Шмидта ортонормирования базиса подпространства Крылова

$\mathbf{v}_1 = \mathbf{v}/|\mathbf{v}|$,

$$\alpha_{k+1}\mathbf{v}_{k+1} = A\mathbf{v}_k - \sum_{j=1}^{k}\beta_{jk}\mathbf{v}_j, \quad |\,\mathbf{v}_{k+1}\,| = 1,$$
$$\beta_{jk} = \mathbf{v}_j \cdot A\mathbf{v}_k, \;\; \alpha_{k+1} = \left|A\mathbf{v}_k - \sum_{j=1}^{k}\beta_{jk}\mathbf{v}_j\right|, \;\; k = 1,2,...,n \qquad (4)$$

что приводит к т.н. факторизации Арнольди

$$A[\mathbf{v}_1,\mathbf{v}_2,...,\mathbf{v}_n] = [\mathbf{v}_1,\mathbf{v}_2,...,\mathbf{v}_n]\hat{A} + [0,0,...,\alpha_{n+1}\mathbf{v}_{n+1}]$$

$$\hat{A} = \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & ... & \beta_{1,n-1} & \beta_{1n} \\ \alpha_2 & \beta_{22} & \beta_{23} & ... & \beta_{2,n-1} & \beta_{2n} \\ 0 & \alpha_3 & \beta_{33} & ... & \beta_{3,n-1} & \beta_{3n} \\ ..... & ..... & ..... & ..... & ..... & ..... \\ 0 & 0 & \alpha_4 & ... & \beta_{n-1,n-1} & \beta_{n-1,n} \\ 0 & 0 & 0 & ... & \alpha_n & \beta_{nn} \end{bmatrix} \qquad (5)$$

Если начальный вектор $\mathbf{v}$ является линейной комбинацией лишь $m < n$ собственных векторов исходной матрицы A, то $\dim \mathcal{K}_n(A,\mathbf{v}) = m$, и формулы (4) обрываются при $k = m$, т.к. $\alpha_{m+1} = 0$, т. е. матрица $\hat{A}$ имеет меньшую размерность $m \times m$. В этом случае также можно расширить матрицу $\hat{A}$ до размерности $n \times n$. Достаточно в качестве $\mathbf{v}_{m+1}$ выбрать любой вектор единичной длины, ортогональный к векторам $\mathbf{v}_1$, $\mathbf{v}_2$,…, $\mathbf{v}_m$.

8.8. Неявный итерационный метод Ланцоша-Арнольди (ARPACK)

Требуется найти $m \ll N$ собственных значений и векторов разреженной матрицы A размерности $N \times N$. Выбираем целое число $n = m + p$, $p > 0$, $p \ll N$.

1) Выбираем вектор $\mathbf{v}$ и согласно (4) с учетом прим. 1, 2 находим $\mathbf{v}_1$, $\mathbf{v}_2$,…, $\mathbf{v}_n$ и $\hat{A}$ в разложении Арнольди (5).

2) Находим собств. значения $\hat{\lambda}_1$, $\hat{\lambda}_2$,…, $\hat{\lambda}_n$ матрицы $\hat{A}$ и выбираем «сдвиги» $\mu_j = \hat{\lambda}_{p+j}$, $j = 1,2,...,m$.

3) При $j = 1,2,...,p$ выполняются преобразования

a) находится QR-разложение $\hat{A} - \mu_j E = Q_j R_j$

b) $[\mathbf{v}_1,\mathbf{v}_2,...,\mathbf{v}_n] \leftarrow [\mathbf{v}_1,\mathbf{v}_2,...,\mathbf{v}_n]Q_j$, $\hat{A} \leftarrow Q_j^H \hat{A} Q_j$

(при этих преобразованиях $\hat{A}$ остается верхней квазитреугольной)

4) Сохраняем векторы $\mathbf{v}_1$, $\mathbf{v}_2$,…, $\mathbf{v}_m$ и первые m столбцов матрицы $\hat{A}$. Далее по формулам (4) при $k = m+1,...,n$ снова вычисляем векторы

$\mathbf{v}_{m+1}$, $\mathbf{v}_{m+2}$,…, $\mathbf{v}_n$ и столбцы матрицы $\hat{A}$ с номерами $m+1, m+2, ..., n$.

5) Переходим к п. 2)

8.9. Вызовы MATLAB (Octave) для решения частичной задачи на собственные значения

`d=eigs(A)`

– возвращает вектор из 6 наибольших собственных значений матрицы A

`[D,V]=eigs(A)`

– возвращает диагональную матрицу D из 6 наибольших по абсолютной величине собственных значений матицы A и матрицу V, столбцы которой представляют собой принадлежащие им собственные векторы;

`d=eigs(A,B); [D,V]=eigs(A,B)`

– поиск 6 наибольших собственных значений и принадлежащих им собственных векторов обобщенной задачи на собств. зн. $A\mathbf{x} = \lambda B\mathbf{x}$

`d=eigs(A,n); [D,V]=eigs(A,n)`

`d=eigs(A,B,n); [D,V]=eigs(A,B,n)`

– поиск n наибольших собственных значений и принадлежащих им собственных векторов

`d=eigs(A,n,sigma); [D,V]=eigs(A,n,sigma)`

`d=eigs(A,B,n,sigma); [D,V]=eigs(A,B,n,sigma)`

- Если `sigma` – числовой скаляр, то ищутся ближайшие к нему собственные значения.
- Если `sigma` – строка символов, равная: a) `'lm'`, то ищутся наибольшие собств. значения; b) `'sm'`, то ищутся наименьшие собств. значения.

Стандартные средства пакета ARPACK не требуют явного вычисления разреженных матриц A и B; вместо этого необходимо указать функцию, соответствующую умножению на матрицу столбцевого вектора соответствующей размерности.

`eigs(AFun,N,…)`

`AFun` – дескриптор функции, соотв. конечномерному линейному оператору, а **`N`** – размерность задачи. По умолчанию, ищутся наибольшие собственные значения, и предполагается, что для столбцевого вектора **`x`** функция **`AFun`** возвращает величину $A\mathbf{x}$.

Если используется вызов **`eigs(AFun,N,…)`** и среди параметров имеется параметр `sigma`:

- Если `sigma` – строка символов, равная `'lm'`, то ищутся наибольшие собств. значения, а функция **`AFun`** для столбцевого вектора **`x`** должна возвращать величину $A\mathbf{x}$.
- Если `sigma` – строка символов, равная `'sm'`, то ищутся наименьшие собств. значения, а функция **`AFun`** для столбцевого вектора **`x`** должна возвращать величину $A^{-1}\mathbf{x}$.

- Если `sigma` – числовой скаляр, то ищутся ближайшие к нему собственные значения. В случае задачи на собственные значения функция **`AFun`** для столбцевого вектора **`x`** должна возвращать величину $(A-\sigma E)^{-1}\mathbf{x}$, а в случае обобщенной задачи на собственные значения должна возвращаться величина $(A-\sigma B)^{-1}B\mathbf{x}$

Пример. Найти 10 наименьших собственных значений разреженной комплексной матрицы A размерности $N\times N$, ненулевые элементы которой суть

$A_{kk}=10+k,\ k=1,2,..,N$

$A_{k,k+1}=i,\ A_{k+1,k}=1,\ k=1,2,..,N-1$

$A_{N1}=-1,\ A_{1N}=-i$

Решение.

```
function Lam=Eigs_Tst(N)
   Diags=zeros(N,3);
   Diags(1:N,1)=10+(1:N);
   Diags(1:N,2)=i;
   Diags(1:N,3)=1;
   tic;
   A=spdiags(Diags,[0,1,-1],N,N);
   A(N,1)=-1;
   A(1,N)=-i;
   toc;
   tic;
   Lam=eigs(A,10,'sm');
   toc;
end
```

Нахождение первых 10 наименьших собственных значений разреженной матрицы с комплексными элементами размерности 1000000x1000000 реализуется вызовом

```
>>Eigs_Tst(1000000)
```

В результате найдено

```
ans =
  10.7529 - 0.7925i
  12.1882 - 0.2621i
  13.0653 + 0.0513i
  13.9937 + 0.0037i
  14.9999 - 0.0003i
  16.0000 - 0.0000i
  17.0000 + 0.0000i
  18.0000 + 0.0000i
  19.0000 - 0.0000i
```

```
20.0000 - 0.0000i
```

8.10. Некоторые функции линейной алгебры

Пусть $A = \{a_{kj}\},\ k = 1,2,...,m,\ j = 1,2,...,n.$ По определению, след матрицы – это сумма ее диагональных элементов:

$$\mathrm{Tr}\, A = \sum_{k=1}^{\min(m,n)} a_{kk}$$

Функция T = trace(A) возвращает след матрицы A, т.е. сумму ее диагональных элементов.

```
theta = subspace(A, B)
```

в пространстве наибольшей размерности возвращает угол между двумя гиперплоскостями. Первая гиперплоскость задается столбцами матрицы A, вторая – столбцами матрицы B.

```
nrm = norm(A);
nrm = norm(A, p)
```

вычисление норм векторов и матриц. По умолчанию, p = 2. По определению, p-норма вектора *a* суть:

$$\left\|\mathbf{a}\right\|_p = \sum_{j=1}^{m} |\, a_j \,|^p \,^{1/p}$$

в частности,

$$\left\|\mathbf{a}\right\|_2 = |\, \mathbf{a} \,|,\ \left\|\mathbf{a}\right\|_1 = \sum_{j=1}^{m} |\, a_j \,|,\ \left\|\mathbf{a}\right\|_\infty = \max_{1 \le j \le m} |\, a_j \,|$$

По определению, p-норма матрицы $A = \{a_{kj}\}$ размерности n x m суть:

$$\left\|A\right\|_p = \sup_{\mathbf{x} \neq \mathbf{0}} \frac{\left\|A\mathbf{x}\right\|_p}{\left\|\mathbf{x}\right\|_p}$$

При p = 2 $\left\|A\right\|_2$ - это квадратный корень из максимального собственного значения матрицы. $\left\|A\right\|_2 = \sqrt{\lambda}$

$$\left\|A\right\|_1 = \max_j \sum_i |\, a_{ij} \,|,\ \left\|A\right\|_\infty = \max_i \sum_j |\, a_{ij} \,|$$

Норма Фробениуса матрицы A суть:

$$\left\|A\right\|_F = \sum_{k=1}^{m} \sum_{j=1}^{n} |\, a_{kj} \,|^2 \,^{1/2}$$

Для матриц параметр p может принимать значения 1, 2, $\inf(\infty)$, ‘fro’, при-

чем значение ‘fro’ параметра p означает вычисление нормы Фробениуса.

```
nrm = normest(A);
nrm = normest(A, tol)
```

для разреженной матрицы A выполняет оценку величины $\|A\|_2$. Если указывается второй параметр, то вместо значения по умолчанию относительной погрешности 10^{-6} используется относительная погрешность tol.

```
U = schur(A);
[Q, U] = schur(A);
```

Применяется лишь к плотно заполненным квадратным матрицам A и возвращает унитарную матрицу Q и верхнее треугольную матрицу U, являющиеся компонентами формы Шура: $A = QUQ^H$, $Q^{-1} = Q^H$ исходной матрицы A.

В матричных вычислениях достаточно часто используется так называемые сингулярное разложение матрицы и сингулярные числа матрицы. Пусть A – некоторая, вообще говоря, прямоугольная матрица размерности m x n. Легко проверить следующие факты:

- матрицы $A^H A$ размерности n x n и AA^H размерности m x m являются Эрмитовыми и состоят из скалярных произведений столбцов и строк соответственно исходной матрицы A;
- по указанной причине собственные значения как матрицы $A^H A$, так и матрицы AA^H не могут быть отрицательными;
- если x – некоторый собственный вектор матрицы $A^H A$, принадлежащий собственному значению $\lambda \geq 0$, т.е. $A^H Ax = \lambda x$, то $y = Ax$ суть собственный вектор матрицы AA^H, принадлежащий тому же самому собственному значени$\lambda \geq 0$, т.е. $AA^H y = \lambda y$.

Отсюда следует:

- ненулевые собственные значения матрицы $A^H A$ и AA^H совпадают;
- можно так выбрать унитарную матрицу Q размерности n x n из взаимно ортогональных собственных векторов матрицы $A^H A$ и унитарную матрицу Z размерности m x m из взаимно ортогональных собственных векторов матрицы AA^H, что:

$$Z^H AQ = S,$$

где $S = \mathrm{diag}\{s_1, s_2, ..., s_{\min(m,n)}\}$ - диагональная матрица размерности m x n, у которой отличны от нуля лишь упорядоченные по возрастанию элементы главной диагонали $s_j \geq 0$, называемые сингулярными числами исходной матрицы A. Т.е. сингулярное разложение матрицы A имеет вид:

$A = ZSQ^H$,

причем $A^H A = QS^H SQ^H$, $AA^H = Z^H SS^H Z$

Отсюда следует, что сингулярные числа суть $s_j = \sqrt{\lambda_j} \geq 0$, где λ_j - собственные значения матрицы $A^H A$ либо матрицы AA^H.

Для нахождения сингулярного разложения плотно заполненной прямоугольной матрицы A используется функция svd.

```
S = svd(A)
```

Данная функция возвращает вектор s сингулярных значений матрицы A.
Вызов функции:

```
[Z,S,Q]=svd(A)
```

возвращает компоненты сингулярного разложения матрицы $A = ZSQ^H$
Вызов

```
[Z,S,Q]=svd(A, 'econ')
```

возвращает «экономное» сингулярное разложения матрицы A размерности m x n. Если $m \geq n$, то вычисляется лишь n столбцов матрицы Z, а диагональная матрица S имеет размерность n x n. В противном случае вычисляется m столбцов матрицы Q, а диагональная матрица S имеет размерность m x m.

Функция svds используется для нахождения отдельных сингулярных значений и сингулярных векторов разреженных матриц значительного размера (фактически используется алгоритм, сходный с алгоритмом функции eigs).

```
S = svds(A)
```

Находит 6 наибольших сингулярных значений матрицы A;

```
S = svds(A, k)
```

Находит k наибольших сингулярных значений матрицы A;

```
S = svds(A, k, sigma)
```

Находит k ближайших к значению sigma сингулярных значений матрицы A;

```
[Z,S,Q]=svds(A, …)
```

Возвращает диагональную матрицу S из нескольких сингулярных значений матрицы A, а также матрицы Z и Q, столбцы которых являются сингулярными векторами матрицы A.

Функция

```
K = rank(A)
K = rank(A, tol)
```

Находит ранг плотно заполненной числовой матрицы A, т.е. число сингулярных значений матрицы A, превосходящих либо точность по умолчанию, либо значение tol.

Функция

```
K = sprank(A)
```

Возвращает так называемй структурный ранг разреженной матрицы A, представляющий собой наибольшую верхнюю границу ее числового ранга.

Функция

```
B = orth(A)
```

Возвращает ортогональный базис линейного пространства, связанного со столбцами плотно заполненной числовой матрицы A.

Функция

```
Z = null(A)
```

Применима лишь к плотно заполненными матрицам и возвращает ортонормированный базис ядра линейного оператора, связанного с матрицей A. Ядро, или нулевое пространство матрицы (линейного оператора) A определяется множеством таких векторов x, что $Ax = 0$.

Функция

```
C = cond(A)
C = cond(A, p)
```

Возвращает так называемые числа обусловленности квадратной матрицы A по отношению к операции обращения в p-норме. По умолчанию, p = 2, и число обусловленности представляет собой отношение наибольшего сингулярного значения матрицы A к ее наименьшему сингулярному числу. Для разреженных матриц параметр p игнорируется и принимается равным 2.

Функция

```
C = condest(A)
```

Оценивает нижнюю границу для числа обусловленности матрицы A.

8.11. Задача о нахождении собственных значений и собственных векторов матричного полинома

Задача о нахождении собственных значений и собственных векторов т.н. матричного полинома имеет вид:

$$(A_0 + A_1\lambda + A_2\lambda^2 + ... + A_m\lambda^m)\mathbf{x} = \mathbf{0}$$

Здесь $\lambda \in \mathbb{C}$, $A_0, A_1, ..., A_m \in \mathbb{C}^{n\times n}$ - некоторые квадратные матрицы размерности n x n, $\mathbf{x} \in \mathbb{C}^n$. Аналогично построению сопровождающей матрицы для нахождения корней обычных полиномов, данная задача сводится в обобщенной задаче на собственные значения для двух блочных матриц A и B размерности mn x m (квадратных матриц большей размерности:

$$A = \begin{bmatrix} A_0 & 0 & 0 & ... & 0 & 0 \\ 0 & E & 0 & ... & 0 & 0 \\ 0 & 0 & E & ... & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & ... & E & 0 \\ 0 & 0 & 0 & ... & 0 & E \end{bmatrix},$$

$$B = \begin{bmatrix} -A_1 & -A_2 & -A_3 & \dots & -A_{m-1} & -A_m \\ E & 0 & 0 & \dots & 0 & 0 \\ 0 & E & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & E & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & E & 0 \end{bmatrix}$$

Функции

```
lam=polyeig(A0,A1,…,Am);
[X,lam]=polyeig(A0,A1,…,Am);
```

Возвращают вектор lam размерностри mn из собственных значений матричного полинома, а также матрицу X размерности n x mn, столбцы которой являются собственными векторами матричного полинома.

9. Матричные функции и обработка данных

9.1. Элементарные функции по обработке данных в MATLAB

С точки зрения математического определения, подразумевается, что аргументы рассматриваемых в данной главе функций MATLAB должны быть векторами некоторой длины. По указанным соображениям, рассматриваемые функции возвращают скалярные значения в тех случаях, когда их входными параметрами являются векторы-строки или векторы-столбцы. Если же входными параметрами являются матрицы, то возвращаемым значение будет вектор-строка, элементы которого являются результатом вычисления соответствующих операций над столбцами входных матриц.

Функция

```
C = max(A)
```

Возвращает максимальный по модулю элемент вектора.

```
[C, I] = max(A)
```

Возвращает максимальный по модулю элемент (C) и его индекс (I).

```
C = max(A, B)
```

Возвращает массив той же размерности, что и массивы A и B, содержащий в соответствующих позициях максимальные по абсолютной величине значения элементов массивов A и B. Размерности которых должны совпадать.

```
C = max(A, [], dim)
```

Поиск максимального элемента в векторах, которые задаются индексом многомерного массива с номером dim.

Аналогично указанным функциям используется функция:

```
C = min(A)
[C, I] = min (A)
C = min (A, B)
C = min (A, [], dim)
```

Среднее арифметическое элементов вектора:

```
M = mean(A)
M = mean(A, dim)
```

Сортировка элементов вектора

```
B = sort(A)
B = sort (A, dim)
B = sort (A, dim, mode)
[B, I] = sort (A, dim , mode)
```

Параметр mode может принимать одно из двух значений: ‘ascend’ (по возрастанию) или ‘descend’ (по убыванию). Выходной параметр I содержит индексы, характеризующие перестановки элементов вектора при его сортировке. Сортировка комплексных элементов осуществляются сначала по их модулю, а затем по аргументу.

Пусть $\mathbf{a} = (a_1, a_2, ..., a_n)^T$ - некоторый вектор, в котором выполнена сортировка элементов по возрастанию, т.е. $| a_{k+1} | \geq | a_k |$. Если вектор вектор содержит N = 2n + 1 элементов, то под его медианным значением понимают величину a_n. В противном случае - $(a_{[n/2]} + a_{[(n-1)/2]}) / 2$.

Функция

```
M = median(A)
M = median(A, dim)
```

Находит медианное значение вектора A (либо векторов в составе многомерного массива).

Функция

```
M = mode(X)
M = mode(X, dim)
```

Возвращает наиболее часто встречающееся значение в векторе X.
Вызов

```
[M, F] = mode(X, dim)
```

дополнительно возвращает число вхождений в вектор X наиболее часто встречающегося значения M.

Сумма и произведение элементов вектора A вычисляются функциями:

```
S = sum(A)
S = sum(A, dim)
P = prod(A)
P = prod(A, dim)
```

Вектор B, каждый элемент которого представляет сумму соответствующего элемента вектора A и всех предшествующих ему элементов, возвращается функцией:

```
B = cumsum(A)
B = cumsum(A, dim)
```

Аналогичего, вектор B, каждый элемент которого представляет собой произведение соответствующего элемента вектора A и всех предшествующих ему элементов, возвращается функцией:

```
B = cumprod(A)
B = cumprod(A, dim)
```

9.2. Функции от матриц

Пусть $_{\text{s}}$, $z, f \in \mathbb{C}$ аналитичная функция комплексной переменной z, и соответствующий степенной ряд сходится в некотором круге $|\, z \,| < R$. Тогда, если A – некоторая квадратная матрица размерности n x n, причем $\left\|A\right\|_F < R$, то ряд $f(A) = \sum_{k=0}^{\infty} f_k A^k$ сходится. Тем самым это ряд в той же сфере $\left\|A\right\|_F < R$ определяет функцию от квадратной матрицы A. Из

линейной алгебры известно, что после решения полной задачи о нахождении собственных значений матрицы A, преобразованием подобия матрица A может быть приведена к так называемой канонической форме Жордана. В таком случае, вычисление матричного ряда f(A) сводится к вычислению f(z) и, возможно, некоторых ее производных $f^{(k)}(z) = d^k f(z) / dz^k$ на множестве собственных значений матрицы A. Заметим, что при этом потребуется не более чем m-1 производных функции f(z), где m – наибольшая кратность собственных значений матрицы A. Заметим также, что подобный способ вычисления функций от матриц может иметь смысл и за пределами области сходимости матричного ряда $\|A\|_F < R$ как некоторый вариант аналитического продолжения функции f(A).
Функция

```
F = funm(A, func)
```

выполняет вычисление функции f(A). При этом, A – квадратная плотно заполненная матрицы, func – дескриптор функии, возвращающей значения f(z) и некоторых ее производных.
Заметим, что функция

```
w = func(z, k)
```

должна принимать 2 аргумента: вектор z и целое число k, и возвращать вектор w той же размерности, что и z, причем $w_j = f^{(k)}(z_j)$.

9.3. Матричные математические функции MATLAB

При вычислении следующих функций от квадратных плотно заполненных матриц используются специальные точные и производительные алгоритмы.

Функция

```
F = expm(A)
```

возвращает матричную экспоненту (e^A) на основе аппроксимации Падэ.

Функция

```
F = logm(A)
```

возвращает логарифм матрицы $\ln A$

Функция

```
X = sqrtm(A)
```

возвращает квадратный корень из матрицы A, т.е. $X^2 = A$

9.4. Статистические функции

Функция

```
S = std(X)
```

Возвращает стандартное отклонение для вектора $\mathbf{x} = (x_1, x_2, ..., x_n)^T$

$$V = \frac{1}{n-1}\sum_{k=1}^{n} | x_k - \overline{x} |^2, \; S = \sqrt{V}, \; \overline{x} = \frac{1}{n}\sum_{k=1}^{n} x_k$$

Где V – величина вариации для указанного вектора (квадрат стандартного отклонения) возвращается функцией:

```
V = var(X)
```

Вызовы данных функций:

```
S = std(X, flag)
V = var(X, flag)
```

При flag = 1 используют следующие формулы для вычисления отклонения и вариации:

$$V = \frac{1}{n}\sum_{k=1}^{n} | x_k - \overline{x} |^2, \; S = \sqrt{V}, \; \overline{x} = \frac{1}{n}\sum_{k=1}^{n} x_k$$

По умолчанию flag = 0. Тогда используется первый вариант формул.

Для многомерных массивов X возвращаются соответствующие значения отклонения и вариации содержащихся в массиве векторов.

Вызовы

```
S = std(X, flag, dim)
V = var(X, flag, dim)
```

Возвращают соответствующие величины для набора векторов, содержа-

щихся в многомерном массиве X и определяемых индексом с номером dim.

Функция

```
V = var(X, w)
```

Вычисляет вариацию с использованием весового вектора w с положительными элементами. При этом вектор w нормируется так, что сумма его элементов равна 1.

$w_k > 0\,, \sum_{k=1}^{n} w_k = 1$

Ковариация двух случайных величин x1 и x2 суть

$$\mathrm{cov}(x_1, x_2) = \mathbb{E}((x_1 - \mu_1)(x_2 - \mu_2)),\ \mu_1 = \mathbb{E}x_1,\ \mu_2 = \mathbb{E}x_2$$

Где $\mathbb{E}$ - математическое ожидание.
Вычисление матрицы ковариации осуществляется функцией:

```
C = cov(X)
```

Если X – вектор, то данная функция вернет его вариацию. Если же X - матрица, то ее строки рассматриваются как результаты наблюдений в фиксированный момент времени, а ее столбцы рассматриваются как наборы значений отдельных случайных величин. Возвращаемое значение является матрицей ковариации для столбцов исходной матрицы X. Диагональные элементы V суть вариации столбцов исходной матрицы X.

Вызов функции

```
C = cov(X, Y)
```

Возвращает матрицу ковариации для векторов X и Y, которые должны иметь одинаковую длину. В данном случае всегда возвращается матрица размерности 2 x 2. Если X и Y – матрицы, то они рассматриваются как одномерные векторы и, следовательно, они должны содержать одинаковое число элементов. По умолчанию выполняется нормировка на величину N – 1, где N – число наблюдений.

При вызовах функции

```
C = cov(X, 1)
C = cov(X, Y, 1)
```

Выполняется нормировка на величину N.

Коэффициенты корреляции определяются по матрице ковариации $C = \{c_{ij}\}$ следующим образом:

$$r_{ij} = \frac{c_{ij}}{\sqrt{c_{ii}c_{jj}}}$$

Их можно получить с помощью функции:

```
R = corrcoeff(X)
R = corrcoeff(X, Y)
[R, P] = corrcoeff(…)
```

Последний вызов дополнительно возвращает матрицу вероятностей, внедиагональные элементы которой p_{ij} связаны с процервой гипотезу о независимости случайных величин x_i и x_j. Если величина p_{ij} достаточно мала, то влияние коэффициента корреляции r_{ij} существенно.

Вызов функции:

```
[R, P, RLo, RUp] = corrcoeff(…)
```

Дополнительно генерирует верхнюю (RUp) и нижнюю (RLo) границы доверительного интервала для коэффициентов корреляции r_{ij}.

10. Интерполяция данных в MATLAB

10.1. Одномерная интерполяция

Функция

```
yi = interp1(x, Y, xi)
```

выполняет интерполяцию для нахождения значений функции yi в точках xi. Интерполирование выполняется по известным величинам первичного аргумента x и соответствующих им значениям функции Y. Параметр x должен быть действительным вектором. Параметр Y может быть скаляром, вектором либо либо массивом с любым числом измерений.

Если Y – вектор, то длины векторов x и Y должны совпадать.

Если Y - скаляр, то его длина автоматически расширяется до длины вектора x.

Если Y - многомерный массив, то его длина по первому измерению должна совпадать с длиной вектора x.

Параметр xi можеть быть скаляром, вектором или многомерным масси-

вом.
Соответсвие размерностей параметров показаны в таблице:

x	n
Y	$n \times n_2 \times ... \times n_k$
xi	$m_1 \times m_2 \times ...m_j$
yi	$m_1 \times m_2 \times ...m_j \times n_2 \times ... \times n_k$

Функция

```
yi = interp1(Y, xi)
```

использует в качестве x вектор **1:n**, где n – размерность Y по первому измерению.

Функция

```
yi = interp1(x, Y, xi, method)
```

позволяет выбрать желаемый метод интерполирования:

'nearest'	Интерполяция значением функции в ближайшей точке
'linear'	Линейная интерполяция (по умолчанию)
'spline'	Интерполяция кубическим сплайном с сохранением непрерывности 1-й и 2-й производных
'pchip'	Интерполяция посредством кубических полиномов Эрмита; сохраняется непрерывность 1-й производной

Функция

```
yi = interp1(x, Y, xi, method, 'extrap')
```

выполняет экстраполяцию для значений xi, выходящих за пределы диапазона изменения x.

Функция

```
yi = interp1(x, Y, xi, method, extrapval)
```

запоняет значением **extrapval** в массиве **yi** те элементы, которые соответствуют элементам **xi**, выходящим за пределы изменения **x.**

Функция

```
pp = interp1(x,Y,xi,method,'pp');
```

генерирует наборы коэффициентов соответствующих интерполяционных полиномов. Непосредственная интерполяция далее может быть выполнена следующей функцией:

```
y = ppval(pp, xi)
```

10.2. Тригонометрическая интерполяция периодических функций

Функция

```
y = interpft(x, n),
```

выполняет одномерную тригонометрическую интерполяцию периодических функций. Входной вектор x содержит значения некоторой периодической функции в раноотстоящих точках. Выходной вектор y содержит значения периодической функции, рассчитанные в n равноотстоящих точках. Если x – вектор, то число n не может быть меньше его размерности. Если x - матрица, то все операции выполняются над столбцами вектора x, а возвращаемая матрицы содержит n строк и столько столбцов, сколько их в матрице x.

Функция

```
y = interpft(x, n, dim)
```

обрабатывает измерение, указанное в параметре dim.

10.3. Подготовка двумерных и трехмерных сеток для интерполяции

Функция

```
[X, Y] = meshgrid(x, y)
```

Готовит сетку для двумерной интерполяции по наборам независимых переменных x и y в соответствующих векторах. Выходные параметры X и Y представляют собой матрицы одинаковой размерности $m \times n$, где m - длина вектора x, а n – длина вектора. Строки выходной матрицы X представляют собой копии входного векотра x, а столбцы выходной матрицы Y представляют собой копии входного вектора y.

Функция

```
[X, Y] = meshgrid(x)
```

является аналогом предыдущей функии, за тем исключением, что вмето y повторно используется x.

Для подготовки наборов независимых переменных для трехмерной интерполяции используется

Функция

```
[X, Y, Z] = meshgrid(x, y, z)
```

Интерполяция значений функции z = f(x, y) двух независимых переменных в прямоугольной области обеспечивается следующей функций:

```
ZI = interp2(X, Y, Z, XI, YI)
```

Данная функция возвращает матрицу значений функции z = f(x, y), вычисленных посредством интерполяции для значений независимых переменных в матрицах XI и YI. Узлы сетки для интерполирования задаются в матрицах X и Y, а значение функции – в матрице Z. Требуется, чтобы значения независимых переменных x и y изменялись монотонно.

Функция

```
ZI = interp2(Z, XI, YI)
```

подразумевает X = 1:n, Y = 1:m, где n и m – размерности матрицы Z.

Функция

```
ZI = interp2(Z, ntimes)
```

вставляет в выходной массив дополнительные промежуточные точки, рассчитанные посредством линейной интерполяции так, что обе размерности матрицы ZI увеличиваются в ntimes раз по сравнению с исходной матрицей Z.

Функция

```
ZI = interp2(X, Y, Z, XI, YI, method)
```

позволяет дополнительно указать используемый для интерполяции метод.

'nearest'	Интерполяция значением функции в ближайшей точке
'linear'	Линейная интерполяция (по умолчанию)
'spline'	Интерполяция кубическими сплайнами
'cubic'	Кубическая интерполяция, если используется равномерная сетка изменения независимых переменных. В противном случае – кубический сплайн

Функция

```
ZI = interp2(X, Y, Z, XI, YI, method, extrapval)
```

заполняет в массиве ZI значением extrapval те элементы, которые соответствуют значениям XI либо YI, выходящим за пределы изменения элементов массивов X и Y.

Трехмерная интерполяция выполняется функцией interp3. Ее вызовы и параметры аналогичны таковым для функции interp2:

```
VI = interp3(X, Y, Z, V, XI, YI, ZI);
VI = interp3(V, XI, YI, ZI);
VI = interp3(V, ntimes);
VI = interp3(X, Y, Z, V, XI, YI, ZI, method);
VI = interp3(X, Y, Z, V, XI, YI, ZI, method,
             extrapval).
```

10.4. Многомерная интерполяция данных

Подготовка значений независимых переменных для дальнейшего интерполирования функции N независимых переменных $v = f(x_1, x_2, ..., x_n)$ в N – мерном параллелепипеде выполняется при помощи следующей функции:

```
[X1, X2, X3…] = ndgrid(x1, x2, x3, …)
```

где x1, x2, x3 – векторы.

Функция

```
[X1, X2, X3…] = ndgrid(x)
```

Подразумевает использование вектора x в качестве векторов x2, x3, …

Для интерполяции в N – мерном параллелепипеде используется функция:

```
VI = interpn(X1, X2, X3, …, V, Y1, Y2, Y3, …)
```

Узлы интерполирования задаются в массивах X1, X2, X3, …, значения функции – в массиве V, а значения независимых переменных, для которых необхдимо выполнить интерполирование – в массивах Y1, Y2, Y3, … При этом предполагается, что значения независимых переменных X1, X2, X3, … изменяются монотонно.

Варианты использования функции intern совпадают с таковыми для функций двух и трехмерной интерполяции:

```
VI = interpn(V, Y1, Y2, Y3, …)
VI = interpn(V, ntimes)
VI = interpn(X1, X2, X3, …, V, Y1, Y2, Y3, …,
             method)
VI = interpn(X1, X2, X3, …, V, Y1, Y2, Y3, …,
             method, extrapval)
```

11. Численное интегрирование в MATLAB

11.1. Численное интегрирование функции одной переменной

Функция

```
q = quad(fun, a, b)
```

Реализует вычисление определенного интеграла $q = \int_a^b f(x)dx$, $a,b \in \mathbb{R}$, $f : \mathbb{R} \to \mathbb{R}$ на основе адаптивной квадратурной формулы Симпсона. a, b - . конечные скаляры, fun – дескриптор функции f, возвращающий подынтегральное выражение. Возможен случай, когда $a,b \in \mathbb{C}$, $f : \mathbb{C} \to \mathbb{C}$. Тогда интегрирование выполняется по отрезку комплексной плоскости, соединяющему точки a и b.

Функция

```
q = quad(fun, a, b, tol)
```

использует значение tol в качестве абсолютной погрешности. Значение по умолчанию 10^{-6}.

Функция

```
q = quad(fun, a, b, tol, trace)
```

с ненулевым параметром trace выполняет трассировку рекурсивно выполняющейся процедуры численного интегрирования.

Функция

```
[q, fcnt] = quad(…)
```

Возвращает в качестве второго параметра число обращений к функции, вычиляющей подынтегральное выражение.

Для вычисления интегралов от достаточно гладких бесконечно интегрируемых функций на конечных отрезках целесообразно использовать квадратурные формулы более высокого продярка точности, чем квадратурная формула Симпсона. Следующие функции аналогичны по смыслу функции quad, однако используют при численном интегрировании более точные квадратурные формулы, соответствующие так называемому адаптивному методу Лобатто.

```
q = quadl(fun, a, b);
q = quadl(fun, a, b, tol);
q = quadl(fun, a, b, tol, trace);
q = quadl(fun, a, b, tol);
[q, fcnt] = quadl(…);
```

Следующий набор функций применяется в том случае, когда подынтегральное выражение возвращает вместо скалярного значения возвращает матрицу или вектор:

```
Q = quadv(fun, a, b);
Q = quadv(fun, a, b, tol);
Q = quadv(fun, a, b, tol, trace);
Q = quadv(fun, a, b, tol);
[Q, fcnt] = quadv(…);
```

Функция

```
q = quadgk(fun, a, b);
[q, errbnd] = quadgk(fun, a, b, tol);
```

Использует адаптивные высокоточные формулы Гаусса – Конрада для вычисления интегралов. Во втором случае дополнительный параметр errbnd возвращает оценку абсолютной погрешности интегрирования.

Заметим, что функция quadgk применима также для вычисления сходящихся несобственных интегралов на интервале $[a,\infty)$ либо $(-\infty,b)$. Ее не рекомендуется использовать для быстро осциллирующих функций, однако можно использовать, если при $x \to \pm\infty$ функция f(x) убывает достаточно быстро.

Заметим также, что функцию quadgk можно применять и в случае, если граничные точки a и b являются особыми точками (например типа $(x-a)^p$ при $p \geq -1/2$, ln(x-a) и т.д.). Если же особые точки находятся внутри отрезка интегрирования [a, b], то исходный интеграл потребуется разбить на сумму интегралов так, чтобы особенности соответствовали граничным точкам отрезков интегрирования.

Функция

```
[q, errbnd] = quadgk(fun, a, b, param1, val1,
                     param2, val2, …)
```

Выполняет численное интегрирование с использованием специальных параметров настройки алгоритма численного интегрирования.

Параметр	Действие
'AbsTol'	Абсолютная точность численного интегрирования. 10^{-10} по умолчанию
'RelTol'	Относительная точность численного интегрирования. 10^{-6} по умолчанию
'WayPoints'	Вектор из точек комплексной плоскости, определяющих путь интегрирования от точки a до точки b. Если a, b и другие точки пути интегрирования все являются действительными, то они используются в отсортированном порядке.
'MaxIntervalCount'	Макс. позволенное число разбиений отрезка интегрирования (650 по умолчанию)

11.2. Численное интегрирование функции двух переменных

Вычисление интеграла $q = \int_{x_{\min}}^{x_{\max}} dx \int_{y_{\min}}^{y_{\max}} dy f(x,y)$ в прямоугольнике $x_{\min} \leq x \leq x_{\max}$, $y_{\min} \leq y \leq y_{\max}$ реализуется следующими функциями:

```
q=dblquad(fun,xmin,xmax,emin,ymax);
```

Функция

```
q=dblquad(fun,xmin,xmax,emin,ymax,tol);
```

использует значение tol в качестве абсолютной погрешности. Значение по умолчанию 10^{-6}.

Для выполнения двукратного интегрирования данная функция вызывает функцию quad, т.е. используется численное интегрирование на основе квадратурной формулы Симпсона. Если требуется использовать квадратурные формулы Лобитто, либо другие квадратурные формулы повышенной точности, то используется функция:

```
q=dblquad(fun,xmin,xmax,emin,ymax,tol,method);
```

где в качестве параметра method используется дискриптор функции quadl, либо дескриптор любой другой функции, выполняющей численное интегрирование на одномерном отрезке пользовательской функции, семантика вызова которой совпадает с семантикой вызова функций quad и quadl.

Двумерное интегрирование $q = \int_a^b dx \int_{c(x)}^{d(x)} dy f(x,y)$ в двумерной плоской области $a \le x \le b$, $c(x) \le y \le d(x)$ обеспечивается функцией:

```
q = quad2d(fun, a, b, c, d)
```

Здесь fun – дескриптор функции. Входные параметры c и d также могут быть дескрипторами функций. Все входные параметры должны быть векторизированы. Т.е. функция z = fun(X, Y) должна принимать двумерные массивы – матрицы, содержащие значения x и y, одних и тех же размерностей, и возвращать матрицу Z соответствующей размерности. Функции y_min = c(X) и y_max = d(X) должны в качесве входного параметра принимать матрицу и возвращать также матрицу соответствующей размерности.

Также поддерживаются варианты вызовов:

```
[q, errbnd] = quad2d(fun, a, b, c, d)
```

Где дополнительный выходной параметр errbnd представляет собой оценку абсолютной погрешности интегрирования.

```
q = quad2d(fun, a, b, c, d, param1, val1,
                param2, val2, …)
```

выполняет численное интегрирование в соответствии с переданными настройками алгоритма численного интегрирования. Возможны следующие значения:

Параметр	Действие
'AbsTol'	Абсолютная точность числ. интегрир.
'RelTol'	Относит. точн. числ. интегрирования
'MaxFunEvals'	Макс. допустимое число обращений к ф-ии **fun**.
'Singular'	**true**, если $f(x)$ сингулярна вблизи границ

Трехмерное интегрирование $q = \int_{x_{\min}}^{x_{\max}} dx \int_{y_{\min}}^{y_{\max}} dy \int_{z_{\min}}^{z_{\max}} dx f(x,y,z)$ в параллелипеде $x_{\min} \le x \le x_{\max}$, $y_{\min} \le y \le y_{\max}$, $z_{\min} \le z \le z_{\max}$ выполняется посредством следующих вызовов:

```
q = triplequad(fun, xmin, xmax, ymin, ymax,
                    zmin, zmax);
q = triplequad(fun, xmin, xmax, ymin, ymax,
                    zmin, zmax, tol);
q = triplequad(fun, xmin, xmax, ymin, ymax,
                    zmin, zmax, tol, method);
```

Параметры данных функций аналогичны параметрам фукнции dblquad. fun – дескриптор функции, вычисляющей подынтегральное выражение, tol – требуемая абсолютная точность численного интегрирования, method - дескриптор функции численного интегрирования в одномерной области (на отрезке). Вызов функции triplequad аналогичен вызову функций quad и dblquad.

12. Численное дифференцирование

Пусть $\mathbf{x} = (x_1, x_2, ..., x_N)^T$ - N-мерный вектор.

Функция

```
Y = diff(X)
```

возвращает (N-1)-мерный вектор правых конечных разностей $Y = \Delta\mathbf{x} = (x_2 - x_1, x_3 - x_2, ..., x_N - x_{N-1})^T$.

Если X – матрица, то конечные разности вычисляются в каждом ее

столбце. Т.е. результирующая матрица будет содержать разности строк исходной матрицы, и размерность результирующей матрицы будет на одну строку меньше чем исходная.

Конечные разности высших порядков находятся последовательным выполнением операции нахождения конечных разностей 1-го порядка. Так, для исходного вектора x правые конечные разности 2-го порядка образуют вектор размерности (N-2) и вычисляются следующим образом:

$$\Delta^2\mathbf{x} = \Delta(\Delta\mathbf{x}) = ((x_3 - x_2) - (x_2 - x_1),...,(x_N - x_{N-1}) - (x_{N-1} - x_{N-2}))^T =$$
$$= (x_3 + x_1 - 2x_2,...,x_N + x_{N-2} - 2x_{N-1})^T$$

Функция

```
Y = diff(X, n)
```

Возвращает вектор правых конечных разностей порядка n.

Функция

```
Y = diff(X, n, dim)
```

Вычисляет правые конечные разности порядка n в векторах, которые задаются индексом многомерного массива X с номером dim.

Приближенное вычисление градиента функии одной или нескольких переменных обеспечивается следующими функциями.

Пусть вектор-строка F содержит значения функции f(x), заданные в равноотстоящих точках с шагом 1. Приближенное значение df/dx в соответствующих точках возвращается при помощи вызова:

```
FX = gradient(F)
```

Заметим, что размерности векторов FX и F совпадают, поскольку в начальной точке градиент оценивается правой конечной разностью; в конечной точке градиент вычисляется посредством левой конечной разности; во внутренних точках посредством центральных разностей. Вычисленный таким образом градиент имеет второй порядок точности.

$$\mathbf{f}_x = \left(f_2 - f_1, \frac{f_3 - f_1}{2},...,\frac{f_N - f_{N-2}}{2}, f_N - f_{N-1}\right)$$

Пусть теперь матрица F содержит значения функции двух переменных f(x, y), заданное в равноотстоящих точках с шагом 1. При этом предполагается, что изменение 1-ой переменной (x) соответствует движению по горизонтали по строкам матрицы F, а изменение 2-ой переменной (y) соответствует

движению по вертикали вниз по столбцам матрицы F. Приближенные значения компонент градиента $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)^T$ вычисляются следующей функцией:

```
[FX, FY] = gradient(F)
```

Пусть далее многомерный массив F содержит значения функции нескольких переменных f(x, y, z, …), заданные в равноотстоящих точках с шагом 1. Предполагается, что при значении номера $n \geq 3$ изменению n-ой независимой переменной в списке аргументов функции f(x, y, z, …) соответствует изменение n-го индекса многомерного массива F. Приближенные значения компонент градиента $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, ...\right)^T$

```
[FX, FY, FZ, …] = gradient(F)
```

Вызов функции

```
[ …. ] = gradient(F, h)
```

Задает в качестве шага изменения всех независимых переменных величину h.
Вызов функции

```
[ …. ] = gradient(F, h1, h2, …)
```

Позволяет задать шаг изменения для каждой независимой переменной.

12.1. Приближенное вычисление дифференциального оператора Лапласа

Пусть многомерный массив U содержит значения функции нескольких переменных $f(x_1, x_2, ..., x_n)$, заданные в равноотстоящих точках с шагом 1. Аналогично ранее рассмотренному, значения переменной x_k соответствует изменению k-го индекса массива U.
Функция

```
L = del2(U)
```

Возвращает набор приближенных значений величины

$$L = \frac{\nabla^2 u}{2n} = \frac{1}{2n}\left(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2} + ... + \frac{\partial^2 u}{\partial x_n^2}\right)$$

Вызов функции

```
L = del2(U, h)
```

Позволяет указать в качестве шага изменения всех независимых переменных величину h.
Вызов функции

```
L = del2(U, h1, h2, h3, …)
```

Позволяет задать шаг изменения для каждой независимой переменной.

13. Дискретное преобразование Фурье в MATLAB

При выполнении дискретного преобразования Фурье ориентированные на матричные вычисления системы MATLAB, SCILAB и Octave, как правило, используют функции свободно распространяемого пакета FFTW3. Заметим, что библиотека для поддержки высокопроизводительных вычислений на процессорах x86 и x86-64 Intel Math Kernel Library (IMKL) поддерживает интерфейсы для функций дискретного преобразования Фурье, аналогичные интерфейсам пакета FFTW3, и, следовательно, может быть использована (и часто используется) совместно с MATLAB.

13.1. Одномерное дискретное преобразование Фурье

Дискретное преобразование Фурье является аналогом рядов Фурье T-периодических функций времени t

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{2ik\pi t/T},\ c_k = \frac{1}{T}\int_0^T x(t)e^{-2ik\pi t/T}dt, t \in \mathbb{R},\ x : \mathbb{R} \to \mathbb{C},\ i = \sqrt{-1}$$

и получающихся из рядов Фурье при $T \to \infty$ прямого и обратного преобразования Фурье абсолютно интегрируемых на числовой оси функций:

$$\hat{x}(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt,\ x(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} \hat{x}(\omega)e^{i\omega t}d\omega$$

Пусть $\mathbf{x} = (x_0, x_1, ..., x_{N-1})^T \in \mathbb{C}^N$ - некоторый N-мерный комплексный вектор. Прямое и обратное преобразование Фурье определены следующим образом:

$$y_k = \sum_{j=0}^{N-1} x_j \omega_N^{-kj},\ x_k = \frac{1}{N} \sum_{j=0}^{N-1} y_j \omega_N^{kj},\ k = 0,1,...,N-1;\ \omega_N = e^{2\pi i/N}$$

Очевидно, величина ω_N представляет собой один из корней степени N из единицы.

В смысле нумерации, дискретное преобразование Фурье несколько отличается от рядов Фурье. Так, величина y_0 аналогична Фурье - коэффициенту c_0, y_1 аналогична c_1, y_{N-1} аналогична c_{-1}, y_2 аналогична c_2, y_{N-2} аналогична c_{-2} и так далее.

В общем случае, выполнение дискретного преобразования Фурье требует порядка N^2 действий. Однако, если $N = 2^M$, где M – целое число, то число действий резко сокращается до $N \log_2 N$, и говорят, что имеет место быстрое преобразование Фурье.

Функция

```
Y = fft(X)
```

Возвращает результат дискретного преобразования Фурье вектора X. Если X – матрица, то дискретное преобразование Фурье выполняется для каждого столбца. Если X – многомерный массив, то преобразование Фурье выполняется по измерению, которому соответствует первый индекс, диапазон изменения которого превышает 1.

Вызов функции

```
Y = fft(X, n)
```

Выполняет n-точечное дискретное преобразование Фурье. Если длина вектора X превосходит n, то лишние элементы не участвуют в преобразовании Фурье. Если длина вектора X меньше, чем n, то недостающие элементы заменяются нулями.

Вызовы функции

```
Y = fft(X, [], dim)
Y = fft(X, n, dim)
```

Выполняют дискретное преобразование Фурье по измерению, которое определяется номером dim.

Обратное дискретное преобразование Фурье реализуется функциями:

```
Y = ifft(X)
Y = ifft(X, n)
Y = ifft(X, n, dim)
Y = ifft(X, [], dim)
```

Заметим, что если исходный вектор $\mathbf{x} = (x_0, x_1, ..., x_{N-1})^T$ является чисто действительным, то компоненты преобразованного вектора y обладают свойством: $y_{N-k} = \overline{y}_k$, $k = 1,2,3,...$ Для того, чтобы явно учесть это обстоятельство, при выполнении обратного преобразования Фурье используется вызов:

```
Y = ifft(…, 'symmetric')
```

Пусть `Y = fft(X)`, т.е. массив Y является результатом прямого дискретного преобразования Фурье.
Функция:

```
YY = fftshift(Y)
```

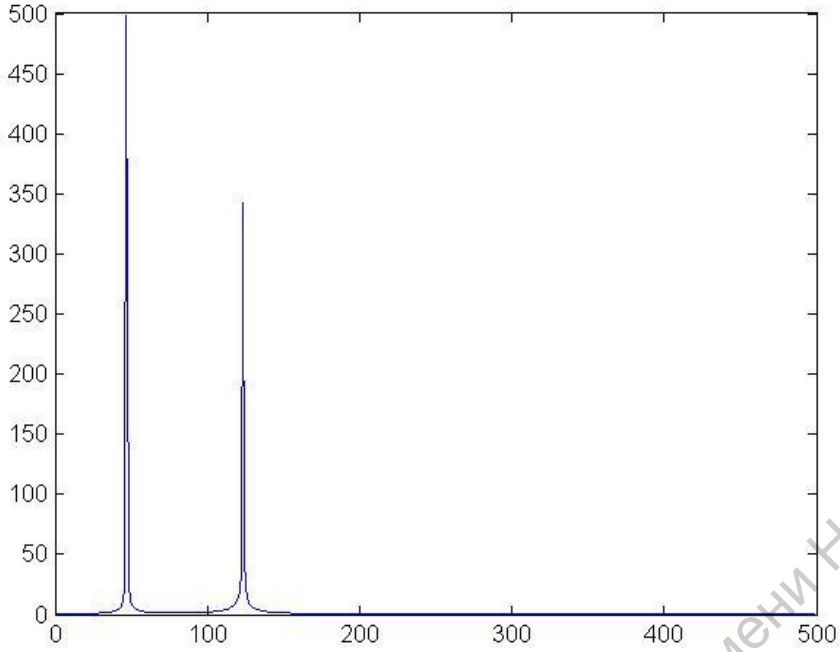
Переупорядочивает элементы массива Y таким образом, что центральному элементу массива Y соответствует нулевое значение частоты.
Обратное переупорядочивание осуществляется вызовом:

```
Y = ifftshift(YY)
```

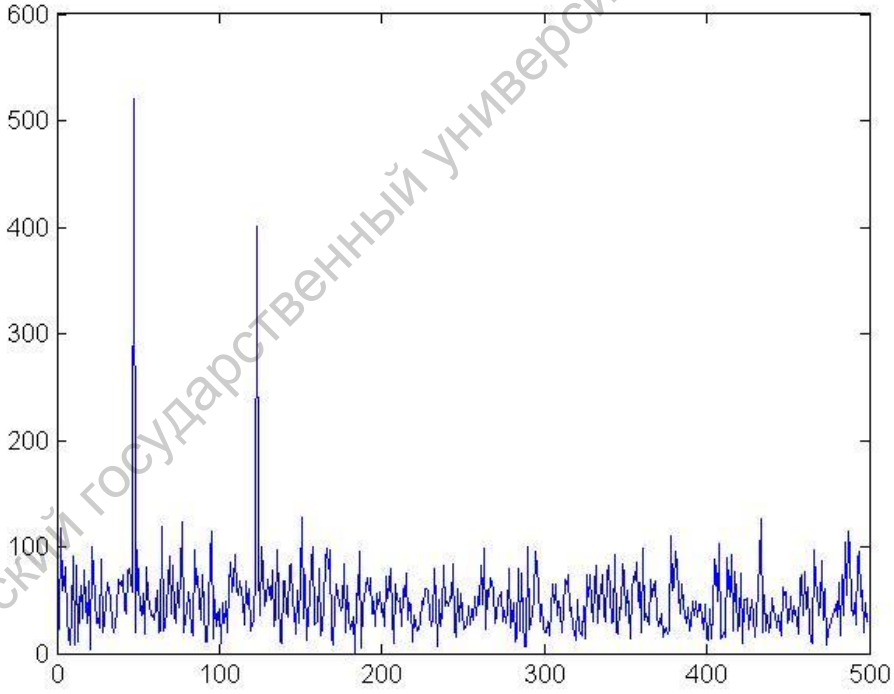
Пример. Анализ зашумленного сигнала

```
%Анализ частотного спектра зашумленного сигнала
W=1000;          % Характерная частота в Гц
dt=1/W;          % Характерный период между временными отсчетами
N=500;          % 2N+1 - Число точек в сигнале
dw=W/(2*N);      % Характерный шаг частоты
t=(0:2*N)*dt; %Набор отсчетов времени
w1=47; w2=123; %Частоты исходного сигнала в Гц
A1=1; A2=0.7; %Амплитуды исходного сигнала
x=A1*sin(2*pi*w1*t)+A2*cos(2*pi*w2*t); %Исходный сигнал
y=x+(A1+A2)*randn(size(t)); %Зашумленный сигнал
w=dw*(0:N-1); %Вектор из отсчетов частот
X=fft(x); %Преобразование Фурье исходного сигнала
figure; plot(w,abs(X(1:N)));
Y=fft(y); %Преобразование Фурье зашумленного сигнала
```

```
figure; plot(w,abs(Y(1:N)));
```



Спектр исходного сигнала



Спектр зашумленного сигнала

13.2.Двумерное дискретное преобразование Фурье

Двумерное дискретное преобразование Фурье является аналогов двойных

рядов Фурье функий двух переменных t_1 и t_2 с периодом T_1 и T_2.

$$x(t_1,t_2)=\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}c_{k_1k_2}e^{2\pi\left(\frac{k_1t_1}{T_1}+\frac{k_2t_2}{T_2}\right)},$$

$$c_{k_1k_2}=\frac{1}{T_1T_2}\int_0^{T_1}\int_0^{T_2}x(t_1,t_2)e^{-2\pi i\left(\frac{k_1t_1}{T_1}+\frac{k_2t_2}{T_2}\right)}dt_1dt_2$$

И получающегося из них при $T_1\to\infty$ и $T_2\to\infty$ двукратного прямого и обратного преобразования Фурье абсолютно интегрируемых в числовой плоскости (t_1,t_2) функций:

$$\hat{x}(\omega_1,\omega_2)=\frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}x(t_1,t_2)e^{-i\,\omega_1t_1+\omega_2t_2}\,dt_2dt_1$$

$$x(t_1,t_2)=\frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\hat{x}(\omega_1,\omega_2)e^{i\,\omega_1t_1+\omega_2t_2}\,dt_2dt_1$$

Пусть

$X=\{x_{k_1k_2}\}\in\mathbb{C}^{N_1\times N_2}$ - некоторая матрица. Прямое и обратное дискретное преобразование Фурье (двумерное) суть:

$$y_{k_1k_2}=\sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1}x_{j_1j_2}\omega_{N_1}^{-k_1j_1}\omega_{N_2}^{-k_2j_2},\ x_{k_1k_2}=\frac{1}{N_1N_2}\sum_{j_1=0}^{N_1-1}\sum_{j_2=0}^{N_2-1}y_{j_1j_2}\omega_{N_1}^{k_1j_1}\omega_{N_2}^{k_2j_2}$$

$$\omega_{N_1}=e^{2\pi i/N_1},\ \omega_{N_2}=e^{2\pi i/N_2},k_1=0,1,...,N_1-1;\ k_2=0,1,...,N_2-1$$

Функция

```
Y = fft2(X)
```

Выполняет двумерное дискретное преобразование Фурье для матрицы X.

Функция

```
Y = fft2(X, m, n)
```

Выполняет двумерное дискретное преобразование Фурье для фрагмента матрицы размерности m x n. Если размеры матрицы X меньше, чем m x n,

то она дополняется нулями.
Обратное двумерное дискретное преобразование Фурье некоторой действительной матрицы реализуется функиями:

```
Y = ifft2(X)
Y = ifft2(X, m, n)
Y = ifft2(…, 'symmetric')
```

Последний вызов целесообразен, если входная матрица представляет собой результат двумерного дискретного преобразования Фурье некоторой действительной матрицы.
Если `Y = ifft2(X)` – результат прямого дискретного двумерного преобразования Фурье, то вызов функции:

```
YY = fftshift(Y)
```

Позволяет сопоставить нулевую частоту центральному элементу массива YY. А вызов:

```
YY = fftshift(Y, dim)
```

Позволяет выполнить подобное переупорядочивание по измерению с номером dim.
Обратное переупорядочивание реализуется вызовами:

```
Y = ifftshift(YY)
Y = ifftshift(YY, dim)
```

13.3. Многомерное дискретное преобразование Фурье

Пусть $X = \{x_{k_1 k_2 \ldots k_n}\} \in \mathbb{C}^{N_1 \times N_2 \times \ldots \times N_n}$ - n мерный массив. Дискретное прямое и обратное n-кратное преобразование Фурье имеет вид:

$$y_{k_1 k_2 \ldots k_n} = \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} \ldots \sum_{j_n=0}^{N_n-1} x_{j_1 j_2 \ldots j_n} \omega_{N_1}^{-k_1 j_1} \omega_{N_2}^{-k_2 j_2} \ldots \omega_{N_n}^{-k_n j_n}$$

$$x_{k_1 k_2 \ldots k_n} = \frac{1}{N_1 N_2 \ldots N_n} \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} \ldots \sum_{j_n=0}^{N_n-1} y_{j_1 j_2 \ldots j_n} \omega_{N_1}^{k_1 j_1} \omega_{N_2}^{k_2 j_2} \ldots \omega_{N_n}^{k_n j_n}$$

$$\omega_{N_1} = e^{2\pi i / N_1},\ \omega_{N_2} = e^{2\pi i / N_2}, \ldots,\ \omega_{N_n} = e^{2\pi i / N_n}, k_1 = 0,1,\ldots,N_1 - 1;\ k_2 = 0,1,\ldots,N_2 - 1; \ldots$$

Вызов функции:

```
Y = fftn(X)
```

Реализует n-кратное дискретное прямое преобразование Фурье для n-мерного массива. Выходной массив Y имеет ту же размерность, что и X.
Вызов:

```
Y = fftn(X, size)
```

Реализует n-кратное дискретное прямое преобразование Фурье для фрагмента массива X, размерность которого задается компонентами вектора size. Если массив меньше, чем размерность, указанная в векторе size, то он дополняется нулями.
Обратное дискретное n-кратное преобразование Фурье реализуется вызовами:

```
Y = ifftn(X)
Y = ifftn(X, size)
Y = ifftn(X, 'symmetric')
```

Последний вызов целесообразен, если входной массив представляет собой результат n-кратного дискретного преобразования Фурье некоторого n-мерного действительного массива.
Если массив Y – результат n-кратного дискретного преобразования Фурье, то сопоставление нулевой частоты центральному элементу массива Y осуществляется вызовом:

```
YY = fftshift(Y)
YY = fftshift(Y, dim)
```

Последний вызов осуществляет данное преобразование по измерению с номером dim.
Обратное переупорядочивание реализуется вызовами:

```
Y = ifftshift(YY)
Y = ifftshift(YY, dim)
```

14. Стандартные функции MATLAB для решения задач оптимизации

Число стандартных функций MATLAB для решения задач оптимизации относительно невелико, однако это органичение компенсируется одним из пакетов расширения для MATLAB – пакетом MATLAB Optimization Toolbox.

14.1. Поиск локального минимума функции

Поиск локального минимума функции $y = f(x),\ f : \mathbb{R} \to \mathbb{R}$, при $x \in [a,b]$ осуществляется вызовом функции:

```
x = fminbnd(fun, a, b)
```

здесь fun – дескриптор функции, вычисляющей значения f(x), a и b - границы интервала, где требуется найти минимум.

Вызов функции

```
x = fminbnd(fun, a, b, options)
```

позволяет задать дополнительные параметры в структуре ‘options’ со следующими полями:

`Display` – уровень выдачи диагностических сообщений.
- `'off'` – не выдавать сообщения
- `'iter'` – выводить сообщения на каждой итерации
- `'final'` – выводить сообщения лишь на финальной стадии
- `'notify'` – (по умолчанию) – выводить лишь сообщения об отсутствии сходимости

`FunValueCheck` – проверка корректности значений, возвращенных целевой функцией
- `'on'` – сообщение об ошибке, если целевая функция возвращает нечисловое или комплексное значение
- `'off'` – подавление сообщения об ошибке

`MaxFunEvals` – максимально допустимое число обращений к целевой функции

`MaxIter` – максимально допустимое число итераций

`OutputFcn` – дескриптор функции, которая вызывается на каждой итерации и служит для анализа промежуточных данных

`PlotFcn` – дескриптор функции, которая вызывается на каждой итерации и служит для графического отображения промежуточных

результатов.

`TolX` – требуемая точность вычисления точки, соответствующей минимуму целевой функции.

Вызов функции:

```
[x, fval] = fminbdn(…)
```

Возвращает сверх того само значение целевой функции в точке минимума.

Вызов функции:

```
[x, fval, exitflag] = fminbnd(…)
```

Возвращает дополнительно значение exitflag, которое кодирует условие выхода из функции:

1 – достигнута сходимость с требуемой точностью к искомому решению;

0 – превышено максимально допустимое число обращений к целевой функции либо максимально допустимое число итераций;

-1 – алгоритм был завершен функцией, дескриптор которой задан в параметре OutputFcn;

-2 – границы интервала заданы ошибочно (a > b).

Вызов функции:

```
[x, fval, exitflag, output] = fminbnd(…)
```

Дополнительно возвращает структуру output, поля которой содержат информацию о ходе оптимизации:

`Algorithm` – используемый алгоритм

`FuncCount` – число обращений к целевой функции

`Iterations` – число итераций;

`Message` – диагностическое сообщение при выходе из функции.

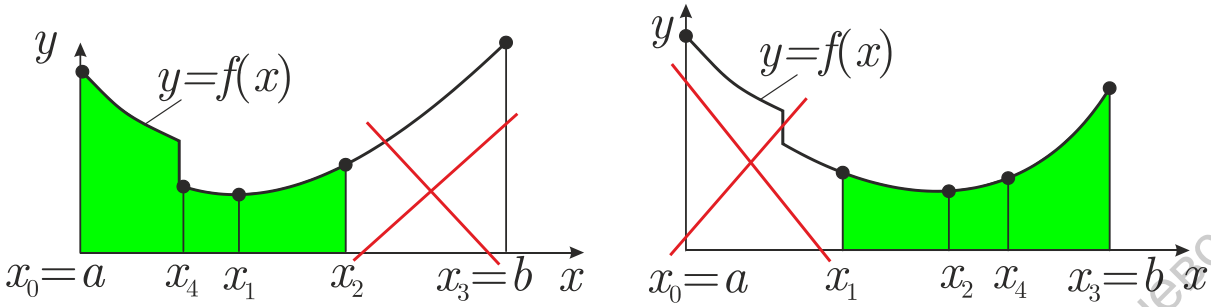
Для ускорения сходимости используется алгоритм, основанный на методе золотого сечения и на методе парабол.

14.2. Метод «золотого сечения»

Поиск локального минимума $y = f(x)$, $f : \mathbb{R} \to \mathbb{R}$, при $x \in [a,b]$. Применяется для негладких функций

$x_0 = a$, $x_3 = b$, $x_1 = a + \xi(b-a)$, $x_2 = a - \xi(b-a)$

$$\frac{b-x_1}{b-a}=\frac{b-x}{b-x_1} \Leftrightarrow \frac{a-x_2}{a-b}=\frac{a-x_1}{a-x_2} \Rightarrow \xi=\frac{2}{3+\sqrt{5}}\approx 0.38$$



$f(x_0)\to\min$ или $f(x_1)\to\min$ $\quad$ $f(x_2)\to\min$ или $f(x_3)\to\min$

$$x_4=x_0+\xi(x_2-x_0) \qquad x_4=x_3-\xi(x_3-x_1) \qquad (1)$$

Далее x_j перенумеруются так, что $x_0<x_1<x_2<x_3$, и повторяется (1)

14.3. Метод парабол

Как правило, в малой окрестности локального минимума $y=f(x)$ дважды дифференцируема. Разделенные разности:

$$f[t_1,t_2]=\frac{f(t_1)-f(t_2)}{t_1-t_2},\quad f[t_1,t_2,t_3]=\frac{f[t_1,t_2]-f[t_2,t_3]}{t_1-t_3}$$

Пусть в окрестности локальн. минимума известны $f(x_1)$, $f(x_2)$, $f(x_3)$

$$f(x)\approx f(x_3)+(x-x_3)f[x_3,x_2]+(x-x_3)(x-x_2)f[x_3,x_2,x_1]$$

$$f'(x_{\min})\approx f[x_3,x_2]+(2x_{\min}-x_3-x_2)f[x_3,x_2,x_1]=0$$

$$x_{\min}\approx\frac{1}{2}\left(x_3+x_2-\frac{f[x_3,x_2]}{f[x_3,x_2,x_1]}\right)$$

Т.е

$$x_{k+3}\approx\frac{1}{2}\left(x_{k+2}+x_{k+1}-\frac{f[x_{k+2},x_{k+1}]}{f[x_{k+2},x_{k+1},x_k]}\right),\quad k=1,2,3,\ldots$$

14.4. Поиск минимума действительной функции n действительных переменных

Поиск локального минимума функции $f:\mathbb{R}^n\to\mathbb{R}$ в неограниченной области реализуется стандартной функцией `fminsearch`. Данная функция использует так называемый алгоритм метода симплексов, метода

потоков или метода Нелдера – Мида, который не требует выражения градиента целевой функции. Следовательно, целевая функция может быть негладкой.
Вызов функции:

```
x = fminsearch(fun, x0)
```

используя начальное приближение x0, возвращает значение x, которое соответствует координатам точки локального минимума функции f(x). Здесь параметр x0 может быть скаляром, вектором или матрицей, а параметр fun является дескриптором функции, соответствующей целевой функции f(x).

Вызов функции:

```
x = fminsearch(fun, x0, options)
```

позволяет задать дополнительные параметры оптимизации в полях структуры options, аналогичной соответствующей структуре options для функции fminbnd. Дополнительно поле `TolFun` задает требуемую точность нахождения локального минимума функции f(x).

Вызов функции:

```
[x, fval] = fminsearch(…)
```

Возвращает дополнительно значение целевой функции в точке минимума.

Вызов функции:

```
[x, fval, exitflag] = fminsearch(…)
```

Дополнительно возвращает код выхода с возможными значениями:

1 – достигнута сходимость с требуемой точностью;
0 – превышено максимально допустимое число обращений к целевой функции либо максимально допустимое число итераций;
-1 – работа алгоритма прервана функцией, дескриптор которой указан в поле `OutputFcn` входного параметра options.

Вызов функции:

```
[x, fval, exitflag, output] = fminsearch(…)
```

Дополнительно возвращает структуру output, поля которой аналогичны

полям структуры output для функции fminbnd.

14.5. Безградиентный метод Нелдера-Мида (политопов)

Применяется для поиска локального минимума функции $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$ в неограниченной области без ограничения на степень гладкости f. По своей природе, метод Нелдера-Мида применим в случае, когда целевая функция зависит от относительно небольшого числа аргументов (порядка десятка, или в крайнем случае нескольких десятков). Однако данный метод становится неприменим, если число аргументов превышает сотню.

Метод Нелдера-Мида оперирует с n-мерными симплексами, вершинами которых служат точки (n-мерные векторы) $\mathbf{x}_1$, $\mathbf{x}_2, \ldots, \mathbf{x}_n$, $\mathbf{x}_{n+1}$.

На каждой итерации метода Нелдера-Мида выполняются следующие действия:

1. Точки $\mathbf{x}_j$, $j = 1, 2, ..., n, n+1$ упорядочиваются по возрастанию значений $f(\mathbf{x}_j)$.

2. Либо выполняется перестройка симплекса.

3. Либо выполняется сжатие симплекса.

Этап перестройки симплекса

1) Центр масс всех точек, кроме наихудшей: $\mathbf{m} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$, $i = 1, 2, ..., n$

2) Вычисляется отраженная точка
$\mathbf{r} = 2\mathbf{m} - \mathbf{x}_{n+1}$ и значение $f(\mathbf{r})$

3) Если $f(\mathbf{x}_1) \le f(\mathbf{r}) \le f(\mathbf{x}_n)$, перестройка симпл. сводится к замене $\mathbf{x}_{n+1}$ на $\mathbf{r}$

4) $f(\mathbf{r}) < f(\mathbf{x}_1)$: вычисляется продолженная точка $\mathbf{s} = \mathbf{m} + 2(\mathbf{m} - \mathbf{x}_{n+1})$ и знач. $f(\mathbf{s})$

a) $f(\mathbf{s}) < f(\mathbf{r})$: перестройка симпл. сводится к замене $\mathbf{x}_{n+1}$ на $\mathbf{s}$

b) в противн. случае: перестр. симпл. сводится к замене $\mathbf{x}_{n+1}$ на $\mathbf{r}$

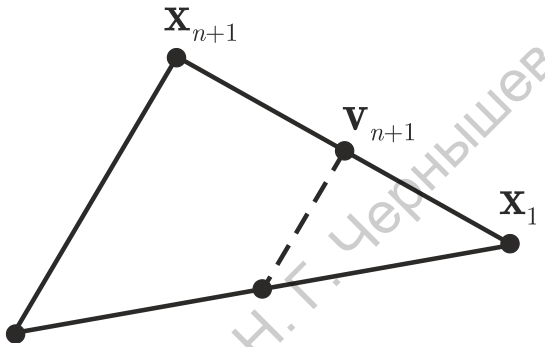
5) $f(\mathbf{r}) \ge f(\mathbf{x}_n)$: усредн. между т. $\mathbf{m}$ и наилучшей из точек $\mathbf{x}_{n+1}$ и $\mathbf{r}$

a) $f(\mathbf{r}) < f(\mathbf{x}_{n+1})$: вычисляется т. $\mathbf{c}_1 = \mathbf{m} + (\mathbf{r} - \mathbf{m}) / 2$ и $f(\mathbf{c}_1)$

- $f(\mathbf{c}_1) < f(\mathbf{r})$: перестройка симплекса сводится к замене вершины $\mathbf{x}_{n+1}$ на точку $\mathbf{c}_1$
- в противном случае: выполняется сжатие симплекса.

b) $f(\mathbf{r}) \geq f(\mathbf{x}_{n+1})$: вычисляется т. $\mathbf{c}_2 = \mathbf{m} + (\mathbf{x}_{n+1} - \mathbf{m})/2$ и $f(\mathbf{c}_2)$

- $f(\mathbf{c}_2) < f(\mathbf{x}_{n+1})$: перестройка симплекса сводится к замене вершины $\mathbf{x}_{n+1}$ на точку $\mathbf{c}_2$
- в противном случае: сжатие симплекса



Этап сжатия симплекса:

$\mathbf{v}_j = (\mathbf{x}_j + \mathbf{x}_1)/2,\ j = 2,3,\ldots,n,n+1$ и далее рассматривается симплекс с вершинами $\mathbf{x}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$,…,$\mathbf{v}_n$, $\mathbf{v}_{n+1}$.

14.6. Поиск корня непрерывной монотонной функции

Поиск корня непрерывной и монотонной в окрестности корня функции одной переменной $f : \mathbb{R} \to \mathbb{R}$ обеспечивается функцией:

```
x = fzero(fun, x0)
```

Данный вызов пытается найти корень функции f(x), которой соответствует дескриптор функции fun, скаляр x0 используется в качестве начального приближения.

Вызов функции:

```
x = fzero(fun, x0, options)
```

позволяет задать параметры алгоритма оптимизации в структуре options, поля которой аналогичны полям соответствующей структуры при вызове функции fminbnd.

Вызов

```
[x, fval] = fzero(…)
```

Дополнительно возвращает значение f(x), соответствующее

приближенно найденному корню x.

Вызов функции

```
[x, fval, exitflag] = fzero(…)
```

Возвращает также код выхода exitflag со следующими возможными значениями:

1 – достигнута сходимость к искомому корню x;
-1 – алгоритм завершен функцией, дескриптор которой указан в поле OutputFun входного параметра options;
-3 – В процессе поиска интервала, на котором функция f(x) меняет знак, были обнаружены бесконечность, либо нечисловое значение функции f(x);
-4 – В процессе поиска интервала, на котором функции f(x) меняет знак, были обнаружены комплексные значения функции f(x);
-5 – Алгоритм сходится к бесконечно удаленной точке;
-6 – Не удалось обнаружить интервал, на котором функция f(x) меняет знак.

Вызов функции:

```
[x, fval, exitflag, output] = fzero(…)
```

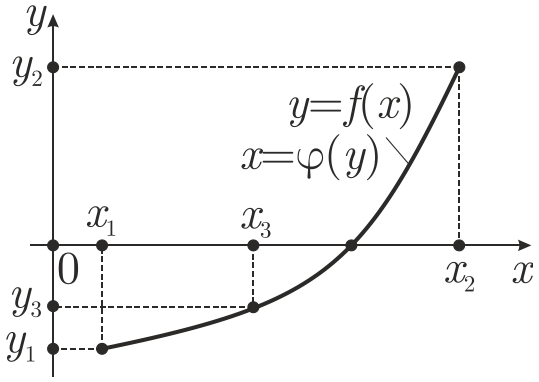
Дополнительно возвращает структуру, содержащую информацию о ходе процесса оптимизации со следующими полями:

`Algorithm` – используемый алгоритм;
`FuncCount` – число обращений к функции f(x);
`Interval Iterations` – число итераций для нахождения интервала, где f(x) меняет знак;
`Iterations` – число итераций при нахождении корня;
`Message` – диагностическое сообщение.

Пояснение
Для ускорения сходимости, функция fzero использует алгоритмы методов деления отрезка пополам, секущих и обратного квадратичного интерполирования.
Если функция y = f(x) монотонна в некоторой окрестности искомого корня, то обратная функция $x = \varphi(y)$ существует и однозначна. В таком случае, если имеются значения $x_1, x_2, x_3, \ldots$ и соответствующие им значения $y_1, y_2, y_3, \ldots$, то по ним можно построить интерполяционный полином

(например, полином Ньютона) для зависимости $x = \varphi(y)$.

$$x = \varphi(y) \approx x_3 + (y - y_3)\varphi[y_3, y_2] + (y - y_3)(y - y_2)\varphi[$$



$$\varphi[y_k, y_j] = \frac{x_k - x_j}{y_k - y_j}$$

$$\varphi[y_k, y_j, y_l] = \frac{\varphi[y_k, y_j] - \varphi[y_j, y_l]}{y_k - y_l}$$

Искомый корень:

$$x = \varphi(0) \approx x_3 - y_3\varphi[y_3, y_2] + y_3 y_2 \varphi[y_3, y_2, y_1] + \ldots$$

$$x_{k+3} = x_{k+2} - y_{k+2}\varphi[y_{k+2}, y_{k+1}] + y_{k+2}y_{k+1}\varphi[y_{k+2}, y_{k+1}, y_k]$$

$$y_{k+3} = f(x_{k+3}),\ k = 1, 2, 3, \ldots$$

15. Численное интегрирование задачи Коши для обыкновенных дифференциальных уравнений (ОДУ)

15.1. Задача Коши для ОДУ

Пусть $t \in \mathbb{R}$, $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T \in \mathbb{R}^N$, $\mathbf{f} : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N$, $\dot{()} = d() / dt$. Требуется найти решение $\mathbf{x} = \mathbf{x}(t)$ нормальной системы ОДУ

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}) \tag{1.1}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{1.2}$$

Пусть $\exists G \subset \mathbb{R} \times \mathbb{R}^N : \forall (t, \mathbf{x}) \in G \ \ || \mathbf{f}(t, \mathbf{x}) || < M$ (1.3)

Полагаем $(t_0, \mathbf{x}_0) \in G$. Из (1.3) следует существование отрезка Пеано

$$| t - t_0 | \le h,\ h = h(t_0, \mathbf{x}_0) \tag{1.4}$$

причем

$$G^* = \{(t, \mathbf{x}) :| t - t_0 | \le a, \ || \mathbf{x} - \mathbf{x}_0 || \le b\} \in G \tag{1.5}$$
$$h = \min\{a, b / M\}$$

Пусть выполнено условие Липшица по $\mathbf{x}$

$$\forall (t, \mathbf{x}_1), (t, \mathbf{x}_2) \in G$$
$$|| \mathbf{f}(t, \mathbf{x}_1) - \mathbf{f}(t, \mathbf{x}_2) || \le L || \mathbf{x}_1 - \mathbf{x}_2 || \tag{1.6}$$

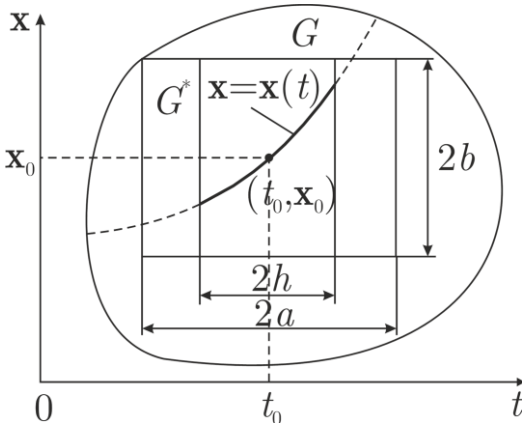


Рис. 1

Теорема Коши. Из (1.3), (1.5), (1.6) следует, что решение $\mathbf{x} = \mathbf{x}(t)$ задачи Коши (1.1), (1.2) существует и единственно по крайней мере в пре-

делах отрезка Пеано (1.4).

Доказательство теоремы, как известно, сводится к преобразованию исходной задачи Коши к интегральной форме записи

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{f}(\tau, \mathbf{x}(\tau))d\tau \tag{1.7}$$

и применению процесса последовательных приближений Пикара

$$\begin{aligned} &\mathbf{x}(t) = \lim_{m\to\infty} \mathbf{x}_m(t), \quad \mid t - t_0 \mid \leq h \\ &\mathbf{x}_{m+1}(t) = \mathbf{x}_0 + \int_{t_0}^{t} \mathbf{f}(\tau, \mathbf{x}_m(\tau))d\tau, \ m = 0,1,2,..., \ \mathbf{x}_0(t) = \mathbf{x}_0 \end{aligned} \tag{1.8}$$

Скорость роста искомого решения оценивается следующим образом

$$\| \mathbf{x}(t) - \mathbf{x}_0 \| \leq Const \cdot e^{L|t-t_0|} \tag{1.9}$$

Степень гладкости $\mathbf{x} = \mathbf{x}(t)$ на единицу выше, чем степень гладкости $\mathbf{f}(t, \mathbf{x})$

Аналогично рассматривается случай, когда правые части ОДУ (1.1) являются комплекснозначными: $t \in \mathbb{R}$, $\mathbf{x} \in \mathbb{C}^N$, $\mathbf{f} : \mathbb{R} \times \mathbb{C}^N \to \mathbb{C}^N$

Пусть $t \in \mathbb{C}$, $\mathbf{x} \in \mathbb{C}^N$, $\mathbf{f} : \mathbb{C} \times \mathbb{C}^N \to \mathbb{C}^N$ - аналитична в $G \subset \mathbb{C} \times \mathbb{C}^N$ по всем своим переменным. Тогда решение $\mathbf{x} = \mathbf{x}(t)$ аналитично в шаре (1.4), т.е.

$$\mathbf{x}(t) = \sum_{m=0}^{\infty} \mathbf{x}^{(m)}(t_0)(t - t_0)^m \ / \ m! \tag{1.10}$$

Последнее обстоятельство существенно для проектирования методов высокого порядка численного интегрирования ОДУ (1.1).

15.2. Явные методы Рунге-Кутта

Известно $\mathbf{x}(t_0) = \mathbf{x}_0$. Требуется найти

$$\mathbf{x}_1 \approx \mathbf{x}(t_0 + h) = \sum_{m=0}^{\infty} \mathbf{x}^{(m)}(t_0) h^m \ / \ m!$$

Пусть $s \in \mathbb{N}$ - число «стадий», a_{21}, a_{31}, a_{32},..., a_{s1}, a_{s2},..., $a_{s,s-1}$, b_1,..., b_s, c_2,..., c_s - действительные константы, причем $c_i = \sum_j a_{ij}$

Явный s-стадийный метод Рунге-Кутта имеет вид

$$\begin{aligned} &\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{x}_0) \\ &\mathbf{k}_2 = \mathbf{f}(t_0 + c_2 h, \mathbf{x}_0 + h a_{21} \mathbf{k}_1) \\ &\mathbf{k}_3 = \mathbf{f}(t_0 + c_3 h, \mathbf{x}_0 + h(a_{31}\mathbf{k}_1 + a_{32}\mathbf{k}_2)) \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &\mathbf{k}_s = \mathbf{f}\left[t_0 + c_s h, \mathbf{x}_0 + h\sum_{j=1}^{s-1} a_{sj}\mathbf{k}_j\right] \\ &\mathbf{x}_1 = \mathbf{x}_0 + h(b_1\mathbf{k}_1 + b_2\mathbf{k}_2 + \ldots + b_s\mathbf{k}_s) \end{aligned}$$

Параметры a_{21}, a_{31}, a_{32},..., a_{s1}, a_{s2},..., $a_{s,s-1}$, b_1,..., b_s, c_2,..., c_s находятся подстановкой (1.10) для $t = t_0 + h$ в исходные обыкновенные дифференциальные уравнения, разложением в ряд Тейлора по шагу h левых и правых частей обыкновенных дифференциальных уравнений и приравни-

ванием коэффициентов при одинаковых степенях h с целью достижения требуемого порядка локальной погрешности метода $p \in \mathbb{N}$:

$$|| \mathbf{x}(t_0 + h) - \mathbf{x}_1 || \leq Const \cdot h^{p+1}, \quad h \to 0 \tag{2.1}$$

Схематическая запись метода

$$\begin{array}{c|cccccc}
0 & & & & & & \\
c_2 & a_{21} & & & & & \\
c_3 & a_{31} & a_{32} & & & & \\
\ldots & \ldots & \ldots & \ldots & & & \\
c_s & a_{s1} & a_{s2} & \ldots & a_{s,s-1} & & \\
- & -- & -- & -- & -- & -- & \\
 & b_1 & b_2 & \ldots & b_{s-1} & b_s &
\end{array} \tag{2.2}$$

Метод Рунге-Кутта порядка 4

$$\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
-- & --- & ----- & --- & ---- \\
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

Вложенные методы порядка $p(q),\ q = p \pm 1$

$$\begin{array}{c|cccccc}
0 & & & & & & \\
c_2 & a_{21} & & & & & \\
c_3 & a_{31} & a_{32} & & & & \\
\ldots & \ldots & \ldots & & & & \\
c_s & a_{s1} & a_{s2} & \ldots & a_{s,s-1} & & \\
- & -- & -- & -- & -- & - & \\
 & b_1 & b_2 & \ldots & b_{s-1} & b_s & \\
- & -- & -- & -- & -- & - & \\
 & \hat{b}_1 & \hat{b}_2 & \ldots & \hat{b}_{s-1} & \hat{b}_s &
\end{array}$$

$$\begin{aligned}
&\mathbf{k}_1 = \mathbf{f}(t_0, \mathbf{x}_0) \\
&\mathbf{k}_2 = \mathbf{f}(t_0 + c_2 h, \mathbf{x}_0 + h a_{21} \mathbf{k}_1) \\
&\mathbf{k}_3 = \mathbf{f}(t_0 + c_3 h, \mathbf{x}_0 + h(a_{31} \mathbf{k}_1 + a_{32} \mathbf{k}_2)) \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&\mathbf{k}_s = \mathbf{f}\left[t_0 + c_s h, \mathbf{x}_0 + h \sum\nolimits_{j=1}^{s-1} a_{sj} \mathbf{k}_j\right] \\
&\mathbf{x}_1 = \mathbf{x}_0 + h(b_1 \mathbf{k}_1 + b_2 \mathbf{k}_2 + \ldots + b_s \mathbf{k}_s) \\
&\hat{\mathbf{x}}_1 = \hat{\mathbf{x}}_0 + h(\hat{b}_1 \mathbf{k}_1 + \hat{b}_2 \mathbf{k}_2 + \ldots + \hat{b}_s \mathbf{k}_s)
\end{aligned} \tag{2.3}$$

$|| \mathbf{x}_1 - \hat{\mathbf{x}}_1 ||$ оценивает локальную погрешность

Метод Дормана и Принса 5(4)

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
y_1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$\hat{y}_1$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

Непрерывные методы:

$\mathbf{x}(t_0 + \theta h) \approx \mathbf{x}_0 + h\sum_{j=1}^{s} b_j(\theta)\mathbf{k}_j,\ 0 \le \theta \le 1$

Непрерывный метод Дормана и Принса, 4-й порядок точности

$$\mathbf{x}(t_0 + \theta h) \approx \mathbf{x}_0 + h\sum_{j=1}^{6} b_j(\theta)\mathbf{k}_j,\ 0 \le \theta \le 1$$

$b_1(\theta) = \theta\,(1 + \theta\,(-1337/480 + \theta\,(1039/360 + \theta\,(-1163/1152))))$,
$b_2(\theta) = 0$,
$b_3(\theta) = 100\theta^2\,(1054/9275 + \theta\,(-4682/27825 + \theta\,(379/5565)))/3$,
$b_4(\theta) = -5\theta^2\,(27/40 + \theta\,(-9/5 + \theta\,(83/96)))/2$,
$b_5(\theta) = 18225\theta^2\,(-3/250 + \theta\,(22/375 + \theta\,(-37/600)))/848$,
$b_6(\theta) = -22\theta^2\,(-3/10 + \theta\,(29/30 + \theta\,(-17/24)))/7$,

15.3. Жесткие и нежесткие задачи.

Пусть $\lambda_j,\ j = 1,2,...,n$ - собственные значения матрицы Якоби $\partial\mathbf{f}(t,\mathbf{x}) / \partial\mathbf{x}$ правых частей ОДУ (1.1).

- Нежесткие задачи: $|\lambda_j| = O(1),\ j = 1,2,...,n$
- Жесткие задачи: есть как $|\lambda_j| = O(1)$, так и $|\lambda_j| \gg 1$

Явные методы Рунге-Кутта пригодны лишь для нежестких задач.

15.4. Явный и неявный многошаговые методы Адамса

Пригодны лишь для нежестких задач.

При $t = t_0,\ t = t_j = t_{j-1} + h_j,\ j = 1,2,...,n$ известно:

$$\mathbf{x}_j = \mathbf{x}(t_j),\quad j = 0,1,2,...,n \tag{4.1}$$

При $t = t_{n+1} = t_n + h_{n+1}$ требуется найти $\mathbf{x}_{n+1} = \mathbf{x}(t_{n+1})$.

$$\mathbf{F}(t) = \mathbf{f}(t, \mathbf{x}(t)) \tag{4.2}$$

Т. е. известны $\mathbf{F}_j = \mathbf{f}(t_j, \mathbf{x}_j), \quad j = 0,1,2,...,n$ (4.3)

По наборам t_j и $\mathbf{F}_j$ строится интерполяционный полином Ньютона:

$$\mathbf{F}(t) \approx \mathbf{P}_n(t) = \mathbf{F}_n + (t-t_n)\mathbf{F}[t_n, t_{n-1}] + (t-t_n)(t-t_{n-1})\mathbf{F}[t_n, t_{n-1}, t_{n-2}] + ... +$$
$$+(t-t_n)(t-t_{n-1})(t-t_{n-2})...(t-t_1)\mathbf{F}[t_n, t_{n-1}, t_{n-2}, ..., t_1, t_0]$$

$$\mathbf{F}[t_k, t_j] = \frac{\mathbf{F}_k - \mathbf{F}_j}{t_k - t_j}, \quad \mathbf{F}[t_k, t_j, t_i] = \frac{\mathbf{F}[t_k, t_j] - \mathbf{F}[t_j, t_i]}{t_k - t_i}$$

$$\mathbf{F}[t_k, t_j, t_i, t_l] = \frac{\mathbf{F}[t_k, t_j, t_i] - \mathbf{F}[t_j, t_i, t_l]}{t_k - t_l}$$

Явный метод Адамса.

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \int_{t_n}^{t_{n+1}} \mathbf{F}(t)dt \approx \mathbf{x}_n + \int_{t_n}^{t_{n+1}} \mathbf{P}_n(t)dt =$$
$$= \mathbf{x}_n + \mathbf{F}_n(t_{n+1} - t_n) + \mathbf{F}[t_n, t_{n-1}]\frac{(t_{n+1} - t_n)^2}{2} + \tag{4.4}$$
$$+\mathbf{F}[t_n, t_{n-1}, t_{n-2}]\int_{t_n}^{t_{n+1}} (t-t_n)(t-t_{n-1})dt + ... +$$
$$+\mathbf{F}[t_n, t_{n-1}, t_{n-2}, ..., t_1, t_0]\int_{t_n}^{t_{n+1}} (t-t_n)(t-t_{n-1})(t-t_{n-2})...(t-t_1)dt$$

При $h_1 = h_2 = ... = h_n = h_{n+1} = h$ явный метод Адамса (4.4) имеет порядок $n+1$.

Неявный метод Адамса.

Добавляя формально к (4.1), (4.3) величины

$$t_{n+1} \quad \text{и} \quad \mathbf{F}_{n+1} = \mathbf{f}(t_{n+1}, \mathbf{x}_{n+1}) \tag{4.5}$$

запишем формально интерполяционный полином Ньютона

$$\mathbf{F}(t) \approx \mathbf{P}_{n+1}(t) = \mathbf{F}_{n+1} + (t-t_{n+1})\mathbf{F}[t_{n+1}, t_n] + (t-t_{n+1})(t-t_n)\mathbf{F}[t_{n+1}, t_n, t_{n-1}] +$$
$$+(t-t_{n+1})(t-t_n)(t-t_{n-1})\mathbf{F}[t_{n+1}, t_n, t_{n-1}, t_{n-2}] + ... +$$
$$+(t-t_{n+1})(t-t_n)(t-t_{n-1})(t-t_{n-2})...(t-t_1)\mathbf{F}[t_{n+1}, t_n, t_{n-1}, t_{n-2}, ..., t_1, t_0]$$

Тогда

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \int_{t_n}^{t_{n+1}} \mathbf{F}(t)dt \approx \mathbf{x}_n + \int_{t_n}^{t_{n+1}} \mathbf{P}_{n+1}(t)dt = \mathbf{x}_n + \mathbf{F}_{n+1}(t_{n+1} - t_n) -$$
$$-\mathbf{F}[t_{n+1}, t_n]\frac{(t_{n+1} - t_n)^2}{2} - \mathbf{F}[t_{n+1}, t_n, t_{n-1}]\frac{(t_{n+1} - t_n)^3}{6} + \tag{4.6}$$
$$+\mathbf{F}[t_{n+1}, t_n, t_{n-1}, t_{n-2}]\int_{t_n}^{t_{n+1}} (t-t_{n+1})(t-t_n)(t-t_{n-1})dt + ... +$$
$$+\mathbf{F}[t_{n+1}, t_n, t_{n-1}, t_{n-2}, ..., t_1, t_0]\int_{t_n}^{t_{n+1}} (t-t_{n+1})(t-t_n)(t-t_{n-1})(t-t_{n-2})...(t-t_1)dt$$

Последнее равенство представляет собой систему нелинейных уравнений относительно $\mathbf{x}_{n+1}$. При $h_1 = h_2 = ... = h_n = h_{n+1} = h$ неявный ме-

тод Адамса (4.6) имеет порядок $n+2$. Сначала приближенное значение $\mathbf{x}_{n+1}$ находят явным методом (4.4), а затем последовательно уточняют при помощи (4.6).

15.5. ФДН-метод (Гира)

Полагаем формально, что известны величины

$$\mathbf{x}_j = \mathbf{x}(t_j), \quad j = 0,1,2,...,n+1 \tag{5.1}$$

Интерполяционный полином Лагранжа, построенный по набору (5.1), имеет вид

$$\mathbf{x}(t) \approx \mathbf{\Lambda}_{n+1}(t) = \sum_{k=0}^{n+1} \mathbf{x}_k \left[\prod_{j=0, j\neq k}^{n+1} (t_k - t_j) \right]^{-1} \prod_{j=0, j\neq k}^{n+1} (t - t_j), \;\; \dot{\mathbf{x}}(t) \approx \dot{\mathbf{\Lambda}}_{n+1}(t)$$

Следовательно,

$$\dot{\mathbf{x}}(t_{n+1}) = \mathbf{f}(t_{n+1}, \mathbf{x}_{n+1}) \approx \dot{\mathbf{\Lambda}}_{n+1}(t_{n+1}),$$

откуда следует система нелинейных уравнений для нахождения компонент величины $\mathbf{x}_{n+1}$

$$\mathbf{x}_{n+1} \sum_{j=0}^{n} \frac{1}{t_{n+1} - t_j} + \sum_{k=0}^{n} \mathbf{x}_k \left[\prod_{j=0, j\neq k}^{n+1} (t_k - t_j) \right]^{-1} \prod_{j=0, j\neq k}^{n} (t_{n+1} - t_j) = \\ = \mathbf{f}(t_{n+1}, \mathbf{x}_{n+1}) \tag{5.2}$$

часто называемая «формулами дифференцирования назад» (ФДН). ФДН-метод при $h_1 = h_2 = ... = h_n = h_{n+1} = h$ имеет порядок $n+1$. ФДН-метод неустойчив при $n > 5$, «жестко» устойчив при $n = 0,1,2,3,4,5$ и применяется для решения «жестких» задач.

Величина $\mathbf{x}_{n+1}$ является решением $\mathbf{y} = \mathbf{x}_{n+1}$ системы нелинейных уравнений

$$\mathbf{\Phi}(\mathbf{y}) = 0, \quad \mathbf{\Phi}(\mathbf{y}) = -\mathbf{f}(t_{n+1}, \mathbf{y}) + \\ + \mathbf{y} \sum_{j=0}^{n} \frac{1}{t_{n+1} - t_j} + \sum_{k=0}^{n} \mathbf{x}_k \left[\prod_{j=0, j\neq k}^{n+1} (t_k - t_j) \right]^{-1} \prod_{j=0, j\neq k}^{n} (t_{n+1} - t_j)$$

для решения которой обычно применяется итерационный метод Ньютона:

$$\mathbf{y}_{\nu+1} = \mathbf{y}_\nu - [\partial \mathbf{\Phi}(\mathbf{y}_\nu) / \partial \mathbf{y}]^{-1} \mathbf{\Phi}(\mathbf{y}_\nu)$$

$$\frac{\partial \mathbf{\Phi}(\mathbf{y})}{\partial \mathbf{y}} = \mathrm{diag}\{1,1,1,...,1\} \sum_{j=0}^{n} \frac{1}{t_{n+1} - t_j} - \frac{\partial \mathbf{f}(t_{n+1}, \mathbf{y})}{\partial \mathbf{y}} \tag{5.4}$$

Поскольку при реализации метода Ньютона матрица $\partial \mathbf{\Phi}(\mathbf{y}) / \partial \mathbf{y}$ может вычисляться с меньшей точностью, чем сами левые части $\mathbf{\Phi}(\mathbf{y})$ системы нелинейных уравнений, то матрица правых частей $\partial \mathbf{f}(t, \mathbf{x}) / \partial \mathbf{x}$ системы обыкновенных дифференциальных уравнений может вычисляться с меньшей точностью, чем сами правые части $\mathbf{f}(t, \mathbf{x})$.

15.6. ОДУ, линейные относительно производных

Модельная система обыкновенных дифференциальных уравнений имеет вид

$$M(t,\mathbf{x})\dot{\mathbf{x}} = \mathbf{f}(t,\mathbf{x}), \tag{6.1}$$

где $M(t,\mathbf{x})$- матрица размерности $N \times N$, называемая матрицей инерции. При численном интегрировании последней системы обыкновенных дифференциальных уравнений возможны следующие случаи:

- Задача является нежесткой, либо размерность системы невелика. В данном случае система ОДУ приводится к нормальной форме Коши

$$\dot{\mathbf{x}} = \mathbf{g}(t,\mathbf{x}), \quad \mathbf{g}(t,\mathbf{x}) = [M(t,\mathbf{x})]^{-1}\mathbf{f}(t,\mathbf{x}) \tag{6.2}$$

и далее интегрируется численно.

- Задача характеризуется большой размерностью, является жесткой, а матрицы $M(t,\mathbf{x})$ и $\partial\mathbf{f}(t,\mathbf{x}) / \partial\mathbf{x}$ являются разреженными. Такая ситуация достаточно часто встречается в математической физике при дискретизации модельных уравнений в частных производных по независимым пространственным переменным. Формальное применение ФДН-метода к (6.2) приводит к результату

$$\mathbf{x}_{n+1}\sum_{j=0}^{n}\frac{1}{t_{n+1}-t_j} + \sum_{k=0}^{n}\mathbf{x}_k\left[\prod_{j=0,j\neq k}^{n+1}(t_k-t_j)\right]^{-1}\prod_{j=0,j\neq k}^{n}(t_{n+1}-t_j) = \\ = [M(t_{n+1},\mathbf{x}_{n+1})]^{-1}\mathbf{f}(t_{n+1},\mathbf{x}_{n+1}) \tag{6.3}$$

Искомая величина $\mathbf{x}_{n+1}$ является решением $\mathbf{y} = \mathbf{x}_{n+1}$ системы нелинейных уравнений (6.3), которые можно представить в форме

$$\boldsymbol{\Phi}(\mathbf{y}) = 0, \quad \boldsymbol{\Phi}(\mathbf{y}) = -\mathbf{f}(t_{n+1},\mathbf{y}) + \\ + M(t_{n+1},\mathbf{y})\left\{\mathbf{y}\sum_{j=0}^{n}\frac{1}{t_{n+1}-t_j} + \sum_{k=0}^{n}\mathbf{x}_k\left[\prod_{j=0,j\neq k}^{n+1}(t_k-t_j)\right]^{-1}\prod_{j=0,j\neq k}^{n}(t_{n+1}-t_j)\right\} \tag{6.4}$$

Как правило, при дискретизации начально-краевых задач математической физики по пространственным переменным матрица инерции слабо изменяется при изменении искомого решения, и

$$\partial M(t,\mathbf{x}) / \partial\mathbf{x} \approx 0 \tag{6.5}$$

$$\frac{\partial\boldsymbol{\Phi}(\mathbf{y})}{\partial\mathbf{y}} \approx M(t_{n+1},\mathbf{y})\sum_{j=0}^{n}\frac{1}{t_{n+1}-t_j} - \frac{\partial\mathbf{f}(t_{n+1},\mathbf{y})}{\partial\mathbf{y}} \tag{6.6}$$

15.7. Функции MATLAB решения задачи Коши для ОДУ

$$\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y}), \quad \mathbf{y}(t_0) = \mathbf{y}_0 \tag{7.1}$$

$$M(t,\mathbf{y})\dot{\mathbf{y}} = \mathbf{f}(t,\mathbf{y}), \quad \mathbf{y}(t_0) = \mathbf{y}_0 \tag{7.2}$$

`[T,Y]=solver(odefun,tspan,y0)`, где **`tspan=[t0 tf]`** – числ. решение (7.1) или (7.2) от начального момента времени `t0` до финального момента времени `tf` с начальными значениями `y0`. Здесь

odefun – дескриптор функции, вычисляющей правые части (7.1) или (7.2), сигнатура которой имеет вид f=odefun(t,y). Здесь t – скаляр, y, f – столбцевые векторы. Выходной столбцевой вектор T содержит моменты времени, для которых численно найдено решение ОДУ. Каждая строка выходной матрицы Y содержит численно найденное решение ОДУ в момент времени, указанный в соотв. элементе столбцевого вектора T. Если требуется найти решение ОДУ в специфичные моменты времени (все убывающие либо возрастающие) , tspan=[t0,t1,t2,…,tf].

Sol=solver(odefun,[t0 tf],y0);

Y=deval(Sol,T); Y=deval(T, Sol);

– численное решение ОДУ и сохранение информации в структуре Sol с последующим интерполированием искомого решения в пределах отрезка интегрирования;

Y=deval(Sol,T,Idx); Y=deval(Sol,T,Idx);

–возврат нужных компонент решения

Solver	Описание
ode45	«Нежесткие» ОДУ, явный метод Дормана и принса 5(4)
ode23	«Нежесткие» ОДУ, явный метод Рунге-Кутта
ode113	«Нежесткие» ОДУ, явно-неявный метод Адамса переменного шага и порядка
ode15s	«Жесткие» ОДУ либо дифференциально-алгебраические уравнения. Неявный ФДН-метод переменного шага и порядка
ode23s	«Жесткие» ОДУ, метод Розенброка
ode23t	«Умеренно жесткие» ОДУ либо дифференциально-алгебраические уравнения
ode23tb	«Жесткие» ОДУ. Модификация ФДН-метода
ode15i	«Полностью неявные ОДУ» $\mathbf{F}(t,\mathbf{y},\dot{\mathbf{y}})=0$. ФДН-метод

[T,Y]=solver(odefun,tspan,y0,options);

Sol=solver(odefun,[t0 tf],y0,options);

– задание параметров алгоритма в полях структуры options.

Задание полей структуры options:

options=odeset('Имя1',Значение1,'Имя2',Значение2, …);

options=odeset(oldopts,'Имя2',Значение2,…);

Имя поля	**Знач. по умолчанию**/Описание
Все решатели	
RelTol	$\mathbf{10^{-3}}$. Относит. погрешн. числ. интегрирования
AbsTol	$\mathbf{10^{-6}}$. Скаляр или вектор. Абсолютная погрешность нахождения отдельных компонент
InitialStep	Начальная величина шага интегрирования
MaxStep	**1/10 длины отрезка интегр.** Максимальная

	величина шага интегрирования
	«Жесткие» решатели
`Jacobian`	Дескриптор функции либо постоянная матрица $\partial \mathbf{f}(t,\mathbf{y}) / \partial \mathbf{y}$. Сигнатура: `J=Jac(t,y)`. Здесь `t` – скаляр, `y` – столбцевой вектор, `J` – матрица
`JPattern`	Разреженная матрица с единичными ненулевыми элементами, задающими расположение ненулевых элементов матрицы $\partial \mathbf{f}(t,\mathbf{y}) / \partial \mathbf{y}$, которые находятся численно
`Vectorised`	`on`|**`off`**. Сокращение числа вызовов функции
	`ode15s, ode23t`
`Mass`	Дескриптор ф-ии либо постоян. матрица $M(t,\mathbf{y})$
`MStateDependence`	`none`|**`weak`**|`strong`. Степень зависимости матрицы $M(t,\mathbf{y})$ от $\mathbf{y}$. По умолчанию – слабая
`MvPattern`	Разреженная матрица, характеризующая заполненность матрицы $\partial(M(t,\mathbf{y})\mathbf{v}) / \partial \mathbf{y}$. Требуется лишь когда `MStateDependence` имеет значение `strong`.
`MassSingular`	`yes`|`no`|**`maybe`**. Может ли быть вырождена матрица $M(t,\mathbf{y})$
`InitialSlope`	**0**. Вектор $\mathbf{y}_{p_0}$, удовлетворяющий ОДУ (7.2)
	`ode15s, ode15i`
`MaxOrder`	`1`|`2`|`3`|`4`|**`5`**. Максимальный порядок метода.

ЛИТЕРАТУРА

1. Поршнев С.В. Компьютерное моделирование физических процессов в пакете MATLAB. – М.: Горячая линия – Телеком. – 2003. – 592 с.
2. Иглин С.П. Математические расчеты на базе MATLAB – СПб. : БХВ-Петербург, 2005. – 634 с.
3. Дьяконов В. П., Круглов В. Математические пакеты расширения MATLAB [Текст] : спец. справ. / - СПб. : Питер, 2001. - 475 с.
4. Голуб Дж., Ван Лоун Ч. Матричные вычисления. – М: Мир, 1999. – 536 с.
5. Васильев Ф.П. Методы оптимизации. . – М.: Факториал Пресс, 2002. – 823 с.
6. Ануфриев И. Е., Смирнов А. Б., Смирнова Е. Н. Matlab 7: учеб. пособие - СПб. : БХВ-Петербург, 2005. - 1080 с.
7. Эстербю О., Златев З. Прямые методы для разреженных матриц. – М.: Мир, 1987. – 120 с.
8. Вержбицкий В.М. Основы численных методов. – М.: Высш. шк., 2002. – 840 с.
9. Хайрер Э., Нерсетт С., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Нежесткие задачи. – М.: Мир, 1990. – 512 с.
10. Хайрер Э., Ваннер Г. Решение обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи. – М.: Мир, 1999. – 685 с.

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ 3
1. Элементы языка MATLAB 5
1.1. Основные типы данных 5
1.2. Нечисловые типы данных 6
1.3. Структуры 7
1.4. Массивы ячеек (cell arrays) 9
2. Операции над данными 9
2.1. Операции над числовыми скалярами 9
2.2. Основные команды среды MATLAB 11
2.3. Скрипты системы MATLAB 12
2.4. Формирование числовых матриц 12
2.5. Арифметические операции над матрицами и массивами 15
2.6. Логические операции 16
3. Функции и управляющие конструкции MATLAB 17
3.1. Управляющие конструкции языка MATLAB 17
3.2. Пользовательские функции MATLAB 20
3.3. Передача произвольного числа входных и выходных параметров 23
3.4. Дескрипторы функций 24
3.5. Внешние интерфейсы (External Interfaces) 24
4. Объектно-ориентированное программирование в MATLAB 28
5. Графические возможности MATLAB 36
6. Поддержка в MATLAB разреженных матриц 38
6.1. Внутренний формат хранения разреженных матриц 38
6.2. Создание разреженных матриц 39
6.3. Функции для работы с разреженными матрицами 40
6.4. Операции над разреженными матрицами 41
7. Факторизация матриц, решение систем линейных уравнений и задач «метода наименьших квадратов» 42
7.1. Методы факторизации матриц 42
7.2. LU-разложение 45
7.3. Разложение Холецкого 46
7.4. LDLH-разложение 46

7.5. QR-факторизация 47
7.6. Решение систем линейных алгебраических уравнений 48
7.7. Алгоритм работы решателей систем линейных алгебраических уравнений 49
7.8. Пояснение к методу наименьших квадратов 50
7.9. Дополнительные функции, реализующие метод наименьших квадратов 52
8. Решение задач на собственные значения и другие функции линейной алгебры 53
8.1. Собственные векторы и собственные значения 53
8.2. QR-алгоритм для нахождения спектра плотно заполненных матриц (LAPACK) 53
8.3. Ускорение QR-алгоритма при помощи приведения исходной матрицы к верхней квазитреугольной форме 54
8.4. Обобщенная задача на собственные значения 55
8.5. Функции MATLAB для решения задач на собственные значения 56
8.6. Частичная задача на собственные значения и спектральные преобразования 57
8.7. Приближенное определение $n \ll N$ максимальных собственных значений 57
8.8. Неявный итерационный метод Ланцоша-Арнольди (ARPACK) 58
8.9. Вызовы MATLAB (Octave) для решения частичной задачи на собственные значения 59
8.10. Некоторые функции линейной алгебры 61
8.11. Задача о нахождении собственных значений и собственных векторов матричного полинома 65
9. Матричные функции и обработка данных 66
9.1. Элементарные функции по обработке данных в MATLAB 66
9.2. Функции от матриц 68
9.3. Матричные математические функции MATLAB 69
9.4. Статистические функции 70
10. Интерполяция данных в MATLAB 72
10.1. Одномерная интерполяция 72
10.2. Тригонометрическая интерполяция периодических функций 74

10.3. Подготовка двумерных и трехмерных сеток для интерполяции ……74
10.4. Многомерная интерполяция данных 76
11. Численное интегрирование в MATLAB 77
11.1. Численное интегрирование функции одной переменной..... 77
11.2. Численное интегрирование функции двух переменных..... 79
12. Численное дифференцирование 81
12.1. Приближенное вычисление дифференциального оператора Лапласа 83
13. Дискретное преобразование Фурье в MATLAB 84
13.1. Одномерное дискретное преобразование Фурье 84
13.2. Двумерное дискретное преобразование Фурье 87
13.3. Многомерное дискретное преобразование Фурье..... 89
14. Стандартные функции MATLAB для решения задач оптимизации..... 90
14.1. Поиск локального минимума функции 91
14.2. Метод «золотого сечения»..... 92
14.3. Метод парабол..... 93
14.4. Поиск минимума действительной функции n действительных переменных 93
14.5. Безградиентный метод Нелдера-Мида (политопов) 95
14.6. Поиск корня непрерывной монотонной функции 96
15. Численное интегрирование задачи Коши для обыкновенных дифференциальных уравнений (ОДУ)..... 98
15.1. Задача Коши для ОДУ 98
15.2. Явные методы Рунге-Кутта 99
15.3. Жесткие и нежесткие задачи. 101
15.4. Явный и неявный многошаговые методы Адамса 101
15.5. ФДН-метод (Гира) 103
15.6. ОДУ, линейные относительно производных 104
15.7. Функции MATLAB решения задачи Коши для ОДУ 104
ЛИТЕРАТУРА 107