Министерство образования и науки Российской Федерации
Нижегородский государственный университет им. Н. И. Лобачевского
Факультет вычислительной математики и кибернетики

Н. Ю. Золотых, М. М. Шульц

Численные методы линейной алгебры

Лабораторная работа

Нижний Новгород
2012

Оглавление

Предисловие

Целью работы является ознакомление студентов с понятиями, недостаточно освещенными (или совсем не освещенными) в курсе «Геометрия и алгебра» (приближенное решение систем линейных уравнений, число обусловленности матрицы, полиномиальная аппроксимация).

Кроме того, студент знакомится с основами системы MATLAB, главным образом, для решения систем линейных уравнений. Рассматриваются вопросы генерации псевдослучайных данных. В связи с численным решением систем рассматривается понятие обусловленности матриц.

1. Работа в системе MATLAB

MATLAB (сокращение от «Matrix Laboratory» – «матричная лаборатория») – мощная интерактивная среда для научных вычислений со своим языком программирования, гибкими графическими возможностями, средствами сопряжения с другими языками и несколькими десятками пакетов приложений.

В настоящем пособии мы описываем лишь небольшую часть возможностей MATLABa, которой, тем не менее, вполне достаточно для достижения целей лабораторной работы. Предполагается использование компьютера с загруженной программой MATLAB «в режиме калькулятора», т.е. путем ввода данных и выполнения некоторых операций с этими данными. Мы не останавливаемся на конструировании собственных функций, хотя некоторые сведения об этом можно почерпнуть из Приложения. Ввод данных и команд происходит через *командное окно* (`Command Window`) на постоянно присутствующую подсказку MATLABa `>>`, в командное окно выводятся и полученные результаты. Вывод результата любой команды можно заблокировать, поставив в ее конце точку с запятой. Содержимое командного окна (полностью или частично) можно скопировать и через буфер обмена вставить, например, в файл отчета (`*.doc`).

Кроме командного окна, желательно иметь на экране *историю команд* (`Command History`), в которой запоминаются все команды, введенные пользователем в данном сеансе. Эти команды можно повторно выполнять (в прежнем виде или предварительно отредактировав). Последняя команда может быть вызвана в командное окно клавишей ↑, ее повторное нажатие вызывает предпоследнюю команду и т.д.

Набор используемых окон можно задать, отметив их в «выпадающем меню» команды `Desktop` основного меню.

В настоящем пособии мы описываем лишь базовые возможности рассматриваемых команд и функций MATLABa. Справку о любой команде или функции можно получить, набрав в командном окне

```
>> lookfor имя_функции
```
(определение функции «в одну строку»),

```
>> help имя_функции
```
(более развернутое описание),

```
>> doc имя_функции
```
(наиболее полное описание).

Замечательным свойством системы MATLAB является ее открытость: многие функции доступны в виде исходных текстов, которые можно изучать. Для этого надо набрать

```
>> edit имя_функции
```

Этот файл следует только читать, его можно скопировать и распечатать. **Ничего изменять в этом файле не следует**.

Сводка основных команд и функций MATLABa приведена в Приложении.

1.1. Ввод, сохранение и загрузка

Небольшие матрицы можно набирать непосредственно в командном окне. Слева и справа матрица ограничивается *квадратными скобками*. Например

```
>> A = [1 2; 3 4]

A =

     1     2

     3     4
```

Обратите внимание: разделитель между числами в строке – *пробел* (вместо него можно использовать *запятую*), разделитель между строками – *точка с запятой* (вместо нее можно использовать клавишу `Enter`)

```
>> A = [1, 2

3, 4]
```

```
A =

     1     2

     3     4
```

В частности, можно ввести вектор-строку, например

```
>> b = [1 2 3 4]

b =

     1     2     3     4
```

или вектор-столбец

```
>> c = [1; 2; 3; 4]


c =

     1

     2

     3

     4
```

Чтобы не использовать большого числа точек с запятыми, можно набрать вектор как строку, а после закрывающей скобки поставить знак транспонирования – апостроф (')

```
>> c = [1 2 3 4]'

c =

     1

     2

     3

     4
```

MATLAB допускает использование *комплексных чисел*. Ввод комплексного числа:

```
>> t=1+2i

t =

       1       +    2i
```

Операция апостроф ('), примененная к комплексным данным, действует несколько иначе – вместе с транспонированием происходит замена комплексных чисел на сопряженные:

```
>> a = [1+i 2+2i]

a =

   1.0000 + 1.0000i   2.0000 + 2.0000i

>> ap = a'

ap =

   1.0000 - 1.0000i

   2.0000 - 2.0000i
```

«Чистое» транспонирование получается с помощью операции (.')

```
>> apt = a.'

apt =

   1.0000 + 1.0000i

   2.0000 + 2.0000i
```

Доступ к элементам матрицы осуществляется по паре индексов: индексы заключаются в круглые скобки и разделяются запятой. Как обычно, вначале указывается номер строки, а затем – номер столбца

```
>> A(1, 2)

ans =

     2
```

Здесь `ans` («ответ», англ) – автоматически создающаяся переменная, которой присваивается последнее вычисленное значение, если в операции присваивания отсутствует левая часть (имя переменной):

```
>> 2+2

ans =

        4
```

Если левой частью операции присваивания является элемент матрицы, задаваемое операцией значение «вставляется» в матрицу:

```
>> A(1, 2) = -2
```

```
A =

     1    -2

     3     4
```

К элементам векторов-строк и векторов-столбцов можно обращаться с помощью одного индекса

```
>> b(3)

ans =

     3

>> c(3)

ans =

     3
```

Узнать размеры матрицы можно с помощью функции `size`, а длину векторов – с помощью функции `length`

```
>> size(A)

ans =

     2     2

>> length(b)

ans =

     4

>> length(c)

ans =

     4
```

Заметим, что команда `size(A, 1)` возвращает число строк матрицы `A`, а команда `size(A, 2)` – число столбцов.

Значения всех переменных, введенные (вместе с именами) через командное окно, хранятся в *рабочем пространстве* (Workspace) и могут свободно использоваться в пределах текущего сеанса.

Информацию об этих переменных можно получить с помощью команд who и whos. Первая выдает список всех переменных рабочего пространства, вторая для каждой переменной сообщает размер соответствующего массива (простая переменная в MATLABe – это массив 1×1), занимаемую память в байтах и *класс* переменной (например, double array).

Workspace можно вывести на экран, отметив его в меню команды Desktop.

Чтобы сохранить переменные рабочего пространства в памяти, необходимо использовать (в том или ином варианте) команду save. По этой команде все переменные рабочего пространства (в примере это A, b, c) сохраняются в файле с расширением .mat. Если имя файла в команде не указано, по умолчанию используется файл matlab.mat. Он сохраняется в *текущей директории* (Current Directory), которую по ходу сеанса выбирает пользователь.

Простейший вариант

```
>> save

Saving to: matlab.mat
```

Проделаем небольшой эксперимент: очистим рабочее пространство, выбрав в меню Edit пункт Clear Workspace. После этого попытки вызвать введенные ранее переменные не приводят к успеху

```
>> A

??? Undefined function or variable 'A'.
```

Чтобы вновь сделать эти переменные доступными, нужно выполнить команду load Ее простейший вариант

```
>> load

Loading from: matlab.mat
```

После этого вновь вызовем переменную

```
>> A

A =

     1     2

     3     4
```

Команды `save` и `load` имеют много вариантов. Можно сохранять (загружать) не в файл по умолчанию (`matlab.mat`), а в файл с именем, заданным пользователем (но с расширением `.mat`). Можно сохранять (загружать) не все переменные рабочего пространства, тогда в команде надо указать их имена.

Некоторым неудобством является то, что файл с расширением `.mat` является двоичным, поэтому его практически невозможно редактировать. Эту трудность можно преодолеть, если использовать команды `save` и `load` с параметром `-ascii`. В этом случае создается текстовый файл, имя (и расширение) задает пользователь, этот файл можно редактировать (и создавать) в любом текстовом редакторе.

Форматы команд `save` и `load` приведены в Приложении.

Еще одна возможность – чтение файла, содержащего матрицу, с помощью команды `textread`. В этом случае файл должен быть заранее создан (или отредактирован) с помощью какого-либо текстового редактора, например, встроенного в диалоговую оболочку `Far` операционной системы. Этот редактор активизируется клавишей `F4` (редактирование имеющегося файла) или комбинацией клавиш `Shift+F4` (создание нового файла). Имя и расширение при создании файла задается произвольно, расширение может отсутствовать.

Пусть в каталоге `E:\mtr\` имеется файл `a4x4`, содержащий некоторую матрицу 4×4. Тогда для ее загрузки в переменную `A` можно выполнить команду

```
>> A = textread('E:\mtr\a4x4');
```

(точка с запятой в конце строки блокирует вывод на экран).

1.2. Матричные и покомпонентные операции

С матрицами можно выполнять операции: матричные сложение, вычитание, умножение

```
>> A = [1 2; 3 4]
```

```
A =

     1     2

     3     4

>> B = [5 6; 7 8]

B =

     5     6

     7     8

>> A + B

ans =

     6     8

    10    12

>> A - B

ans =

    -4    -4

    -4    -4

>> A * B

ans =

    19    22

    43    50
```

Заметим, что операция «звездочка» (`*`) осуществляет матричное умножение. Для покомпонентного умножения имеется операция «звездочка с точкой» (`.*`, сравните с командой `.'`)

```
>> A .* B
ans =
     5    12
```

```
    21    32
```

Кроме того, доступны операции левого и правого матричного деления, о них подробнее см. в разделе 2. Для покомпонентного деления используйте операцию (`./`). Предварительно стоит выполнить команду

```
>> format rat
```

– после нее числа будут выводиться в виде дробей «с числителем и знаменателем» (тот же результат можно получить, выбрав формат `rational` в подменю `Preferences` меню `File`).

```
>> A ./ B
ans =
       1/5            1/3
       3/7            1/2
```

Операция (`^`) используется для возведения матрицы в степень, а (`.^`) – для покомпонентного возведения в степень ее элементов

```
>> A^2
ans =
     7    10
    15    22
>> A.^B
ans =
           1          64
        2187       65536
```

Математические функции, примененные к матрицам, выполняются покомпонентно, например,

```
>> sqrt(A)
ans =
    1.0000    1.4142
    1.7321    2.0000
```

1.3. Конструирование матриц

MATLAB предоставляет широкие возможности конструирования матриц, т. е. создания новых матриц из имеющихся.

Пусть имеем матрицы `A` и `B` (см. выше).

Построим их ***конкатенацию*** (сцепление), сначала горизонтальную, а затем вертикальную. Основные инструменты – квадратные скобки, пробел (или запятая) и точка с запятой.

```
>> Chor = [A B]

Chor =

     1     2     5     6

     3     4     7     8

>> Cvert = [A; B]

Cvert =

     1     2

     3     4

     5     6

     7     8
```

Многократное ***копирование*** имеющейся матрицы

```
>> repmat(A, 1, 2)

ans =

     1     2     1     2

     3     4     3     4

>> repmat(A, 2, 1)

ans =

     1     2

     3     4

     1     2

     3     4

>> repmat(A, 2)
```

```
ans =

     1     2     1     2

     3     4     3     4

     1     2     1     2

     3     4     3     4
```

Из имеющихся матриц можно построить ***блочно-диагональную***

```
>> H = blkdiag(A, B)

H =

     1     2     0     0

     3     4     0     0

     0     0     5     6

     0     0     7     8
```

Имеющуюся матрицу можно ***перевернуть*** «вверх ногами»

```
>> flipud(A)

ans =

     3     4

     1     2
```

или «задом наперед»

```
>> fliplr(A)

ans =

     2     1

     4     3
```

Из имеющихся матриц можно извлекать фрагменты и использовать их автономно (под новыми именами) или для конструирования новых матриц.

Выделение строк

```
>> A1 = A(1, :)

A1 =

     1     2

>> A2 = A(2, :)

A2 =

     3     4
```

Выделение столбцов

```
>> B1 = B(:, 1)

B1 =

     5

     7

>> B2 = B(:, 2)

B2 =

     6

     8
```

Выделение диагонали

```
>> DA = diag(A)

DA =

     1

     4
```

Формирование матрицы с заданной диагональю

```
>> F = diag([1 2 3])
```

```
F =

     1     0     0

     0     2     0

     0     0     3
```

Формирование матрицы с заданной кодиагональю (наклонная линия, отстоящая от диагонали на заданное число шагов)

```
>> F1 = diag([1 2 3], 1)

F1 =

     0     1     0     0

     0     0     2     0

     0     0     0     3

     0     0     0     0

>> F2 = diag([1 2], -2)

F2 =

     0     0     0     0

     0     0     0     0

     1     0     0     0

     0     2     0     0
```

Формирование матрицы с диагональю из заданной матрицы

```
>> DD = diag(diag(A))

DD =

     1     0

     0     4
```

Более содержательный пример – ***формирование жордановой клетки*** 4×4 с собственным числом $\lambda_0 = 2$ на диагонали

```
>> J4 = diag(repmat(2, 1, 4)) + diag(ones(1, 3), 1)

J4 =

     2     1     0     0

     0     2     1     0

     0     0     2     1

     0     0     0     2
```

Тот же результат можно получить с помощью системной функции `gallery`, которая (в зависимости от входных параметров) позволяет генерировать более 50 типов матриц. Команда, создающая жорданову клетку, имеет вид

```
>> J4 = gallery('jordbloc', 4, 2);
```

(вывод заблокирован).

Выделение верхнего и нижнего треугольников матрицы (вместе с диагональю)

```
>> U = triu(A)

U =

     1     2

     0     4

>> L = tril(A)

L =

     1     0

     3     4
```

Из полученных фрагментов можно, например, восстановить исходную матрицу `A`

```
>> U + L - DD
```

```
ans =

     1     2

     3     4
```

Хотя правило Крамера и не является практически приемлемым способом решения систем линейных уравнений значительного размера, покажем в виде иллюстрации, как найти при помощи этого правила две из четырех неизвестных системы с матрицей

```
>> M = textread('E:\mtr\a4x4')

M =

     3     4     1     2

     3     5     3     5

     6     8     1     5

     3     5     3     7
```

и столбцом свободных членов

```
>> b = textread('E:\mtr\b4x1')

b =

     3

     6

     8

     8
```

Найдем определитель матрицы

```
 >> d = det(M)

 d =

     -6
```

Определитель не равен нулю, правило Крамера применимо. Надо вычислить еще четыре определителя (мы ограничимся двумя), в каждом которых один столбец матрицы `A` заменен столбцом `b`. Сначала сформируем соответствующие матрицы

```
>> M1 = [b M(:, 2:4)]

M1 =

     3     4     1     2

     6     5     3     5

     8     8     1     5

     8     5     3     7

>> M2 = [M(:, 1) b M(:, 3:4)]

M2 =

     3     3     1     2

     3     6     3     5

     6     8     1     5

     3     8     3     7
```

Затем вычислим определители

```
>> d1 = det(M1)

d1 =

    12

>> d2 = det(M2)

d2 =

   -12
```

Теперь найдем неизвестные `x1 = d1/d = -2, x2 = d2/d = 2`.

Замечание. Вот одно из немногих преимуществ правила Крамера – неизвестные можно находит по одному, независимо друг от друга (а не все разом, как в методе Гаусса).

Еще один способ формирования матриц `M1`, `M2`. Сначала создадим копии матрицы `M`

```
>> M1 = M; M2 = M;
```

Затем выполним команды замены столбца матрицы-копии на столбец свободных членов

```
>> M1(:, 1) = b

M1 =

     3     4     1     2

     6     5     3     5

     8     8     1     5

     8     5     3     7

>> M2(:, 2) = b

M2 =

     3     3     1     2

     3     6     3     5

     6     8     1     5

     3     8     3     7
```

Переформатирование матрицы – функция `reshape`. Строится новая матрица, заполненная теми же числами, из которых состоит данная матрица, но с другим (заданным) числом строк и столбцов. Чтение данной матрицы и заполнение новой происходит *по столбцам* (наследие Фортрана).

```
>> mtr_1x6 = [1 2 3 4 5 6]

mtr_1x6 =

     1     2     3     4     5     6

>> mtr_2x3 = reshape(mtr_1x6, 2, 3)
```

```
mtr_2x3 =

     1     3     5

     2     4     6

>> mtr_3x2 = reshape(mtr_1x6, 3, 2)

mtr_3x2 =

     1     4

     2     5

     3     6

>> mtr_6x1 = reshape(mtr_1x6, 6, 1)

mtr_6x1 =

     1

     2

     3

     4

     5

     6
```

Общее число элементов новой матрицы должно быть равно числу элементов старой, иначе выдается сообщение об ошибке

```
>> mtr_3x3 = reshape(mtr_1x6, 3, 3)

??? Error using ==> reshape

To RESHAPE the number of elements must not change.
```

Сортировка (упорядочение) матриц (главным образом, строк и столбцов). Введем вектор

```
>> r = [3 -1 2 5]
```

```
r =

        3            -1             2             5
```

и выполним команду

```
>> sort_r = sort(r)
```

– получим отсортированный (упорядоченный) вектор

```
sort_r =

      -1             2             3             5
```

У команды `sort` имеются и другие возможности, с ними можно ознакомиться, обратившись к файлу помощи (`help sort`).

Суммирование элементов матрицы. Для того же вектора `r` выполним команду

```
>> sum_r = sum(r)
```

– получим сумму компонент вектора

```
sum_r =

        9
```

Более интересный результат получается с помощью команды

```
>> cumsum_r = cumsum(r)

cumsum_r =

        3             2             4             9
```

– это ***накопленные суммы*** компонент вектора `r`:

`cumsum_r(1)= r(1), cumsum_r(2)= r(1)+r(2),`

`cumsum_r(3)= r(1)+r(2)+r(3), cumsum_r(4)= r(1)+r(2)+r(3)+r(4).`

Если бы все компоненты вектора `r` были положительными, вектор `cumsum_r` был бы отсортированным (упорядоченным).

1.4. Стандартные и специальные матрицы MATLABa

Матрица, заполненная нулями

```
>> Z23 = zeros(2, 3)

Z23 =

     0     0     0

     0     0     0

>> Z22 = zeros(2)

Z22 =

     0     0

     0     0
```

Матрица, заполненная единицами

```
>> On23 = ones(2, 3)

On23 =

     1     1     1

     1     1     1

>> On22 = ones(2)

On22 =

     1     1

     1     1
```

Единичная матрица

```
>> E23 = eye(2, 3)
```

```
E23 =

     1     0     0

     0     1     0

>> E22 = eye(2)

E22 =

     1     0

     0     1
```

Вектор, заполненный ***арифметической прогрессией***

```
>> 2:3:10

ans =

     2     5     8
```

Матрица Вандермонда. Ее столбцы составлены из степеней чисел, содержащихся в заданном векторе `c` (столбце или строке): 1-й столбец содержит (`n-1`)-е степени, 2-й столбец –

(`n-2`)-е степени, …, (`n-1`)-й столбец – 1-е степени, `n`-й столбец – нулевые степени (т.е. последний столбец заполнен единицами).

```
>> c = [2 3 5]

>> V = vander(c)

V =

     4     2     1

     9     3     1

    25     5     1
```

Матрица Гильберта. Ее элемент `H(i,j) = 1/(i + j - 1)` (эта матрица лучше видна в формате рациональных чисел

```
>> H4 = hilb(4)
```

```
H4 =

     1          1/2          1/3          1/4

    1/2         1/3          1/4          1/5

    1/3         1/4          1/5          1/6

    1/4         1/5          1/6          1/7
```

Для матриц Гильберта характерна очень плохая обусловленность (см. раздел 4).

Обратная матрица к матрице Гильберта может быть вычислена аналитически, она является целочисленной. Получить ее можно с помощью функции `invhilb`

```
>> invhilb(4)

ans =

          16        -120         240        -140

        -120        1200       -2700        1680

         240       -2700        6480       -4200

        -140        1680       -4200        2800

>> hilb(4)*invhilb(4)

ans =

    1.0000         0         0         0

         0    1.0000         0         0

         0         0    1.0000   -0.0000

         0         0         0    1.0000
```

Матрица Фробениуса, или ***присоединенная*** матрица (в английской терминологии `companion matrix`), определяется коэффициентами ее характеристического многочлена. Для многочлена

$$s_0\lambda^n + s_1\lambda^{n-1} + \ldots + s_{n-1}\lambda + s_n$$

строится матрица

$$C=\begin{bmatrix} -s_1/s_0 & -s_2/s_0 & \cdots & -s_{n-1}/s_0 & -s_n/s_0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 1 & 0 \end{bmatrix}.$$

Замечание. В современной вычислительной алгебре корни произвольного многочлена ищутся как собственные числа его присоединенной матрицы (в MATLABe – функция `roots`).

Пример

```
>> C = compan([1 2 4 -5])

C =

    -2    -4     5

     1     0     0

     0     1     0
```

Матрица Паскаля – симметричная положительно определенная матрица, заполненная биномиальными коэффициентами

```
>> pascal(4)

ans =

     1     1     1     1

     1     2     3     4

     1     3     6    10

     1     4    10    20
```

1.5. Случайные матрицы

Существует направление, называемое *методами Монте-Карло*, которое занимается использованием случайных чисел для решения различных математических задач: интерполя-

ции, вычисления интегралов, решения дифференциальных и интегральных уравнений, решения систем линейных уравнений, поиска экстремума, моделирования процессов и т.д.

На ранних этапах для получения случайных чисел использовались заранее составленные таблицы и физические датчики. Очевидным недостатком таблиц является их ограниченный объем, а использование физических датчиков наталкивается на сложность реализации, медлительность датчиков, их «капризность» и невоспроизводимость полученных результатов.

Поэтому вместо «чисто случайных» чисел стали использовать псевдослучайные числа, генерируемых непосредственно на ЭВМ. Случайность при таком подходе заменяется *непредсказуемостью* для неосведомленного пользователя: наблюдая последовательность псевдослучайных чисел, он не в состоянии предсказать, каким будет следующий член этой последовательности, хотя на самом деле все они вычисляются по некоторому алгоритму.

В данной лабораторной работе псевдослучайные числа используются для получения практически неограниченного количества различных матриц, которые являются материалом для решения тех или иных математических задач.

В MATLABe для генерации псевдослучайных чисел, равномерно распределенных в интервале `(0.0, 1.0)`, используется функция `rand`, имеющая несколько форм обращения. Повторное обращение к ней (в любой форме, в том числе неявное) порождает *всякий раз другие результаты*. Об их воспроизводимости см. ниже.

Функция `rand` генерирует ***плавающие числа*** в замкнутом интервале $[2^{-53},\ 1-2^{-53}]$.

Любая последовательность, генерируемая алгоритмически, является периодической. Однако период последовательности, создаваемой функцией `rand`, невообразимо велик: может быть сгенерировано более $2^{1492} \approx 10^{450}$ чисел прежде, чем начнутся повторения.

`rand` (без аргументов) – ***случайное число из интервала*** `(0.0, 1.0)` (для краткости будем опускать префикс «псевдо-»),

`rand(n)` – ***случайная квадратная матрица*** **n×n.**

`rand(m, n)` или `rand([m, n])` – ***случайная матрица*** **m×n.**

Если необходимы ***случайные числа из произвольного интервала*** `(a, b)`, надо выполнить преобразование `a+(b-a)*rand` (при любой форме обращения к функции `rand`).

Конкретная последовательность, сгенерированная функцией `rand`, определяется скрытой переменной `state`. Доступ к ней можно получить, выполнив команду `s=rand('state')`, которая порождает `35`-компонентный вектор текущего состояния генератора.

Команда `rand('state', s)` восстанавливает состояние `s`.

Команда `rand('state', 0)` восстанавливает то состояние генератора, которое он имеет при запуске MATLABa.

Команда `rand('state', j)` устанавливает генератор в `j`-тое состояние, считая от начального.

Команда `rand('state', sum(100*clock))` устанавливает генератор в состояние, определяемое текущим моментом времени (функция `clock` выдает вектор из 6 компонент в формате

```
[year month day hour minute seconds],
```

где первые 5 чисел целые, 6-е – с плавающей точкой). Случайные числа, сгенерированные после такой начальной установки, являются практически невоспроизводимыми.

Среди функций, доступ к которым осуществляется через системную функцию `gallery`, есть и случайные. Так, обращение

```
A = gallery('rando', n, k)
```

генерирует ***случайную квадратную матрицу порядка*** **`n`**, ***все элементы которой суть*** `0`, `1` и `-1`. Параметр `k` является управляющим: при `k=1` `A(i, j)=0` или `1`, при `k=2` `A(i, j)=-1` или `1`, при `k=3` `A(i, j)=-1`, `0` или `1`, причем все значения являются равновероятными.

Пример. Выполним команды

```
A1 = gallery('rando', 3, 1),

A2 = gallery('rando', 3, 2),

A3 = gallery('rando', 3, 3)
```

и получим

```
A1 = [0  0  1   A2 = [-1 -1 -1   A3 = [-1  0 -1
      0  1  1          1  1  1          1  1  0
      1  1  0]         1 -1 -1]         1 -1  0].
```

Кроме матриц, могут генерироваться и другие случайные объекты. Можно, например, сгенерировать ***случайную перестановку*** чисел

```
>> p = randperm(6)

p =

     2     4     3     6     5     1
```

При повторном обращении будут генерироваться другие случайные перестановки

```
>> p = randperm(6)

p =

     2     4     1     5     6     3

>> p = randperm(6)

p =

     3     4     2     6     1     5
```

2. Системы линейных уравнений

Систему линейных уравнений

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x = b_1, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n = b_m, \end{cases}$$

кратко записывают в виде `Ax = b`. Основными характеристиками, от которых зависит существование и единственность решения системы, являются

`n` – число неизвестных (число столбцов матрицы `A`),

`r` – ранг матрицы коэффициентов: `r = rank(A)`,

R – ранг расширенной матрицы: R = rank([A b]).

Имеют место следующие положения:

1. Система совместна (имеет решение) тогда и только тогда, когда r = R, в случае r < R система несовместна (не имеет решения).

2. Решение системы единственно тогда и только тогда, когда r = R = n; в случае r = R < n система является неопределенной (имеет бесконечно много решений).

Очевидно, что число уравнений (число строк матрицы) m ≥ R.

2.1. Крамеровская система

Простейшим и наиболее важным с точки зрения приложений случаем является так называемая крамеровская система Ax = b, у которой r = R = n = m, det(A) ≠ 0. Единственное решение такой системы задается формулой $x = A^{-1}b$ («метод обратной матрицы»), что в обозначениях MATLAB записывается x = inv(A)*b.

Здесь b – матрица, содержащая не обязательно только один столбец, т. е. одной матричной операцией можно решить сразу несколько систем линейных уравнений с одной и той же матрицей коэффициентов, но с различными наборами правых частей (этот случай часто встречается в приложениях).

Пример

```
A =

     3     4     1     2

     3     5     3     5

     6     8     1     5

     3     5     3     7
```

```
b =

     3

     6

     8

     8
```

Эта матрица и столбец свободных членов рассматривались в разделе 1.2, где соответствующая система линейных уравнений решалась (частично) с помощью правила Крамера.

Решим эту систему методом обратной матрицы, обратную матрицу найдем с помощью функции `inv`

```
>> format rational

>> iA = inv(A)

iA =

    19/3     -25/6      -7/3       17/6

     -5        7/2         2       -5/2

      2       -1/2        -1        1/2

      0       -1/2         0        1/2

>> x = iA*b

x =

    -2

     2

    -1

     1
```

Альтернативная запись этого метода – левое матричное деление `x=A\b` (результат совпадает).

2.2. Несовместная система

Псевдорешением системы `Ax = b` называется такой вектор `x`, который при подстановке в систему дает *вектор невязки* `r = Ax − b`, минимальный по евклидовой норме (см. раздел 3). Для совместной системы псевдорешение является обычным решением (вектор невязки нулевой).

Роль, которую при решении крамеровской системы играет обратная матрица, в случае несовместной системы переходит к *псевдообратной* матрице. Для матрицы `A` псевдообратная A^+ имеет такие же размеры, как транспонированная матрица `A'`, ранги матриц `A` и A^+ одинаковы. Псевдообратная матрица удовлетворяет уравнениям $AA^+A = A$ и $A^+A*A^+ = A^+$, причем матрицы AA^+ и A^+A – симметричные (в комплексном случае эрмитовы). Для квадратной невырожденной матрицы `A` псевдообратная совпадает с обратной ($A^+ = A^{-1}$). В MATLABe матрица, псевдообратная к `A`, создается командой `pinv(A)`, псевдорешение несовместной системы уравнений `Ax = b` выражается формулой `x = pinv(A)*b` (альтернативная команда `A\b`).

```
>> A = [2 3; 3 17; 11 17; 13 20]

A =

     2     3

     3    17

    11    17

    13    20

>> b = [0 0 1 3]'

b =

     0

     0

     1

     3
```

```
>> rank(A)

ans =

     2

>> rank([A b])

ans =

     3
```

Согласно теореме Кронекера-Капелли система уравнений `Ax=b` несовместна. Найдем псевдообратную матрицу и псевдорешение

```
>> piA = pinv(A)

piA =

    0.0103   -0.1245    0.0508    0.0611

   -0.0022    0.0808   -0.0087   -0.0109

>> x = piA*b

x =

    0.2340

   -0.0415
```

Тот же результат можно получить, выполнив команду `x = A\b`.

3. Нормы векторов и матриц

Во многих задачах, связанных с линейными пространствами, бывает необходимо *сравнивать* векторы. Например, иметь возможность сказать, что один вектор мал по сравнению с другим, или первый вектор ближе ко второму, чем к третьему и т.п. В таких случаях вводят числовую характеристику – *норму вектора*. Имеется несколько вариантов нормы. В MATLABe она вычисляется функцией `norm`. Обращение `norm(a,p)` возвращает величину $\|a\|_p = \sqrt[p]{|a_1|^p + |a_2|^p + \cdots + |a_n|^p}$, которая называется `p`-*нормой* вектора или *нормой*

Гельдера. Параметр p может принимать любые положительные значения, включая Inf, которое в MATLABe изображает +∞.

Более конкретно:

$\|a\|_1 = |a_1| + |a_2| + \cdots + |a_n|$ называется *манхеттенской* нормой.

$\|a\|_2 = \sqrt{|a_1|^2 + |a_2|^2 + \cdots + |a_n|^2}$ называется *евклидовой* нормой.

$\|a\|_\infty = \max(|a_1|, |a_2|, \cdots, |a_n|)$ называется *чебышевской* нормой.

По умолчанию p=2, команда `norm(a)` возвращает $\|a\|_2$.

Понятие нормы распространяется на матрицы. В общем случае p-*норма* матрицы $\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$, *фробениусова* норма $\|A\|_F = \sqrt{\sum_{i,j} |a_{ij}|^2}$.

Норма $\|A\|_2$ называется *спектральной*, она равна максимальному собственному числу матрицы $A^T \cdot A$.

`norm(A, p)` возвращает p-норму матрицы, причем параметр p может принимать любые положительные значения, включая Inf, а также строковое значение `'fro'`, соответствующее фробениусовой норме.

В конечномерном пространстве все перечисленные нормы эквивалентны в том смысле, что отношение двух норм для любого вектора (матрицы) ограничено сверху и снизу некоторыми константами, не зависящими от вектора (матрицы).

Например, $\forall a \left(1 \le \frac{\text{norm(a,2)}}{\text{norm(a,1)}} \le \sqrt{n} \right)$, где n – размерность пространства.

Поэтому в большинстве случаев выбор конкретного варианта нормы не играет большой роли. Чаще всего используется евклидова норма векторов, ей соответствует спектральная норма матриц.

4. Обусловленность матриц

Метод Гаусса является точным настолько, насколько точно выполняются входящие в него арифметические операции. Однако в любой программной системе арифметические

операции в большинстве случаев выполняются неточно, поэтому решение систем линейных уравнений на самом деле является приближенным.

Оценить погрешность этого приближения, основываясь на погрешностях выполнения отдельных арифметических операций, весьма трудно из-за большого их числа (для нахождения обратной матрицы число операций оценивается $O(n^3)$, где n – порядок матрицы).

Проще выяснить, как влияют на решение системы линейных уравнений ее *возмущение*, т. е. небольшие изменения в правых частях и в коэффициентах матрицы. Если при этом решение меняется также незначительно, можно с некоторой уверенностью считать, что и погрешности округления не очень повлияют на окончательное решение.

Основную роль в этом рассмотрении играет норма вектора $\|x\|$ (имеется в виду любая p-норма). Умножение вектора x на матрицу A может изменить его норму. Область возможных изменений определяется двумя числами

$$M = \max_{\|x\|=1} \|Ax\|, \qquad m = \min_{\|x\|=1} \|Ax\|.$$

Их отношение $\mathrm{cond}(A) = \frac{M}{m}$ называется ***числом обусловленности матрицы*** A. Если оно очень велико, матрица называется ***плохо обусловленной***, решение системы с такой матрицей может оказаться неустойчивым относительно возмущений.

В MATLABe число обусловленности можно найти с помощью функции `cond(A, p)` (по умолчанию `p = 2`). Для вырожденной матрицы по определению `cond(A, p) = Inf`.

Пример

Матрицы Гильберта – известный пример плохо обусловленных матриц:

```
>> cond(hilb(4))
ans =
  1.5514e+004
>> cond(hilb(10))
ans =
  1.6025e+013
```

Нахождение числа обусловленности – довольно сложная задача, во многих случаях достаточно знать его нижнюю оценку (поскольку чем больше число обусловленности, тем хуже матрица, нижняя оценка носит пессимистический характер).

Нижнюю оценку числа обусловленности `cond(A, 1)` (по манхеттенской норме) можно найти обращением `condest(A)`.

Рассмотрим систему уравнений `Ax = b` и возмущенную систему, полученную изменением правой части `b` на величину Δb, решение `x` при этом также меняется на некоторую величину Δx. Для возмущенных векторов имеем матричное равенство $A(x + \Delta x) = b + \Delta b$, Вычтя из него матричное равенство $Ax = b$, получим $A(\Delta x) = \Delta b$.

Из определений `M` и `m` следуют оценки

$$m \cdot \|\Delta x\| \le \|\Delta b\| \text{ и } \|b\| \le M \cdot \|x\|.$$

Введем *относительные изменения* нормы правой части $\delta b = \frac{\|\Delta b\|}{\|b\|}$ и нормы решения $\delta x = \frac{\|\Delta x\|}{\|x\|}$, из приведенных оценок получим при $m \neq 0$

$$\delta x \le cond(A) \cdot \delta b,$$

т. е. относительные изменения в правой части могут повлечь относительные изменения в решении, в худшем случае большие в `cond(A)` раз. То же самое справедливо для изменений в коэффициентах матрицы.

Из определения числа обусловленности видно, что $cond(A) \ge 1$. Нижняя оценка достигается, например, для ортогональных (унитарных) матриц. Чем *больше* величина `cond(A)`, тем *хуже* обусловлена матрица; значения порядка десятков или даже сотен считаются неплохими.

Пример

$$\begin{aligned} x_1 + 0.99x_2 &= 1.99 \\ 0.99x_1 + 0.98x_2 &= 1.97 \end{aligned}$$

Решение системы с помощью команды `x = A\b` дает $x_1 = 1.0000$, $x_2 = 1.0000$. Легко видеть, что этот результат является точным.

Рассмотрим возмущенную систему с измененными свободными членами

$$x_1 + 0.99x_2 = 1.989903$$
$$0.99x_1 + 0.98x_2 = 1.970106$$

Решение этой системы с помощью той же команды дает

$x_1 = 3.0000,\ x_2 = -1.0203$ (приближенно).

«Невооруженным глазом» видно, что решение изменилось очень сильно. Рассмотрим вектор приращений свободных членов

$\Delta b = [-1.97\cdot10^{-4},\ 1.06\cdot10^{-4}]'$.

Его евклидова норма $\|\Delta b\| = \mathrm{norm}(\Delta b) = 1.4368\cdot10^{-4}$. Относительное приращение нормы свободных членов $\delta b = \dfrac{\|\Delta b\|}{\|b\|} = 5.1312\cdot10^{-5}$ (здесь $\|b\| = \mathrm{norm}(b) = 2.8002$). Видим, что относительное приращение вектора свободных членов весьма мало.

Найдем вектор приращений решения $\Delta x = [2.0000,\ -2.0203]'$ и его норму $\|\Delta x\| = \mathrm{norm}(\Delta x) = 2.8428$.

Относительное приращение вектора решения $\delta x = \dfrac{\|\Delta x\|}{\|x\|} = 2.0102$ *не мало* (здесь $\|x\| = \mathrm{norm}(x) = 1.4142$). Отношение $\dfrac{\delta x}{\delta b} = 3.9175\cdot10^4$ – почти 40 тысяч!

Такой плохой результат объясняется не малостью определителя матрица A, хотя он и в самом деле невелик: $\det(A) = -1.0000\cdot10^{-4}$, но его легко сделать равным -1, умножив на 100 оба уравнения, входящих в систему, при этом решение x и отношение $\dfrac{\delta x}{\delta b}$ не изменятся.

Гораздо важнее плохая обусловленность матрицы A. Ее число обусловленности весьма велико: $\mathrm{cond}(A) = 3.9206\cdot10^4$, что лишь ненамного превышает найденное выше значение $\dfrac{\delta x}{\delta b} = 3.9175\cdot10^4$.

В методе Гаусса (с выбором главного элемента по столбцу или по всей матрице) ошибки в процессе решения эквивалентны малым возмущениям во входных данных. Справедлива следующая приближенная формула

$\delta x \approx \varepsilon \cdot \mathrm{cond}(A)$,

где $\varepsilon = 2.2204 \cdot 10^{-16}$ – машинная точность, или «машинный эпсилон», – расстояние между числом 1 и следующим за ним числом с плавающей запятой двойной точности. В MATLABe машинный эпсилон создается командой `eps`.

5. Задания на лабораторную работу

Конкретные варианты заданий формируются с использованием датчика псевдослучайных чисел `rand`, встроенного в MATLAB. Для создания различных вариантов заданий необходима настройка датчика, которая выполняется командой `rand('state', pin)`, где `pin` – *персональный идентификатор* студента, целое число, состоящее из двух частей: номер группы и порядковый номер студента в группе, например `820221`.

Размер данных (матриц и векторов) вначале надо взять небольшой (5–8), чтобы видеть результаты вычислений на мониторе и вывести на печать; формат представления чисел – `short`, в работе № 3 матрицу Гильберта (только ее!) лучше выводить в формате `rational`.

В дальнейшем размеры данных следует увеличивать до 20, 50, 100, но вывод векторов и матриц надо заблокировать (точка с запятой в конце командных строк), выводить *только скалярные результаты* (нормы невязок и возмущений, числа обусловленности).

Вместо формирования случайной матрицы и случайной правой части системы уравнений можно ввести эти данные с клавиатуры или из предварительно подготовленного файла, либо воспользоваться стандартными матрицами MATLABa.

Возможно программное конструирование специальных хорошо обусловленных и плохо обусловленных матриц. Некоторые способы этого описаны в книге С. А. Белов, Н. Ю. Золотых, Численные методы линейной алгебры, Н. Новгород, 2005, стр. 244–245. (Случайные матрицы, как правило, являются хорошо обусловленными.)

Плохо обусловлены, например, матрицы Гильберта начиная с 7–8 порядка. Матрица Гильберта порядка `n` создается командой `H = hilb(n)`.

Плохо обусловленная случайная матрицы создается командой

```
>> A = rnd_bad_matrix(n,c),
```

где `n` – порядок матрицы,

`c` – «заказанное» число обусловленности.

`A` – построенная матрица, `cond(A)` $\approx$ `c`.

5.1. Влияние возмущения матрицы и правой части на решение системы уравнений

Постановка задачи

Целью работы является изучение влияния малых изменений (*возмущений*) матрицы и/или правой части системы линейных уравнений на решение системы.

Последовательность шагов

1. Сформировать случайную квадратную матрицу `A`.
2. Сформировать случайную правую часть `b`.
3. Найти решение системы `A*x = b` с помощью левого матричного деления `x = A\b`.
4. Найти невязку `r = A*x - b` и ее норму `nr = norm(r)`.
5. Сформировать случайное возмущение правой части `db` (малое по сравнению с первоначальной правой частью), найти абсолютную и относительную нормы возмущения `norm(db)` и `reldb = norm(db)/norm(b)`, найти возмущенную правую часть `b_ = b + db`.
6. Найти возмущенное решение `x_ = A\b_` и возмущение решения `dx = x_ - x`, найти абсолютную и относительную нормы возмущения решения `norm(dx)` и `reldx = norm(dx)/norm(x)`.
7. Найти отношение норм возмущения `relnorm = reldx /reldb`.
8. Найти число обусловленности матрицы `cA = cond(A)`, сравнить его с отношением норм `relnorm`.

В отчет должны входить 2–3 примера небольших размеров (7–8) с полным выводом матриц, правых частей, решений и т. д. (все пункты задания).

В отчет должны также входить 2–3 примера больших размеров (20–100) с выводом только скалярных результатов (норма невязки, нормы возмущения правой части, нормы возмущений решения, отношение норм, число обусловленности).

5.2. Сравнение точного и приближенного решения системы уравнений

Постановка задачи

Целью работы является сравнение точного решения системы линейных уравнений, заданного исполнителем работы, и полученного обычным путем приближенного решения.

Последовательность шагов

1. Сформировать случайную квадратную матрицу `A`.
2. Задать некоторое решение x_0 (случайное или ввести с клавиатуры). Решение x_0 считается точным.
3. Вычислить правую часть, соответствующую точному решению `b = A*x0`.
4. Найти приближенное решение `x` системы `A*x = b` с помощью левого матричного деления `x = A\b`.
5. Найти невязку `r = A*x - b` и ее норму `nr = norm(r)`.
6. Найти ошибку приближенного решения `dx = x - x0` и относительную ошибку `relerr = norm(dx)/norm(x0)`.
7. Сравнить `relerr` с произведением «машинного эпсилона» $\varepsilon = 2.2204 \cdot 10^{-16}$ на число обусловленности матрицы `cA = cond(A)` (см. стр. 35, «машинный эпсилон» создается командой `eps`).

В отчет должны входить 2–3 примера небольших размеров (7–8) с полным выводом матриц, правых частей, решений и т. д. (все пункты задания).

В отчет должны также входить 2–3 примера больших размеров (20–100) с выводом только скалярных результатов (норма невязки, норма ошибки и относительная ошибка, число обусловленности и его произведение на машинный эпсилон).

5.3. Системы уравнений с плохо обусловленной матрицей

Постановка задачи

Целью работы является изучение поведения решения системы линейных уравнений с плохо обусловленной матрицей при малых изменениях правой части системы.

5.3.1. Система с заданной матрицей

Последовательность шагов

1. Ввести с клавиатуры матрицу `A` и правую часть `b` плохо обусловленной системы уравнений (отличается от примера в «4. Обусловленность матриц» множителем 100): `A=[100 99;99 98], b=[199;197].`
2. Найти решение системы `x = A\b`.
3. Найти невязку `r = A*x - b` и норму невязки `nr = norm(r)`.

4. Ввести возмущение правой части `db = [-0.0097;0.0106]`, сформировать возмущенную правую часть `b_ = b + db`, найти абсолютную и относительную нормы возмущения `norm(db)` и `reldb = norm(db)/norm(b)`.
5. Найти решение возмущенной системы `x_ = A\b_`, невязку для найденного решения `r_ = A*x_ - b_` и норму невязки `nr_ = norm(r_)`.
6. Найти возмущение решения `dx = x_ - x`, абсолютную и относительную нормы возмущения решения `norm(dx)` и `reldx = norm(dx)/norm(x)`.
7. Найти отношение норм возмущения `relnorm = reldx /reldb`.
8. Найти число обусловленности матрицы `cA = cond(A)`, сравнить его с отношением норм `relnorm`.
9. Найти собственные числа матрицы, их отношение равно (по модулю) числу обусловленности

5.3.2. Система с матрицей Гильберта

Последовательность шагов

1. Построить плохо обусловленную матрицу Гильберта `H` порядка `n = 7`. Для вывода матрицы предварительно задать `format rat`.
2. Задать правую часть `b` в виде столбца из единиц.
3. Найти численное решение системы `x = H\b`.
4. Найти невязку `r = H*x - b` и норму невязки `nr = norm(r)`.
5. Для матрицы Гильберта `H` аналитически построить обратную матрицу `invH = invhilb(n)`.
6. Найти аналитическое решение системы `xA = invH*b`, невязку `rA = H*xA - b` и норму невязки `nrA = norm(rA)`.
7. Найти разность численного и аналитического решения `difA = x - xA`, норму разности `ndifA = norm(difA)` и относительную норму разности `reldifA = ndifA/norm(x)`
8. Ввести возмущение правой части `db`, найти абсолютную норму возмущения `norm(db)` и относительную норму `reldb = norm(db)/norm(b)`, сформировать возмущенную правую часть `b_ = b + db`.
9. Найти решение возмущенной системы `x_ = H\b_`, невязку для найденного решения `r_ = H*x_ - b_` и норму невязки `nr_ = norm(r_)`.

10. Найти возмущение решения `dx = x_ - x`, абсолютную и относительную нормы возмущения решения `norm(dx)` и `reldx = norm(dx)/norm(x)`.
11. Найти отношение норм возмущения `relnorm = reldx /reldb`.
12. Найти число обусловленности матрицы Гильберта `cH = cond(H)`, сравнить его с отношением норм `relnorm`..

5.3.3. Система со случайной плохо обусловленной матрицей

Последовательность шагов

1. Построить случайную плохо обусловленную матрицу `A` порядка `n = 8` с ожидаемым числом обусловленности `c = 10^5` (`A = rnd_bad_matrix(n,c)`).
2. Сформировать случайную правую часть `b`.
3. Найти решение системы `A*x = b` с помощью левого матричного деления `x = A\b`.
4. Найти невязку `r = A*x - b` и ее норму `nr = norm(r)`.
5. Сформировать случайное возмущение правой части `db` (малое по сравнению с первоначальной правой частью), найти абсолютную и относительную нормы возмущения `norm(db)` и `reldb = norm(db)/norm(b)`, найти возмущенную правую часть `b_ = b + db`.
6. Найти возмущенное решение `x_ = A\b_` и возмущение решения `dx = x_ - x`, найти абсолютную норму возмущения решения `norm(dx)` и и относительную норму `reldx = norm(dx)/norm(x)`.
7. Найти отношение норм возмущения `relnorm = reldx /reldb`.
8. Найти число обусловленности матрицы `cA = cond(A)`, сравнить его с отношением норм `relnorm`.

В отчет должны входить 2–3 примера небольших размеров (7–8) с полным выводом матриц, правых частей, решений и т. д. (все пункты задания).

В отчет должны также входить 2–3 примера больших размеров (20–100) с выводом только скалярных результатов (число обусловленности матрицы, норма невязки, норма возмущения правой части, норма возмущения решения, отношение норм).

5.4. Полиномиальная аппроксимация

Постановка задачи

Пусть заданы точки числовой оси $x_1 < x_2 < \ldots < x_n$, называемые *узлами*, и значения $y_1, y_2, \ldots, y_n$, где каждое y_i – вещественное число, соответствующее узлу x_i. Требуется построить полином $p(x)$, аппроксимирующий (приближающий) эти данные, т.е. такой, что во всех узлах $p(x_i) \approx y_i$. Если равенство является точным, это *интерполяционный* полином. Он всегда существует и единственен, его степень не превосходит n–1.

Для аппроксимации обычно используются полиномы меньшей степени. Неизвестные коэффициенты $p_1, p_2, \ldots, p_{k+1}$ полинома заданной степени k, для которого погрешность аппроксимации (норма невязки) минимальна, ищутся как *псевдорешение* системы уравнений (почти всегда несовместной). Ее матрицу будем называть *матрицей степеней*. Система уравнений

$$\begin{cases} p_1 + p_2 x_1 + p_3 x_1^2 + \ldots + p_{k+1} x_1^k = y_1, \\ p_1 + a_2 x_2 + p_3 x_2^2 + \ldots + p_{k+1} x_2^k = y_2, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ p_1 + p_2 x_n + p_3 x_n^2 + \ldots + p_{k+1} x_n^k = y_n, \end{cases}$$

матрица степеней x0_k $= \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & \cdots & x_2^k \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & \cdots & x_n^k \end{bmatrix}$.

Левая часть системы представляет собой произведение `x0_k*[`$p_1, p_2, \ldots, p_{k+1}$`]'`.

Последовательность шагов

1. Сформировать случайный вектор-столбец `x` с положительными компонентами.
2. Сформировать отсортированный (упорядоченный) вектор-столбец с помощью одной из команд `x = sort(x)` или `x = cumsum(x)` (абсциссы точек).
3. Сформировать случайный вектор-столбец `y` (ординаты точек).
4. Сформировать векторы-столбцы степеней абсцисс (`x0 = [1 1 1 ...]'`, `x1 = x, x2 = x1.*x, x3 = x2.*x` и т. д.)
5. Сформировать матрицу степеней абсцисс для квадратичной аппроксимации `x0_2 = [x0 x1 x2]`.
6. Найти псевдорешение (коэффициенты полинома 2-й степени) `p2 = x0_2 \ y`.
7. Найти ординаты аппроксимации `y2 = x0_2*p2`.
8. Найти вектор невязки квадратичной аппроксимации `r2 = y2 - y`
9. Найти норму невязки `nr2 = norm(r2)`.

Повторить шаги 6–9 для аппроксимации более высоких степеней, формируя матрицы `x0_3 = [x0_2 x3]` для кубической аппроксимации, `x0_4 = [x0_3 x4]` для аппроксимации 4-й степени и т. д.

Альтернативный способ построения матрицы степеней

1. Построить полную матрицу степеней `v = vander(x)`
2. Переставить столбцы матрицы `v` в обратном порядке `v = fliplr(v)`.

3. Выделить первые три столбца для квадратичной аппроксимации `x0_2 = v(:, 1:3)`. Убедиться в том, что результат не изменился.
4. Для кубической аппроксимации надо выделить первые четыре столбца `x0_3 = v(:, 1:4)` и т. д.

В отчет должны входить 2–3 примера небольших размеров (7–8) с полным выводом векторов `x` и `y`, матриц степеней, псевдорешений и т. д. (все пункты задания). Векторы-столбцы выводить в транспонированном виде, т.е. в виде строк.

Для каждого примера построить график, содержащий аппроксимируемые точки и найденные полиномы.

Построить сводную таблицу норм невязок в зависимости от степени аппроксимации (2, 3, 4 и т.д.).

В отчет должны также входить 2–3 примера больших размеров (20–100) с выводом только норм невязки для аппроксимации различных степеней, построить сводную таблицу норм невязок в зависимости от степени аппроксимации.

Приложение.

Краткая сводка по MATLAB

Краткая сводка по языку MATLAB

Н. Ю. Золотых

05.05.08, 12.05.11, 03.06.11, 05.09.12

Имена переменных, функций и т. п. чувствительны к регистру. Разделителями команд являются: ENTER, «,» или «;». Результат команды, после которой идет ENTER или «,», выдается на экран. Для продолжения команды на следующей строке используется «...».

`format short` переход в режим с «коротким» представлением чисел с плавающей запятой

`format long` «длинный» формат при выводе чисел с плавающей запятой

`format rat` «рациональный» формат при выводе чисел с плавающей запятой

`help имя_функции` справка в командном окне

`doc имя_функции` справка в справочном навигаторе

`edit имя_функции` редактирование функции

`lookfor тема` поиск ключевого слова в описаниях функций

1. Числа

`1 + 2*(3 - 4)/5` вычисление выражения

`a = 1 + 2*(3 - 4)/5` вычисление выражения и присваивание результата переменной `a`

`ans` результат последнего не присвоенного выражения

`x^y` x^y

`pi` число π

`Inf` бесконечность ∞

`NaN` «не-число» (например, как результат `0/0`, `Inf/Inf`)

`eps` машинная точность $2^{-52} = 2.2204 \times 10^{-16}$

`realmax` максимальное число $(2 - 2^{-52}) \times 2^{1023} = 1.7977 \times 10^{308}$

`realmin` минимальное положительное нормализованное число $2^{-1022} = 2.2251 \times 10^{-308}$

`abs(x)` модуль $|x|$

`sign(x)` знак $\operatorname{sign} x$

`sqrt(x)` корень квадратный $\sqrt{x}$

`exp(x)` e^x

`log(x)` $\ln x$

`log2(x)` $\log_2 x$

`log10(x)` $\log_{10} x$

`sin(x)` $\sin x$

`cos(x)` $\cos x$

`tan(x)` $\operatorname{tg} x$

`asin(x)` $\arcsin x$

`acos(x)` $\arccos x$

`atan(x)` $\operatorname{arctg} x$

`floor(x)` «пол» $\lfloor x \rfloor$

`ceil(x)` «потолок» $\lceil x \rceil$

`round(x)` ближайшее целое $\lfloor x \rceil$

`fix(x)` число с отброшенной дробной частью

`gcd(m, n)` НОД(m, n)

`lcm(m, n)` НОК(m, n)

`rem(m, n)` $m - \texttt{fix}(m/n)\, n$

`mod(m, n)` $m - \lfloor m/n \rfloor\, n$

`primes(n)` список простых чисел $\le n$

`isprime(n)` проверка числа на простоту

`factor(n)` разложение на простые множители числа n

`factorial(n)` $n!$

`i`, `j`, `1i`, `1j` мнимая единица

`1+1i`, `1-2i`, `3i` комплексные числа

`complex(a, b)` комплексное число `a+bi`

`real(z)` действительная часть комплексного числа z

`imag(z)` мнимая часть комплексного числа z

`abs(z)` модуль комплексного числа z

`angle(z)` угол (аргумент) комплексного числа z

`z'`, `conj(z)` сопряженное число

2. Векторы и матрицы

`[1, 2, 3]` или `[1 2 3]` вектор–строка $[1, 2, 3]$

`[1; 2; 3]` вектор–столбец $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

`[1, 2; 3, 4]` матрица $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

(Вместо `;` можно нажимать ENTER)

`a(1)` 1-й элемент вектора (строки или столбца) a

`a(end)` последний элемент вектора a

`A(2, 4)` элемент 2-й строки 4-го столбца матрицы A

`A(2, end)` элемент 2-й строки последнего столбца

`A(end, 4)` элемент последней строки 4-го столбца

`A(2, :)` 2-я строка матрицы A

`A(:, 4)` 4-й столбец матрицы A

`A(2, :) = []` удаление 2-й строки матрицы A

`A(:, 4) = []` удаление 4-го столбца матрицы A

`A([1, 2], [2, 5])` матрица из элементов, стоящих на пересечении 1-й и 2-й строк и 2-го и 5 столбцов

`a:b` строка, составленная из чисел от a до b с шагом 1

`a:h:b` строка из чисел от a до b с шагом h

`1:10` строка $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

`1:2:10` строка $[1, 3, 5, 7, 9]$

`size(A)` размеры матрицы A (вектор из двух компонент)

`size(A, 1)` число строк матрицы A

`size(A, 2)` число столбцов матрицы A

`length(a)` длина вектора a

`[a, b]` конкатенация векторов-строк

`[a; b]` конкатенация векторов-столбцов

`[A, B; C, D]` блочная матрица $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$

`repmat(A, m, n)` матрица, полученная тиражированием A в m строках и n столбцах

`flipud(A)` переворачивает матрицу «вверх ногами»; эквивалентно `A(end:-1:1, :)`

`fliplr(A)` переворачивает матрицу «задом наперед»; эквивалентно `A(:, end:-1:1)`

`zeros(n)` нулевая квадратная матрица порядка n

`zeros(m, n)` нулевая прямоугольная матрица размера $m \times n$

`ones(n)`, `ones(m, n)` матрица, заполненная единицами

`eye(n)`, `eye(m, n)` единичная матрица

`rand` случайное число на отрезке $[0, 1]$

`randn` случайное число, распределенное по нормальному закону с мат. ожиданием 0 и среднеквадратическим отклонением 1

`rand(n)`, `randn(n)` случайные квадратные матрицы порядка n

`rand(m, n)`, `randn(m, n)` случайные прямоугольные матрицы размера $m \times n$

`randi(n)` случайное целое число, выбираемое из $1, 2, \ldots, n$

`sort(a)` сортировка элементов массива a по возрастанию

`sort(a, 'descend')` сортировка элементов массива a по убыванию

`A + B` сумма матриц $A + B$

`A - B` разность матриц $A - B$

`a*A` произведение aA числа a на матрицу A

`A*B` матричное произведение AB

`A^n` матричная степень A^n

`A.*B` покомпонентное произведение

`A./B` покомпонентное деление

`A.^B` покомпонентная степень

`A'` сопряженная матрица $\overline{A}^{\mathrm{T}}$

`A.'` транспонированная матрица A^{T}

`A\b` решение с.л.у. $Ax = b$
`A\B` решение матричного уравнения $AX = B$
`A/B` решение матричного уравнения $YA = B$
`inv(A)`, `A^(-1)` обратная матрица A^{-1}
`det(A)` определитель матрицы
`rref(A)` приведенный ступенчатый вид матрицы
`linspace(a, b)` вектор из 100 равномерно отстоящих узлов от a до b
`linspace(a, b, n)` вектор из n равномерно отстоящих узлов от a до b
`logspace(a, b)` эквивалентно `10.^linspace(a, b, 50)`
`logspace(a, b, n)` эквивалентно `10.^linspace(a, b, n)`
`tril(A)` нижнетреугольная часть матрицы
`tril(A, k)` элементы ниже k-й кодиагонали ($k = 0$ — главная диагональ, $k > 0$ — выше главной диагонали, $k < 0$ — ниже главной диагонали)
`triu(A)` верхнетреугольная часть матрицы
`triu(A, k)` элементы выше k-й кодиагонали
`diag(A)` главная диагональ матрицы
`diag(A, k)` k-я кодиагональ матрицы
`diag(d)` формирует диагональную матрицу с элементами из вектора d на диагонали
`diag(d, k)` формирует матрицу с элементами из d на k-й кодиагонали
`blkdiag(D1, D2, D3, ...)` блочно-диагональная матрица
`[X, Y] = meshgrid(x, y)` генерирование решеток:
`[X, Y] = meshgrid([1,2], [11,22,33])` возвращает

$$X = \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}, \quad Y = \begin{bmatrix} 11 & 11 \\ 22 & 22 \\ 33 & 33 \end{bmatrix}$$

Математические функции (`abs`, `sqrt`, `exp`, `log`, `sin` и т. д.), примененные к матрицам, действуют покомпонентно.
См. также раздел 9.

3. Сохранение и загрузка данных

`save имя_файла` сохранение значений всех переменных рабочего пространства в `mat`-файле
`save имя_файла a1 a2 ... an` сохранение значений указанных переменных
`load имя_файла` загрузка данных из файла
`load имя_файла a1 a2 ... an` загрузка указанных переменных
`save имя -ascii` сохранение значения переменной в одноименном текстовом файле
`load имя -ascii` загрузка переменной из текстового файла
`[a1, a2, a3, ...] = textread(имя_файла, формат)` чтение из текстового файла в переменные `a1`, `a2`, `a3`, ...

4. Графические функции

`plot(x, y)` график функции
`plot(x, y, стиль)` график функции с указанием стиля линии:
`c m y r g b w k` цвет линии и маркера
`- -- : -.` стиль линии
`+, o, *, x, s, d, ^, v, >, <, p, h` тип маркера
`xlabel('Text')` подпись к оси Ox
`ylabel('Text')` подпись к оси Oy
`title('Text')` заголовок вверху графика
`clf` очищает текущее графическое окно
`shg` выдвигает текущее графическое окно вперед
`figure` создает новое графическое окно и делает его активным
`figure(n)` делает активным окно с номером n
`hold on` переходит в режим сохранения результатов графического вывода
`hold off` выходит из режима
`hold` меняет режим
`plot(x1, y1, x2, y2, ..., xn, yn)` несколько кривых
`plot(x1, y1, стиль1, ..., xn, yn, стильn)` несколько кривых с указанием их свойств
`plot(x, Y)` если `Y` — матрица, то эквивалентно `plot(x, Y(:, 1), ..., x, Y(:, 2))`
`legend('текст1', 'текст2', ..., 'текстn')` легенда
`xlim([xmin, xmax])` диапазон изменения координаты x
`ylim([ymin, ymax])` диапазон изменения координаты y
`zlim([zmin, zmax])` диапазон изменения координаты z
`axis([xmin, xmax, ymin, ymax, zmin, zmax])` диапазоны
`axis equal` одинаковый масштаб по всем осям
`axis square` оси координат квадратные
`axis on` включает отображение осей
`axis off` выключает отображение осей
`grid on` включает отображение сетки
`grid off` выключает отображение сетки
`logx(...)` аналогично `plot`, но используется логарифмическая шкала по оси Ox
`logy(...)` аналогично `plot`, но используется логарифмическая шкала по оси Oy
`semilog(...)` аналогично `plot`, но используется двойная логарифмическая шкала
`polar(phi, r)` график в полярных координатах
`polar(phi, r, стиль)` график в полярных координатах с указанием стиля
`plot3(x, y, z)` график кривой в пространстве
`plot3(x, y, z, стиль)` график кривой в пространстве с указанием стиля
`plot3(x1, y1, z1, x2, y2, z2, ..., xn, yn, zn)` несколько графиков в пространстве
`plot3(x1, y1, z1, стиль1, ..., xn, yn, zn, стильn)` несколько графиков в пространстве с указанием стиля
`mesh(X, Y, Z)` «проволочная» поверхность
`surf(X, Y, Z)` закрашенная поверхность
`surfl(X, Y, Z)` моделирование освещения
`colormap палитра` задание палитры:
`winter spring summer autumn bone`
`copper hot cool gray pink` и др.
`colorbar` отображение цветовой шкалы
`hidden on` включение режима скрытия невидимых линий
`hidden off` включение режима отображения невидимых линий
`shading faceted` режим отрисовки граней
`shading interp`
`shading flat`
`alpha a` задание коэффициента прозрачности
`view(az, el)` угол обзора (долгота и широта в градусах)
`view(x, y, z)` точка обзора
`contour(X, Y, Z)` линии уровня
`contour(X, Y, Z, n)` n линий уровня
`contour(X, Y, Z, [c1, c2, ..., cn])` линии уровня для заданных значений функции
`contourf(X, Y, Z)` `contourf(X, Y, Z, n)` `contourf(X, Y, Z, [c1, c2, ..., cn])` то же с закрашиванием промежутков между линиями уровня
`camlight headlight` размещение источника света в точке наблюдения
`camlight right` размещение источника света справа сверху от точки наблюдения
`camlight left` размещение источника света слева сверху от точки наблюдения
`camlight` то же, что и `camlight right`
`camlight(az, el)` задание долготы и широты источника света
`axis vis3d`
`ezplot('f(x)', a, b)` график функции $f(x)$ на отрезке $[a, b]$
`ezplot('f(x,y)')` график кривой $f(x, y) = 0$
`ezplot('x(t)', 'y(t)', a, b)` график линии $\mathtt{x} = \mathtt{x}(t)$, $y = y(t)$, $a \le t \le b$
`ezplot3('x(t)', 'y(t)', 'z(t)', a, b)` график линии $\mathtt{x} = \mathtt{x}(t)$, $y = y(t)$, $z = z(t)$, $a \le t \le b$

`ezmesh('f(x, y)', [a, b, c, d])` «проволочная» поверхность $z = f(x, y)$, $a \le x \le b$, $c \le y \le d$
`ezsurf('f(x, y)', [a, b, c, d])` закрашенная поверхность
`ezmesh('x(u, v)', 'y(u, v)', 'z(u, v)', [a, b, c, d])`
`ezsurf('x(u, v)', 'y(u, v)', 'z(u, v)', [a, b, c, d])` поверхность $x = x(u, v)$, $y = y(u, v)$, $z = z(u, v)$, $a \le u \le b$, $c \le v \le d$
`ezsurf('f(x, y)', [a, b, c, d])` закрашенная поверхность
`ezcontour('f(x, y)', [a, b, c, d])` линии уровня
`ezpolar('r(phi)', [a, b])` график $r = r(\mathtt{phi})$, $\mathtt{a} \le \mathtt{phi} \le \mathtt{b}$ (в полярных координатах)

5. Конструкции языка

```
if условие
    команды
end

if условие
    команды
else
    команды
end

if условие
    команды
elseif условие
    команды
...
elseif условие
    команды
end

if условие
    команды
elseif условие
    команды
...
elseif условие
    команды
else условие
    команды
end

for переменная = вектор
    команды
end

for переменная = матрица
    команды
end

for переменная = матрица
    команды
end

while условие
    команды
end

switch выражение
    case значение
        команды
    case {значение1, значение2, значение3, ...}
        команды
    otherwise
        команды
end
```

`break` немедленный выход из цикла `for` или `while`
`continue` немедленный возврат к проверке условия в цикле `for` или `while`

`all(a)` истина, если и только если все элементы вектора `a` ненулевые
`all(A)` если `A` — матрица, то применяет функцию `all` к каждому ее столбцу; на выходе — вектор-строка длины, равной количеству столбцов в `A`
`any(a)` истина, если и только если хотя бы один из элементов вектора `a` ненулевой
`any(A)` если `A` — матрица, то применяет функцию `any` к каждому ее столбцу; на выходе — вектор-строка длины, равной количеству столбцов в `A`
`<`, `>`, `<=`, `>=`, `==`, `~=` (поэлементное) сравнение
`&`, `|`, `~` поэлементные «и», «или», «не»
`&&`, `||` логические «и», «или»

6. Массивы структур, массивы ячеек

`varname.field1`, `varname.field1` поля структуры
`{31, [1, 2], 'Hello'}` массив ячеек
`a{i}` `i`-й элемент массива ячеек
`a(indices)` подмассив (срез) массива ячеек

7. Программы-сценарии и функции пользователя

Программа-сценарий («скрипт») — это набор команд, записанных в текстовом файле (`m`-файле). Такие программы работают с переменными рабочего пространства (теми же, с которыми пользователь работает из командной строки). В отличие от них, программы-функции работают со своими локальными переменными.
`%` начинает строку комментариев

7.1. Функция с подфункциями

```
function [y1, y2, ..., ym] = funcname(x1, x2, ..., xn)

команды

    function [...] = funcname(...)

    команды

    function [...] = funcname(...)

    команды
```

Область видимости локальных переменных, появляющихся в основной функции не распространяется на подфункции. Область видимости локальных переменных, появляющихся в подфункции, ограничивается этой подфункцией.

7.2. Вложенные (nested) функции

```
function [y1, y2, ..., ym] = func(x1, x2, ..., xn)

    команды

    function [...] = func1(...)

        команды

        function [...] = func2(...)

            команды

        end
    end

    function [...] = func3(...)

        команды

    end
```

```
end

function [...] = func4(...)

    команды

end
```

Функции `func1` и `func3` вложены в `func`. Функция `func2` вложена в `func1`.

Область видимости локальных переменных, появляющихся в функции (или вложенной подфункции), распространяется на все вложенные поддфункции.

8. Суммы, произведения и т.п.

`sum(a)` сумма элементов вектора `a`

`sum(A)` сумма элементов каждого столбца матрицы `A`. Возвращается вектор-строка длины, равной количеству столбцов матрицы `A`

`cumsum(a)` «кумулятивная» сумма

`cumsum(A)` «кумулятивная» сумма для каждого столбца матрицы `A`. Возвращается матрица того же размера, что и `A`

`prod(a)` произведение элементов вектора `a`

`prod(A)` произведение элементов каждого столбца матрицы `A`

`cumprod(a)` «кумулятивное» произведение

`cumprod(A)` «кумулятивное» произведение для каждого столбца матрицы `A`

`diff(a)` вектор разностей

`min(a)` минимальное значение

`max(a)` максимальное значение

9. Линейная алгебра

См. также раздел 2.

`norm(a, p)` `p`-норма вектора `a`: $\sqrt[p]{\sum_{j=1}^{n} |a_j|^p}$, где $p \geq 1$.

Возможно значение `p = Inf`

`norm(a)` евклидова норма `norm(a, 2)`

`norm(A, p)` `p`-норма матрицы A. Возможные значения `p = 1`, `2`, `Inf`, `'Fro'`. Последнее соответствует фробениусовой норме.

`norm(A)` спектральная норма `norm(A, 2)` матрицы `A`

`cond(A, p)` число обусловленности матрицы `A` для `p`-нормы

`cond(A)` спектральное число обусловленности `cond(A, 2)`

`condest(A)` верхняя оценка для `cond(A, 1)` (оценщик Хэйджера)

`A\b` решение с.л.у. `Ax = b`; псевдорешение для несовместной с.л.у.

`A\B` решение матричного уравнения `AX = B`

`A/B` решение матричного уравнения `YA = B`

`inv(A)` обратная матрица A^{-1}

`pinv(A)` псевдообратная матрица

`[L, U, P] = lu(A)` LU-разложение матрицы A: $PA = LU$, где P — перестановочная матрица, L — нижнетреугольная с единичной диагональю, U — верхнетреугольная

`[Q, R] = qr(A)` QR-разложение матрицы A: $A = QR$, где Q — унитарная (ортогональная) матрица, R — верхнетреугольная

`B = null(A)` ортонормированный базис пространства решений с.л.у. $Ax = o$

`B = null(A, 'r')` «рациональный» базис пространства решений с.л.у. $Ax = o$

`d = eig(A)` собственные числа матрицы `A`

`[Q, D] = eig(A)` возвращается диагональная матрица D и матрица Q, такие, что $D = Q^{-1}AQ$

`vander(x)` матрица Вандермонда

$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \dots & x_2 & 1 \\ \dots & \dots & \dots & \dots & \dots \\ x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix}$$

`hilb(n)` матрица Гильберта порядка n: $H = (h_{ij})$, где

$$h_{ij} = \frac{1}{i+j-1}$$

`invhilb(n)` матрица, обратная к матрице Гильберта

`compan(s)` матрица Фробениуса

$$\frac{1}{s_1}\begin{bmatrix} -s_2 & -s_3 & \dots & -s_{n-1} & -s_n \\ s_1 & 0 & \dots & 0 & 0 \\ 0 & s_1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & s_1 & 0 \end{bmatrix}$$

10. Интерполяция и аппроксимация данных

`f = polyfit(x, y, n)` возвращает коэффициенты многочлена степени `n`, аппроксимирующего данные

`y = polyval(f, x)` вычисление значения `y` многочлена `f` в точке `x`

`interp1(x, y, xx, 'nearest')` ступенчатая интерполяция

`interp1(x, y, xx)` или `interp1(x, y, xx, 'linear')` кусочно-линейная интерполяция

`interp1(x, y, xx, 'spline')` или `spline(x, y, xx)` кубический сплайн

`interp1(x, y, xx, 'pchip')` или `interp1(x, y, xx, 'cubic')` или `pchip(x, y, xx)` кубический эрмитов интерполянт

`fft(x)` дискретное преобразование Фурье

`ifft(x)` обратное дискретное преобразование Фурье

11. Численное интегрирование

`trapz(x, y)` формула трапеций

`quad(func, a, b)` метод Симпсона

`quad(func, a, b, tol)` метод Симпсона с заданием абсолютной погрешности `tol`

`quadl(func, a, b)` метод Лобатто

`quadl(func, a, b, tol)` метод Лобатто с заданием абсолютной погрешности `tol`

`dblquad(func, a, b, c, d, tol)` двойной интеграл

`triplequad(func, a, b, c, d, e, f, tol)` тройной интеграл

12. Оптимизация и решение систем уравнений

`[x, fval] = fminbnd(func, a, b)` минимизация функции одной переменной на отрезке $[a, b]$; возвращается найденная точка минимума x и значение функции в этой точке

`[x, fval] = fminsearch(f, x0)` минимизация функции многих переменных; $x0$ — начальное приближение

`[x, fval] = fzero(func, x0)` нуль функции; $x0$ — начальное приближение

`[x, fval] = fsolve(func, x0)` решение системы уравнений

13. Обыкновенные дифференциальные уравнения

Задача Коши:

`ode45`, `ode23`, `ode113` — для нежестких задач;

`ode15s`, `ode23s`, `ode23t`, `ode23tb` — для жестких задач

`[t, y] = ode***(func, [t0, T], y0)` решение задачи Коши для системы диф. уравнений на отрезке `[t0, T]`; `y0` — начальное значение

14. Symbolic Math Toolbox

`s = sym('выражение')` создание символьного выражения
`syms a b c real` создание вещественных символьных переменных
`syms a b c unreal` создание комплексных символьных переменных
`syms a b c` то же, что и `syms a b c unreal`
`digits(d)` установить количество значащих цифр результата
`vpa(s)` вычислить символьное выражение
`vpa(s, d)` вычислить символьное выражение с `d` цифрами
`simplify(s)` упрощение символьного выражения
`simple(s)` перебор разных способов упрощения
`expand(s)` раскрытие выражения
`factor(n)` факторизация целого числа
`factor(f)` факторизация многочлена
`subs(s, x, a)` подстановка в s значения $x = a$
`limit(f)` $\lim_{x\to 0} f(x)$
`limit(f, a)` $\lim_{x\to a} f(x)$
`limit(s, y, a)` $\lim_{y\to a} f(y)$
`diff(f)` $f'(x)$
`diff(f, y)` $f'(y)$
`diff(f, y, n)` $f^{(n)}(y)$
`int(f)` $\int f(x)dx$
`int(f, a, b)` $\int_a^b f(x)dx$
`int(f, y)` $\int f(y)dy$
`int(f, y, a, b)` $\int_a^b f(y)dy$
`solve(s)` решение уравнения
`solve(s, x)` решение уравнения относительно указанной неизвестной
`solve(s1, ..., sn)` решение системы уравнений
`solve(s1, ..., sn, x1, ..., xn)` решение системы уравнений относительно указанных неизвестных

```
syms a b c d x y
solve(a*x^2+b*x+c)
solve(a*x^3+b*x^2+c*x+d)
solve('x^2+x*y+y=3','x^2-4*x+3=0')
ans.x, ans.y
solve('x^2+x*y+y=3','x^2-4*x+3=0')
```

`dsolve(eq1, ..., eqn, cond1, ..., condn, t)` решение системы дифференциальных уравнений

```
dsolve('Dy+4*y = exp(-t)')
dsolve('Dy+4*y = exp(-t)', 'y(0) = 1')
dsolve('(Dy)^2 + y^2 = 1')
dsolve('(Dy)^2 + y^2 = 1')
dsolve('Dx = y', 'Dy = -x')
dsolve('D2y=-y')
```

Для матриц с символьными элементами переопределены многие функции линейной алгебры и др.