Краткое руководство по работе с пакетами GNU Octave и Gnuplot

Н.Б. Шамрай

22 февраля 2011 г.

Содержание

1 Язык численной математики GNU Octave

GNU Octave — это свободно распространяемый язык программирования высокого уровня, ориентированный на проведение численных расчетов, и по сути являющийся альтернативой коммерческому пакету MatLab . Octave обладает богатым инструментарием для решения задач линейной алгебры, нахождения корней нелинейных уравнений, решения дифференциальных уравнений, вычисления интегралов, решения линейных и нелинейных оптимизационных задач, построения графиков и т.д. Только некоторые из этих возможностей Octave будут описаны в настоящем руководстве.

1.1 Запуск пакета GNU Octave

Все управление процессом вычислений в пакете GNU Octave осуществляется через командную строку. Также можно заранее подготовить на языке Octave скрипт-файл для последующего его исполнения.

Для того, чтобы запустить пакет GNU Octave необходимо в командной строке исполнить команду `octave`. Программа запустится в текущей консоли, при этом сперва будет выдано длинное сообщение о версии, авторах и прочая информация о пакете, а затем приглашение командной строки к работе с Octave

```
octave:1>
```

Если при запуске Octave дать команду `octave -q`, то начальное сообщение выводиться не будет.

Выйти из Octave можно либо нажав комбинацию клавиш <Ctrl>+<D>, либо набрав в командной строке Octave одну из команд `exit` или `quit`

```
octave:1> exit
```

Командой `exit` также можно закончить выполение любой программы, написанной на языке Octave.

Работая из командной строки Octave каждая введенная команда исполняется сразу после нажатия клавиши <Enter>. Рассмотрим простой пример по созданию матрицы

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

Любую матрицу или вектор с заданными элементами в Octave можно создать путем перечисления этих элементов в квадратных скобках (`[ ]`), разделяя столбцы пробелом или запятой, строки — знаком "точка с запятой"

```
octave:1> A=[1 3 5; 2 4 6]
```

После нажатия клавиши <Enter> на экране появится результат выполнения команды

```
A =

   1   3   5
   2   4   6
```

Теперь в памяти Octave есть матрица A и далее с ней можно производить любые доступные операции, например, транспонировать: операция "штрих" (`'`) после имени матрицы

```
octave:2> A'
```

Результат выполнения автоматически будет сохранен в переменную `ans` и выведен на экран

```
ans =

   1   2
   3   4
   5   6
```

Везде далее после введения команды Octave будем печатать результат ее исполнения.

Можно создать новую матрицу $B = A^T$ при помощи операции присвоения (`=`)

```
octave:3> B=A'
B =

   1   2
   3   4
   5   6
```

Далее можно оперировать уже двумя матрицами A и B.

Как уже было отмечено, работа через командную строку Octave — это один из способов проводить процесс вычислений. Альтернативой является создание скрипт-файла и последующая его обработка пакетом GNU Octave.

Опишем процесс создания матрицы A и B в отдельном файле, например, `matrix.oc` [1], который должен содержать следующие строки:

```
A=[1 3 5; 2 4 6]
B=A'
```

После сохранения файла `matrix.oc` из командной строки консоли можно запустить его на исполнение командой

```
octave matrix.oc
```

Результат работы будет выведен на экран.

В ходе реализации более сложных вычислений, как правило, существуют промежуточные операции, результат выполнения которых выводить нет необходимости. Чтобы отменить вывод следует ставить знак "точка с запятой" после соответствующей команды. Например, если поставить "точку с запятой" после команды создания матрицы A

```
A=[1 3 5; 2 4 6];
B=A'
```

то по мере обработки скрипт-файла `matrix.oc` будет выведена только матрица B.

Код программы может сопровождаться комментариями. Octave игнорирует строки, помеченные знаками "решетка" (`#`) или "процент" (`%`).

[1] Несмотря на то, что расширение файлов для UNIX-систем не является обязательным, для скрипт-файлов Octave будем писать расширение `.oc`

```
A=[1 3 5; 2 4 6];
# Транспонирование матрицы A
B=A'
```

Далее возможности языка покажем на примерах, реализованных через командную строку Octave, однако все описанные ниже команды можно записать в отдельный файл `*.oc` и обработать пакетом как самостоятельную программу.

1.2 Простые вычисления

Средствами языка Octave можно выполнить любые арифметические и алгебраические операции, доступные самому продвинутому калькулятору. Операндами могут быть как вещественные, так и комплексные числа.

Вещественные числа определяются в виде целого значения или конечной десятичной дроби, при этом дробная часть числа отделяется точкой, например `3.65`. Допустима экспонециальная форма записи вещественных чисел, например `1e-4`, что эквивалентно значению 0.0001, или `1e4`, что эквивалентно значению 10000. В вычислениях также можно использовать иррациональные константы $\pi = 3.146...$ команда `pi` и $e = 2.718..$ команда `e`.

Любое комплексное число вводится как формальная сумма x `+ iy`, где x и y — вещественные числа, `i` — мнимая единица, то есть число, удовлетворяющее уравнению $i^2 = -1$.

```
octave:16> c = 1-3e-2i
c =  1.000000 - 0.030000i
```

Операции над комплексными числами осуществляются согласно общих правил и отдельно в настоящем руководстве рассматриваться не будут.

Применяя операции сложения (`+`), вычитания (`-`), умножения (`*`), деления (`/`), возведения в степень (`^`) можно вычислять самые разнообразные математические выражения

```
octave:1> (1+2^3-4)/5
ans =  1
```

В Octave легко вычислить квадратный корень, логарифмические и тригонометрические функции, имеются функции округления, взятия минимума и максимума и т.д.Список некоторых(!) функций приведен в таблице 1. Аргумент x для тригонометрических функций задается в радианах.

Напомним, что

$$\mathrm{sign}(x) = \begin{cases} 1, & \text{если } x > 0; \\ 0, & \text{если } x = 0; \\ -1, & \text{если } x < 0. \end{cases}$$

Приведем некоторые примеры вычислений:

1) вычислить синус угла в 30 градусов

```
octave:2> sin(30*pi/180)
ans =  0.50000
```

2) найти наибольший общий делитель чисел 24, 86, 144

Таблица 1: Алгебраические функции Octave

`exp(x)`	вычисление функции e^x	`sign(x)`	сигнум-функция от x
`pow2(x)`	вычисление функции 2^x	`abs(x)`	абсолютное значение x
`pow2(f,e)`	вычисление функции $f \cdot 2^e$	`min(x,y)`	минимум из x и y
`log(x)`	натуральный логарифм x	`max(x,y)`	максимум из x и y
`log10(x)`	десятичный логарифм x	`round(x)`	округление x до ближайшего целого
`sqrt(x)`	квадратный корень x	`ceil(x)`	округление x до целого сверху
`sin(x)`	синус x	`floor(x)`	округление x до целого снизу
`cos(x)`	косинус x	`lcm(x,...)`	наименьшее общее кратное $x, \ldots$
`tan(x)`	тангенс x	`gcd(x,...)`	наибольший общий делитель $x, \ldots$
`cot(x)`	котангенс x	`rem(x,y)`	остаток от деления x на y

```
octave:4> gcd(24,84,144)
ans =  12
```

3) вычислить функцию $\ln 10 + 2^5 \cdot 13.7$

```
octave:5> log(10)+pow2(13.7,5)
ans =  440.70
```

1.3 Функции и переменные

Как и в любом языке программирования в Octave есть понятия переменной и функции, создавать которые намного проще, чем, к примеру, в таких языках как C или Pascal.

Переменные в Octave начинают существовать, как только им было присвоено значение. Тип переменной также определяется присвоенным значением. Одномерные и двумерные массивы переменных определяются как вектора и матрицы через квадратные скобки. Требования к именам переменных стандартные, как для многих языков: имя может состоять из букв латинского алфавита, цифр (везде, кроме первого символа имени) и символа нижнего подчеркивания.

```
octave:1> x1=2
x1 =  2
octave:2> x2=3.7
x2 =  3.7000
octave:3> v=[2 4.6 1]
v =

   2.0000   4.6000   1.0000
octave:4> A = [1 3 4; 2 4 6]
A =

   1   3   4
   2   4   6
```

В приведенном примере показано, как создаются целочисленная переменная $x1$, вещественная переменная $x2$, одномерный массив чисел v и двумерный массив чисел A.

Манипулировать переменной можно только после ее определения. Не вызовет ошибки присвоение значения другого типа существующей переменной.

```
octave:5> x1=v
x1 =

 2.0000  4.6000  1.0000
```

С переменными тесно связано такое понятие, как функция. Даже при всем многообразии встроенных в Octave функций на практике их может оказаться недостаточно. Отсюда возникает необходимость создавать новые функции для решения конкретных задач пользователя.

Как правило, новую функцию создают либо в отдельном файле, либо в скрипт-файле Octave до первого ее вызова. Если предполагается использовать пользовательскую функцию в разных скрипт-файлах, то, конечно, предпочтительно создать ее в отдельном файле. В GNU Octave файлы с функциями имеют расширение `.m` и загружаются автоматически. Имя файла должно строго совпадать с именем функции.

Общий синтаксис создания новой функции следующий

`function` [*res1, res2, ..., resM*] = *имя_функции* (*arg1, arg2, ..., argN*)
тело функции
`endfunction`

Начинает описание ключевое слово `function`, затем в квадратных скобках перечисляются выходные параметры функции *res1, res2, ..., resM*. Если выходной параметр один, то квадратные скобки можно опустить. Если выходных параметров нет, то после ключевого слова `function` сразу указывается имя функции. Требования к составлению имени функции аналогичны требованиям к именам переменных. Как правило, функция имеет один или несколько входных параметров *arg1, arg2, ..., argN*, которые перечисляются после имени функции в круглых скобках через запятую. Тело функции состоит из команд Octave. Заканчивает описание функции ключевое слово `endfunction`.

Все входные *arg1, arg2, ..., argN* и выходные *res1, res2, ..., resM* параметры в описании функции, а также все переменные, созданные в ее теле, являются локальными для данной функции.

В Octave нет необходимости подгружать описание функции из файла перед ее использованием.

Напишем функцию решения квадратного уравнения $ax^2 + bx + c = 0$

```
function [x1,x2] = quad_eq(a,b,c)
 D = sqrt(b^2-4*a*c);
 x1 = (-b-D)/(2*a);
 x2 = (-b+D)/(2*a);
endfunction
```

Заметим, что при вычислении дискриминанта опущена проверка на неотрицательность подкоренного выражения. Это необязательно, поскольку Octave умеет работать как с вещественными, так и с комплексными числами.

Сохраним описание функции в файле `quad_eq.m`. Теперь ей можно воспользоваться при работе с Octave

```
octave:6> a=2; b=3; c=-2;
octave:7> [y1,y2]=quad_eq(a,b,c)
y1 = -2
y2 = 0.50000
```

Вызов функции осуществляется по имени, количество входных и выходных параметров, указанных при вызове, должно точно совпадать с их количеством в описании функции. Имена передаваемых и получаемых параметров при вызове функции могут не совпадать с именами, указанными в описании.

что когда Octave встречает функцию, определенную пользователем, она сперва пытается найти ее в текущей директории, т.е. найти файл с расширением `*.m` и именем, совпадающим с именем функции

1.4 Матрично-векторные вычисления

В настоящем разделе покажем, как средствами Octave легко и быстро можно создавать случайные и специальные матрицы любой размерности и производить над ними разные арифметические и алгебраические действия.

1.4.1 Создание матриц

Чтобы создать в Octave матрицу или вектор с заданными элементами нужно просто перечислить эти элементы в квадратных скобках, разделяя строки знаком "точка с запятой", столбцы запятой или пробелом

```
octave:1> A = [ 2 4 5; 6 4 7; 6 1 8]
A =

   2   4   5
   6   4   7
   6   1   8
octave:2> v = [ 3 6.4 8]
v =

   3.0000   6.4000   8.0000

octave:3> w=[5 ; 7 ; 8]
w =

   5
   7
   8
```

Функция `rand` позволяет генерировать случайные числа из нтервала (0,1). Если функция вызывается без аргументов, то будет создано случайное число.

```
octave:5> r=rand
r =  0.67452
```

Если функция имеет один аргумент `rand(m)`, то будет создана квадратная матрица размерности $m \times m$.

```
octave:6> A=rand(5)
A =
```

```
   0.217300   0.272742   0.439678   0.040161   0.550001
   0.620170   0.379312   0.416012   0.295613   0.608457
   0.387311   0.649955   0.808548   0.166677   0.259772
   0.411485   0.903623   0.204110   0.443964   0.696127
   0.994622   0.598606   0.257824   0.637861   0.969088
```

Если передать два аргумента `rand(m,n)`, то будет создана матрица размерности $m \times n$

```
octave:7> B=rand(4,6)
B =

   0.1410003   0.6681751   0.9649029   0.0092332   0.9570230   0.7681693
   0.1151950   0.0958126   0.2708919   0.4965380   0.3816898   0.2960198
   0.3417755   0.1450904   0.1535524   0.5251388   0.9805246   0.2559105
   0.0609999   0.4968790   0.0741969   0.5419970   0.8560731   0.6810124
```

Если один из аргументов равен 1, то будет создан вектор случайных элементов

```
octave:8> v=rand(1,3)
v =

   0.294654   0.492241   0.079515
octave:9> w=rand(3,1)
w =

   0.767195
   0.027604
   0.159826
```

Аналогично `rand` вызывается функция `randn`, которая генерирует случайные числа, имеющие нормальное распределение

```
octave:10> r = randn
r = -0.41171
octave:11> B = randn(4,6)
B =

  -0.73689   0.46540  -0.72798  -1.64099  -1.11487  -0.30241
   0.53960  -1.54595   0.96346  -0.51525   0.46718  -0.54186
   0.19417  -0.35375  -2.79121  -0.50133   0.19464  -1.13815
   0.14187   2.23729  -0.92617   0.27607   2.62902  -0.25009
octave:12> v=randn(1,3)
v =

  -0.984558   0.070915  -0.781345
```

В теории линейной алгебры выделяют специальные виды матриц, например, единичная, диагональная, блочная или нулевая. Создать такие матрицы в Octave позволяют следующие функции:

- `eye(`m`)`, `eye(`m`,`n`)`
  Функция генерирует единичную матрицу (с одним аргументом — квадратная матрица размерности $m \times m$, с двумя аргументами — матрица размерности $m \times n$).

```
octave:16> I = eye(3)
I =

   1   0   0
   0   1   0
   0   0   1

octave:17> I = eye(3,5)
I =

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
```

- `ones(`m`)`, `ones(`m`,`n`)`
  Функция генерирует матрицу с элементами 1 (количество аргументов трактуется аналогично `eye`.

```
octave:18>  = ones(3)
O =

   1   1   1
   1   1   1
   1   1   1
octave:19> e = ones(1,5)
e =

   1   1   1   1   1
```

- `zeros(`m`), zeros(`m`,`n`)`
  Функция генерирует нулевую матрицу (количество аргументов трактуется аналогично `eye`.

```
octave:29> Z = zeros(3)
Z =

   0   0   0
   0   0   0
   0   0   0

octave:30> z = zeros(1,3)
z =

  0   0   0
```

- `diag(`v`, `k`)`
  Функция генерирует диагональную матрицу с элементами вектора v на диагонали k. Аргумент k выступает в качестве опции: $k = 0$ означает главную диагональ, при $k > 0$ элементы вектора v ставятся на k-ую диагональ выше, а при $k < 0$ на k-ую диагональ ниже главной диагонали. По умолчанию $k = 0$. Размер матрицы определяется в соответствии с аргументами v и k.

```
octave:36> D = diag([1 2 3 4 5])
D =

   1   0   0   0   0
   0   2   0   0   0
   0   0   3   0   0
   0   0   0   4   0
   0   0   0   0   5

octave:37> D = diag([1 2],2)
D =

   0   0   1   0
   0   0   0   2
   0   0   0   0
   0   0   0   0

octave:38> D = diag([1 2],-1)
D =

    0   0   0
    1   0   0
    0   2   0
```

  Отметим еще одно полезное для практического использования свойство функции `diag`. Если первым аргументом v является матрица, то функция возвращает вектор-столбец, состоящий из элементов, стоящих на диагонали маркируемой опцией k.

```
octave:16> A = ceil(rand(3)*100)
A =

    52   100    61
    71    98    66
    41    91    41

octave:17> diag(A,0)
ans =

   52
   98
```

```
   41

octave:18> diag(A,-1)
ans =

   71
   91
```

- `repmat`(A, m), `repmat`(A, m, n)
  Функция генерирует блочную матрицу размерности $m \times m$ ($m \times n$), каждым элементом которой является подматрица A.

```
octave:39> B = repmat(eye(2),2,3)
B =

   1   0   1   0   1   0
   0   1   0   1   0   1
   1   0   1   0   1   0
   0   1   0   1   0   1
```

Полезной в вычислениях может оказаться функция `randperm(n)` генерации целочисленного вектора, элементы которого представляют случайную перестановку чисел от 1 до n

```
octave:13> p=randperm(6)
p =

   6   1   4   3   2   5
```

Сгенерировать вектор индексов от M до N позволяет команда `[M:N]`

```
octave:14> q=[13:23]
q =

   13   14   15   16   17   18   19   20   21   22   23
```

Диапазон значений от M до N с шагом *step* задается командой `[it M:`*step*`:N]`. При положительном шаге для корректной работы данной команды должно быть выполнено условие $M \leq N$, при отрицательном шаге — $M \geq N$.

```
octave:11> q = 13:2:23
q =

  13  15  17  19  21  23

octave:12> q = 23:-2:13
q =

  23  21  19  17  15  13
```

Команда `linspace(`*base, limit, n)* возвращает вектор-строку из n элементов, равномерно распределенных на отрезке $[base, limit]$ числовой прямой.

```
octave:15> v=linspace(0, 1, 8)
v =

   0.00000   0.14286   0.28571   0.42857   0.57143   0.71429   0.85714   1.00000
```

Число элементов n должно быть больше 1. По умолчанию $n = 100$. Результирующий вектор включает элементы *base* и *limit*. Если *base* больше *limit*, то n элементов упорядочиваются по убыванию.

```
octave:16> v=linspace(1, 0, 8)
v =

   1.00000   0.85714   0.71429   0.57143   0.42857   0.28571   0.14286   0.00000
```

И наконец, в Octave можно определить пустую матрицу, для этого используют команду `[]`

```
octave:17> A=[]
A = [](0x0)
```

или вызывают одну из вышеописанных функций (например, `rand, zeros, ones` и т.д.), указав при этом хотя бы одну из размерностей равной 0.

```
octave:21> A=rand(0,0)
A = [](0x0)
octave:22> A=rand(3,0)
A = [](3x0)
octave:23> A=rand(0,3)
A = [](0x3)
```

Пустые матрицы удобно использовать, когда необходимо удалить из заданной матрицы некоторые строки или столбцы.

```
octave:5> A = rand(3,7)
A =

   0.562329   0.670663   0.046794   0.802909   0.019110   0.906351   0.299616
   0.217606   0.696152   0.573524   0.328735   0.355639   0.362626   0.475531
   0.680574   0.578710   0.254646   0.484237   0.080157   0.442885   0.125110

octave:6> A(:,1:2:7)=[]
A =

   0.67066   0.80291   0.90635
   0.69615   0.32874   0.36263
   0.57871   0.48424   0.44288
```

1.4.2 Действия над матрицами

Прежде всего научимся определять размерность матриц, с которыми предстоит работать, поскольку многие бинарные операции предполагают соблюдения определенной согласованности между размерностями своих операндов.

Узнать число строк и столбцов данной матрицы позволяет функция `size`. Аргументом функции является матрица или вектор, первый возвращаемый параметр определяет число строк, второй — число столбцов.

```
octave:43> A = rand(10,60);
octave:44> [rowA, colA]=size(A)
rowA =  10
colA =  60
octave:45> v = ones(1,100);
octave:46> size_v = size(v)
size_v =

     1  100
```

Если интересует только количество строк матрицы, то вместо `size` можно использовать функцию `rows`.

```
octave:49> rows(A)
ans =  10
octave:50> rows(v)
ans =  1
```

А если надо определить только количество столбцов, то используется функция `columns`.

```
octave:51> columns(A)
ans =  60
octave:52> columns(v)
ans =  100
```

Сложение (`+`), вычитание (`-`), умножение (`*`) матриц в Octave основано на тех же правилах, что и в линейной алгебре. Складывать и вычитать можно только матрицы одинаковой размерности.

```
octave:54> A = rand(2,3); B=randn(2,3);
octave:55> A+B
ans =

   9.7920e-04   2.4098e+00  -7.4398e-01
  -2.0905e+00   2.7025e+00   2.1091e-01

octave:56> A-B
ans =

   0.36546  -0.94482   1.32552
   2.13113  -0.97702   0.53200
```

При умножении матриц A на B число столбцов у A должно быть равно числу строк у B.

```
octave:57> A = rand(2,3); B=randn(3,2);
octave:58> A*B
ans =

   1.43483   0.18918
   0.77856   0.26710
```

Результатом умножение матрицы $A = (a_{ij})_{m\times n}$ на скаляр c является матрица $B = (c \cdot a_{ij})_{m\times n}$.

```
octave:3> A = [1 2 3; 4 5 6];
octave:4> B = A*10
B =

   10   20   30
   40   50   60
```

Не вызовет ошибки, если применить бинарные операции сложения, вычитания или деления к матрице и скалярной величине, при этом каждая операция будет выполнятся поэлементно.

```
octave:2> A+10
ans =

   11   12   13
   14   15   16

octave:3> A-10
ans =

  -9  -8  -7
  -6  -5  -4

octave:4> 10-A
ans =

   9   8   7
   6   5   4

octave:5> A/10
ans =

   0.10000   0.20000   0.30000
   0.40000   0.50000   0.60000
```

Транспонировать матрицу позволяет оператор ' (штрих).

```
octave:5> A'
ans =

   1   4
   2   5
   3   6
```

Помимо упомянутых выше операций в Octave есть возможность выполнять поэлементное умножение (`.*`) и деление (`./`) матриц (при этом матрицы должны быть одной размерности).

```
octave:59> A = rand(2,3); B=randn(2,3);
octave:60> A.*B
ans =

   0.7426443   0.0971287  -0.0018915
   0.4088991  -0.2743542  -0.1120034

octave:61> A./B
ans =

   1.2119e+00   1.6707e+00  -7.4780e-04
   1.3371e+00  -2.3146e-01  -1.7738e+00
```

Для квадратных матриц имеется операция поэлементного возведения в степень (`.^`)

```
octave:62> A=randn(3);
octave:63> A.^3
ans =

   2.2750344   0.0389019  -8.4121279
  -0.0784881   0.0046370   1.2989715
   0.4925376  -1.7890019   1.7549553
```

Кроме того, большинство из функций, приведенных в таблице 1 (стр. 6) в качестве аргумента могут получать не только скалярную величину, но также матрицу или вектор, при этом вычисление функции проводится поэлементно.

```
octave:53> sqrt(A)
ans =

   3.1623   5.9161
   5.0000   3.8730
```

Отметим некоторые особенности работы функций вычисления минимального (`min(`x`)`) и максимального (`max(`x`)`) элементов. Для аргумента вектора данные функции возвращают два скалярных параметра, первый — это значение минимального (максимального) элемента вектора x, второй — это номер компоненты, на которой достигается минимум (максимум).

```
octave:19> v = randperm(8)
v =

   6   8   3   1   5   2   7   4

octave:20> [min_v,inx_min]=min(v)
min_v =  1
inx_min =  4
```

```
octave:21> [max_v,inx_max]=max(v)
max_v =  8
inx_max =  2
```

Если функции `min` и `max` в качестве аргумента получают матрицу, то они возвращают два векторных параметра, первый — это вектор минимумов в каждом столбце матрицы x, второй — вектор номеров строк, где достигается минимум в соответствующем столбце.

```
octave:23> A = round(rand(2,5)*100)
A =

   96   18   17   74    6
   60   21   82    5    4

octave:24> [min_A,inx_min]=min(A)
min_A =

   60   18   17    5    4

inx_min =

   2   1   1   2   2

octave:25> [max_A,inx_max]=max(A)
max_A =

   96   21   82   74    6

inx_max =

   1   2   2   1   1
```

Если результат работы функций `max` и `min` присваивать только одному параметру, то будут получены только минимальные значения

```
octave:31> max_v=max(v)
max_v =  8
octave:32> min_A=min(A)
min_A =

   60   18   17    5    4
```

Для матриц и векторов список функций таблицы 1 можно дополнить функциями поэлементного суммирования и умножения компонент вектора.

- `sum`(x)
  Функция поэлементного сложения компонент вектора. Если x — вектор, то возвращает сумму всех компонент x.

```
octave:26> x = randperm(4)
x =

   4   3   1   2

octave:27> sum(x)
ans =  10
```

Если x — матрица, то возвращает вектор сумм компонент каждого столбца x.

```
octave:28> A = [1 2 3 4; 5 4 7 8];
octave:29> sum(A)
ans =

    6    6   10   12
```

- `prod(`x`)`
  Функция поэлементного умножения компонент вектора. Принцип работы аналогичен функции `sum`.

```
octave:32> prod(x)
ans =  24
octave:33> prod(A)
ans =

    5    8   21   32
```

- `cumsum(`x`)`
  Функция кумулятивного сложения компонент вектора. Если x — вектор, то возвращает вектор каждая i-ая компонента которого равна сумме первых i компонент вектора x.

```
octave:37> cumsum(x)
ans =

    4    7    8   10
```

Если x — матрица, то возвращает матрицу, каждый столбец которой представляет кумулятивную сумму компонент соответствующего столбца x.

```
octave:38> cumsum(A)
ans =

    1    2    3    4
    6    6   10   12
```

- `cumprod(`x`)`
  Функция кумулятивного умножения компонент вектора. Принцип работы аналогичен функции `cumsum`.

```
octave:39> cumprod(x)
ans =

    4   12   12   24

octave:40> cumprod(A)
ans =

    1    2    3    4
    5    8   21   32
```

- `sumsq`(x)
  Функция вычисления суммы квадратов. Принцип работы аналогичен функции `sum`.

```
octave:41> sumsq(x)
ans =  30
octave:42> sumsq(A)
ans =

   26   20   58   80
```

1.4.3 Алгебра матриц

Одним из объектов изучения в дисциплине линейная алгебра является анализ системы линейных уравнений $Ax = b$ и поиск ее решения.

Для квадратной матрицы A можно выписать аналитическое решение системы $x = A^{-1}b$, где A^{-1} – обратная к A матрица. Для обращения матриц в Octave существует функция `inv`, аргументом которой должна быть только квадратная матрица.

```
octave:1> A = rand(3);
 octave:2> invA = inv(A)
 invA =

    2.69454  -1.52685   0.47050
   -1.05052   3.11871  -1.29602
   -1.55241   0.87150   1.12319

 octave:3> A*invA
 ans =

    1.00000  -0.00000  -0.00000
    0.00000   1.00000  -0.00000
    0.00000  -0.00000   1.00000
```

Таким образом зная значение вектора правых частей b, средствами Octave можно просто определить неизвестный вектор x

```
octave:4> b = rand(3,1);
```

```
octave:5> x = inv(A)*b
x =

   0.522555
  -0.015841
   0.614833
```

В правой части системы линейных уравнений может стоять матрица b, способ решения такой системы не меняется, главное соблюдать правила согласования размерностей A и b — число строк данных матриц должно быть одинаковым.

```
octave:6> b=rand(3,2);
octave:7> x = inv(A)*b
x =

   1.05918   0.66053
   1.20880   0.86253
   0.17613   0.35082

octave:8> A*x-b
ans =

  -1.1102e-16   0.0000e+00
   0.0000e+00  -1.1102e-16
  -1.1102e-16   0.0000e+00
```

Вместо функции обращения матрицы для решения системы $Ax = b$ можно использовать оператор матричного деления \

```
octave:9> x=A\b
x =

   1.05918   0.66053
   1.20880   0.86253
   0.17613   0.35082
```

Как известно из курса линейной алгебры обратные матрицы существуют только для невырожденных матриц, то есть для матриц, определитель которых отличен от нуля. Поэтому перед тем, как вычислить значения переменных x хорошо бы сперва проверить матрицу A на вырожденность. Функция `det` возвращает определитель своего аргумента — квадратной матрицы

```
octave:10> det(A)
ans =  0.10575
```

При исследовании систем линейных уравнений с произвольной матрицей A размерности $m \times n$ используют такое понятие как ранг матрицы (наибольший из порядков ненулевых миноров матрицы). Функция `rank` возвращает ранг своего аргумента.

```
octave:11> rank(A)
ans =  3
```

1.4.4 Поиск элементов и проверка условий

В ситуациях, когда необходимо определить удовлетворяют ли элементы данной матрицы некоторому заданному условию, могут пригодиться имеющиеся в арсенале Octave функции. Такие функции, как правило, в качестве выходного параметра возвращают булеву величину (скаляр, матрицу или вектор), которая и указывает выполнено условие (на выходе 1) или нет (на выходе 0).

- `any(`x`)`
  Для аргумента вектора проверяет является ли x ненулевым вектором.

```
octave:1> any(zeros(1,3))
ans = 0
octave:2> any([ 2 0 5 6])
ans =  1
```

  Для аргумента матрицы проверяет является ли каждый из столбцов x ненулевым вектором

```
octave:6> A = [1 0 3; 4 0 8];
octave:7> any(A)
ans =

   1   0   1
```

- `all(`x`)`
  Для аргумента вектора проверяет все ли элементы x ненулевые.

```
octave:8> all([2 0 5 6])
ans = 0
octave:9> all([2 1 5 6])
ans =  1
```

  Для аргумента матрицы проверяет все ли элементы каждого из столбцов ненулевые

```
octave:11> A = [1 1 3; 4 0 8];
octave:12> all(A)
ans =

   1   0   1

octave:13> A = [1 1 3; 4 1 8];
octave:14> all(A)
ans =

   1   1   1

octave:15> all(all(A))
ans =  1
```

Таблица 2: операторы сравнения

==	равно	<	меньше	>	больше
!=	не равно	<=	меньше или равно	>=	больше или равно

Помимо вызова специальных функций проверить выполнение условий можно и при помощи обычных операторов сравнения, список которых приведен в таблице 2.
Результатом таких операций также являются булевы величины. При сравнении скаляров результат 1 означает, что условие выполнено, 0 — условие не выполнено. Если операторы сравнения применяются к матрицам и векторам, то условие проверяется поэлементно, результатом является булева матрица или вектор соответственно. Например, если требуется определить какие элементы матрицы A не меньше некоторого заданного числа c, то просто надо дать команду `A>=c`.

```
octave:1> A = randn(2,5)
A =

   0.88935  -1.34448  -0.36394  -0.46218   1.21992
  -0.90290   2.48469  -0.81393   1.13628   1.33236

octave:2> c = 0;
octave:3> A >= c
ans =

   1   0   0   0   1
   0   1   0   1   1
```

Для проверки можно указывать несколько условий, для этого используются следующие булевы операторы

`&`: покомпонентная конъюнкция *boolean1* `&` *boolean2*
Элемент результата равен 1, если оба соответствующих ему элемента *boolean1* и *boolean2* не равны 0.

```
octave:10> A>=c & A<=2
ans =

   1   0   0   0   1
   0   0   0   1   1

octave:4> [1 0 0 1] & [1 0 2 3]
ans =

   1   0   0   1

octave:5> [1 0; 0 1] & [1 0 ;2 3]
ans =
```

```
   1   0
   0   1
```

`|`: покомпонентная дизъюнкция *boolean1* `|` *boolean2*
Элемент результата равен 1, если хотя бы один из двух соответствующих ему элементов *boolean1* и *boolean2* не равен 0.

```
octave:14> A<=-1 | A>=2
ans =

   0   1   0   0   0
   0   1   0   0   0

octave:6> [1 0 0 1] | [1 0 2 3]
ans =

   1   0   1   1

octave:7> [1 0; 0 1] | [1 0; 2 3]
ans =

   1   0
   1   1
```

`!`: покомпонентное отрицание *boolean*
Элемент результата равен 1, если соответствующий ему элемент *boolean* равен 0.

```
octave:15> !(A<=-1 | A>=2)
ans =

   1   0   1   1   1
   1   0   1   1   1

octave:8> ![1 0 ;2 3]
ans =

   0   1
   0   0
```

`&&`: замкнутая конъюнкция *boolean1* `&&` *boolean2*
Выполняется следующая последовательность шагов. Проверяется выражения *boolean1* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean1*`)`. Если получен 0, то значение всего выражения 0. Если получена 1, то проверяется выражения *boolean2* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean2*`)`. Если теперь получен 0, то значение всего выражения 0, в противном случае значение всего выражения равно 1.

```
octave:6> A>-1 && A<3
ans = 0
octave:7> A>-2 && A<2.5
ans =  1
```

`||`: замкнутая дизъюнкция *boolean1* `||` *boolean2*
Выполняется следующая последовательность шагов. Проверяется выражения *boolean1* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean1*`)`. Если получена 1, то значение всего выражения 1. Если получен 0, то проверяется выражения *boolean2* и результат конвертируется в булев скаляр, как если бы была применена операция `all(`*boolean2*`)`. Если теперь получена 1, то значение всего выражения 1, в противном случае значение всего выражения равно 0.

```
 octave:12> A>-1 || A<3
 ans =  1
 octave:13> A>-1 || A<2
ans = 0
```

Замкнутые булевы операторы иногда могут заменить многократное вложение условного оператора `if`. Например, команда

```
A>0 && log(A)
```

предписывает Octave вычислять логарифм элементов матрицы A только в случае, когда все ее элементы положительны. Если это так, то результат выполнения команды равен 1, в противном случае 0.

1.4.5 Формирование подматриц

Обращение к элементам матрицы осуществляется через круглые скобки с указанием номера строки и столбца, на пересечении которых стоит элемент.

```
octave:8> A=[ 2 6 7 5; 8 4 1 3; 2 6 3 5];
octave:9> a_21=A(2,1)
a_21 =  8
```

Средства Octave позволяют указывать не только по одному номеру строки и столбца элемента, но и задавать наборы номеров строк и столбцов, по которым можно вырезать подматрицу. Такие наборы формируются в виде целочисленного вектора, элементы которого не должны превышать количества строк (столбцов) матрицы.

```
octave:11> M=A([1 3],[2 4])
M =

   6   5
   6   5

octave:12> Q=A([1:3],[2 4])
Q =
```

```
   6   5
   4   3
   6   5
```

Если подматрица содержит все строки (столбцы) исходной матрицы, то вместо полного набора номеров можно указать знак двоеточие (:).

```
octave:13> Q=A(:,[2 4])
Q =

   6   5
   4   3
   6   5
```

Вырезать подматрицы можно и по булевым векторам, при этом из исходной матрицы будут браться только столбцы и строки, соответствующие компонентам 1 булева вектора.

```
octave:14> bv = [0 2 0 4] > 0
bv =
   0  1  0  1
octave:15> Q=A(:,bv)
Q =

   6   5
   4   3
   6   5
```

Необходимо обратить особое внимание, что вектор созданный как

```
octave:16> v = [1 0 1 0 0 1]
v =
   1  0  1  0  0  1
```

не является для Octave булевым, поэтому попытка вырезать по нему подматрицу окончится неудачей. Получить булев вектор с аналогичной структурой можно командой

```
v = [1 0 1 0 0 1] > 0
v =
   1  0  1  0  0  1
```

1.5 Операторы управления ходом выполнения программы

На практике решение большинства задач не удается описать с помощью программ линейной структуры, поэтому как и любой высокоуровневый язык программирования Octave содержит условные операторы и операторы циклов. Использование каждого из таких операторов предполагает соблюдение определенного синтаксиса языка.

Для описания операторов примем следующие обозначения:

условие — выражение Octave, результатом выполнения которого является булевая величина;
тело — любая последовательность команд Octave.

1.5.1 Условный оператор `if`

Условный оператор `if` предназначен для проверки некоторого условия. По результатам такой проверки выполняется та или иная последовательность команд. Оператор начинается с ключевого слова `if` и заканчивается ключевым словом `endif`

Существует три основные формы вызова оператора `if`. Самая простая из них имеет вид

```
if (условие)
тело
endif
```

тело оператора будет выполнено, только если значение указанного *условия* истинно.

Заметим, что для оператора `if` любое ненулевое значение *условия* интерпретируется как истина. Если значение *условия* это вектор или матрица, то значение истинность будет только в случае, когда все элементы результирующего вектора или матрицы ненулевые.

```
octave:1> a = [ 2 4 6];
octave:2> b = [ 1 4 6];
octave:3> if a>=b c=a endif
c =

   2   4   6
```

Вторая возможная форма вызова оператора имеет вид:

```
if (условие)
тело1
else
тело2
endif
```

если значение *условия* истинно, то будет выполнено *тело1*, в противном случае будет выполнено *тело2*.

```
if det(A)
 x = A\b
else  x=0
endif
```

В приведенном примере, если определитель матрицы не равен нулю, то будет вычислен вектор x — решение системы уравнений $Ax = b$. В противном случае (если матрица A вырожденная), переменной x будет присвоено нулевое значение.

Третьей и самой общей формой оператора является

```
if (условие1)
тело1
elseif (условие2)
тело2
elseif (условие3)
тело3
...
```

```
elseif (условиеN)
телоN
else
тело(N+1)
endif
```

Последовательно проверяются перечисленные условия до первого их выполнения. Если одно из условий истинно, то выполняется соответствующее тело. Если ни одно из условий не выполнено, то выполняется тело, соответствующее разделу `else`. Ключевое слово *else* в данной конструкции оператора `if` должно встречаться только один раз и только в заключительной части оператора `if`.

```
if (!rem(x,2))
 y = x/2
elseif (!rem(x,3))
 y = x/3
else
 y = 0
endif
```

В приведенном примере сперва осуществляется проверка является ли x четным числом, если да, то значение $y = x/2$. Если x — нечетное, то проверяется делиться ли x на 3. При истинном результате $y = x/3$. Если x не делится ни на 2 ни на 3, то $y = 0$.

1.5.2 Оператор выбора `switch`

Кроме условного оператора `if` в качестве управляющей структуры может оказаться удобным использование оператора выбора `switch`. Эта структура позволяет переходить на одну из ветвей в зависимости от значения заданного выражения. Ее особенность состоит в том, что выбор решения здесь осуществляется не в зависимости от истинности или ложности условия, а является вычислимым.

Оператор начинается ключевым словом `switch`, внутри оператора должен быть хотя бы один раздел `case`, заканчивается оператор ключевым словом `endswitch`. Общая форма вызова оператора следующая

```
switch выражение
case значение1
тело1
case значение2
тело2
...
otherwise
тело
endswitch
```

В конструкции `switch` вычисляется *выражение* и выбирается ветвь, *значение* которой совпадает со значением *выражения*. После выполнения *тела* выбранной ветви происходит выход из конструкции `switch`. Если в последовательности нет *значения*, равного *выражению*, то при наличии раздела `otherwise` выполняется соответствующее ему *тело*, при отсутствии `otherwise` управление передается внешнему оператору, следующему за конструкцией `switch`.

```
switch x<y
case 1
 min=x
otherwise
 min=y
endswitch
```

В приведенном примере проверяется выполнение условия $x < y$, если оно выполнено (значение истинно), то переменной `min` присваивается значение переменной `x`, в противном случае `min= y`.

```
switch sign(x)
case 1
 disp("x>0");
case 0
 disp("x=0")
case -1
 disp("x<0")
endswitch
```

В зависимости от того, чему равно значение сигнум-функции, на экран выводится информация о знаке числа. Описание функции `disp` дано в разделе 1.6 на стр. 31.

1.5.3 Оператор цикла `while`

Как правило под циклом в языках программирования понимают многократное повторение некоторой части кода. Количество таких повторений зависит от условий остановки цикла. Самым простым в Octave можно считать оператор цикла `while`.

Оператор начинается с ключевого слова `while` и заканчивается ключевым словом `endwhile`. Цикл формируется по следующей схеме

`while (`*условие*`)`
тело
`endwhile`

тело цикла будет выполнятся до тех пор, пока *условие* имеет истинное значение. Как и в случае условного оператора `if` истинным считается любое ненулевое значение, если *условие* представляет из себя вектор или матрицу, то истинным оно будет считаться только тогда, когда все элементы вектора или матрицы не равны нулю.

```
F = ones(1,10);
i = 3;
while (i <= 10)
 F(i) = F(i-1) + F(i-2);
 i++;
endwhile
```

Приведен пример программы по созданию последовательности из первых 10 чисел Фибоначчи. Цикл `while` закончит свою работу, когда значение индекса `i` станет равным 11.

1.5.4 Оператор цикла `do-until`

Альтернативой `while` служит оператор цикла `do-until`. Общая схема вызова следующая

```
do
тело
until (условие)
```

Цикл `do-until` отличается от цикла `while` по двум позициям. Первая — *тело* цикла `do-until` выполняется до тех пор, пока *условие* не будет выполнено. Вторая — в `do-until` сначала выполняется *тело*, а потом проверяется *условие*, тогда как в `while` сперва проверяется *условие*, а потом, в зависимости от результата, выполняется *тело*. Таким образом как минимум один раз *тело* цикла `do-until` будет выполнено. Аналогично оператору `if` истинным считается любое ненулевое значение, если *условие* представляет из себя вектор или матрицу, то истинным оно будет считаться только тогда, когда все элементы вектора или матрицы не равны нулю.

Следующий код позволяет построить последовательность первых десяти чисел Фибоначчи с помощью цикла `do-until`.

```
F = ones(1,10);
i = 2;
do
 i++;
 F(i) = F(i-1) + F(i-2);
until (i == 10)
```

1.5.5 Оператор цикла `for`

Цикл с определенным числом повторений некоторых действий удобнее формировать при помощи оператора `for`. Начинается цикл с ключевого слова `for` и заканчивается ключевым словом `endfor`. Общая схема вызова следующая

```
for переменная = выражение
тело
endfor
```

Оператор последовательно перебирает по одной колонке *выражения* и присваивает ее *переменной*. Если *выражение* задает диапазон `[m:n]`, вектор-строку или скаляр, то при каждом выполнении *тела* цикла значением *переменной* будет скаляр. Если *выражение* задает вектор-столбец или матрицу, то при каждом выполнении *тела* цикла значением *переменной* будет вектор-столбец.

Приведем пример генерации последовательности первых десяти чисел Фибоначчи с помощью оператора `for`.

```
F = ones(1,10);
for i=3:10
 F(i) = F(i-1) + F(i-2);
endfor
```

Переменная `i` последовательно пробегает все значения диапазона от 3 до 10, при этом для каждого нового присвоения рассчитывается очередной член последовательности Фибоначчи `F(i)`.

На следующем примере показан процесс построения матрицы B, которая получается из заданной матрицы A путем нормировки каждой из колонок.

```
B = [];
for col = A
 B = [B col / sum(col)];
endfor
```

1.5.6 Дополнительные операторы

Аналогично языку Си, Octave включает в себя операторы досрочного прерывания или пропуска некоторых из операций *тела* цикла.

Оператор `break` обеспечивает прекращение выполнения самого внутреннего из объединяющих его операторов `switch, while, do-until, for`. После выполнения оператора `break` управление передается оператору, следующему за прерванным.

В качестве примера использования оператора `break` приведем функцию, определяющую наименьший делитель некоторого заданного числа.

```
function div = smalldiv(num)
 div = 2;
 while (div*div <= num)
  if rem(num,div) == 0
   break;
  endif
  div++;
 endwhile

 if rem(num,div) div = 1; endif
endfunction
```

Даже если текущее значение переменной `div` удовлетворяет условию `div*div <= num` оператор цикла `while` прервет свою работу, как только число `num` без остатка поделится на `div`.

Если необходимо не навсегда выйти из цикла, а просто досрочно завершить текущую итерацию, то используют оператор `continue`. В отличие от оператора `break`, при выполнении оператора `continue` управление программой передается на начало ближайшего внешнего оператора цикла `for` или `while`.

Рассмотрим следующий пример возможного использования оператора `continue`.

```
vec = randn(5); newvec = [];
for v = vec
 s = sum(v);
 if s == 1
  continue;
 elseif s > 0
  newvec = [newvec v/s];
 else
  newvec = [newvec abs(v)/sum(abs(v))];
 endif
endfor
```

Здесь из системы вектор-столбцов `vec` (по сути `vec` – это матрица) формируется новая система `newvec` по правилу: из `vec` помещаются в `newvec` только те вектора `v`, сумма компонент кото-

рых не равна 1, при этом если эта сумма положительна, то вектор `v` нормируется, если сумма отрицательна, то нормируется вектор `abs(v)`.

Стоит заметить, что в ряде случаев использование оператора `continue` можно заменить индексными выражениями. Например, синтаксис языка Octave позволяет переписать предыдущий пример без оператора `continue` в виде

```
vec = randn(5);
newvec = vec(:, sum(vec) != 1);
bv = sum(newvec) > 0;
newvec(:, bv) = newvec(:,bv) ./ ( ones(5,1) * sum(newvec(:,bv)) );
newvec(:, !bv) = abs(newvec(:,!bv)) ./ ( ones(5,1) * sum(abs(newvec(:,!bv))) );
```

1.6 Ввод и вывод информации

После завершения процесса вычислений результаты работы программы можно сохранить в отдельном файле, чтобы потом иметь возможность воспользоваться ими для анализа или дальнейших преобразований. Рассмотрим лишь самые простые способы сохранения и загрузки данных в Octave. Для более детального знакомства с тонкостями управления процессом чтения и записи данных рекомендуется изучить полное руководство GNU Octave [1].

Сохранить/загрузить данные задачи легко и быстро позволяет пара команд `save/load`. Нетрудно догадаться, что команда `save` сохраняет данные. Общий синтаксис вызова команды следующий

`save` *filename varname1 varname2 ... varnameN*

переменные *varname1 varname2 ... varnameN* и их значения сохраняются в файл *filename*. Если файл с именем *filename* уже существует, то он полностью переписывается новыми данными. Первый после ключевого слова `save` идентификатор интерпретируется Octave как имя файла, он должен быть обязательно. В случае отсутствия других идентификаторов, а именно имен переменных *varname1 varname2 ... varnameN* в файл *filename* будут сохранены все переменные, созданные программой.

```
octave:1> A = rand(2,4); b = ones(2,1);
octave:2> save data A b
```

В приведенном примере матрица `A` и вектор `b` сохраняются в файл `data`, который будет создан Octave в текущей директории. Команда `save` записывает данные в определенном формате, например, файл `data` имеет следующее содержание

```
# Created by Octave 3.0.0, Thu Apr 16 11:00:00 2009 VLAST <shamray@box1>
# name: A
# type: matrix
# rows: 2
# columns: 4
 0.1166630486449072 0.8174254215517049 0.8961450653725824 0.3748763513368052
 0.546587944252344 0.6862222302674722 0.2424641069675111 0.5096399183175931
# name: b
# type: matrix
# rows: 2
# columns: 1
 1
 1
```

Под знаком комментария (#) записывается информация о сохраняемых данных (имя переменной, тип, размерность), далее приводится значение переменной. Всей этой информацией можно будет воспользоваться при очередной загрузке сохраненных данных в исполняемый код Octave. Для этого применяют команду `load`, вызов которой аналогичен `save`

`load` *filename varname1 varname2 ... varnameN*

переменные *varname1 varname2 ... varnameN* и их значения загружаются из файла *filename* в программу Octave. Файл с именем *filename* должен существовать в текущей директории. Первый после ключевого слова `save` идентификатор интерпретируется Octave как имя файла, он должен быть обязательно. В случае отсутствия других идентификаторов, а именно имен переменных *varname1 varname2 ... varnameN*, в программу будут загружены все переменные, которые содержатся в в файле *filename*.

```
octave:1> load data
octave:2> A
A =

   0.11666   0.81743   0.89615   0.37488
   0.54659   0.68622   0.24246   0.50964

octave:3> b
b =

   1
   1
```

Для точного управления вводом и выводом в Octave предлагается модель, аналогичная стандартной I/O-библиотеки языка Си.

Вывести результаты в нужном формате, сопровожденные некоторым информационным сообщением позволяет функция `printf`. Синтаксис вызова функции аналогичен Си

`printf(`*fid, template1, template2, ..., templateN*`);`

Аргумент *fid* может содержать два типа информации — это символы, которые непосредственно выводятся на экран, и спецификаторы формата, определяющие, как выводить аргументы *template1, template2, ..., templateN*.

Спецификатор формата начинается с символа `%` за которым следует код формата. Ниже приведены некоторые из возможных спецификаторов.

- `%d` целое десятичное число
- `%f` десятичное число с плавающей запятой *xx.xxxx*
- `%e` десятичное число в виде *x.xx*`e+`*xx*
- `%c` символ
- `%s` строка символов
- `%%` символ %

Кроме спецификаторов формата данных в аргумент *fid* может содержать управляющие символы, например, такие как

- `\n` новая строка
- `\t` горизонтальная табуляция
- `\v` вертикальная табуляция

Пример использования функции `printf` демонстрирует следующая программа, печатающая квадратные корни всех чисел от 5 до 1.

```
for i=5:-1:1
 printf("Квадратный корень из %d равен %f\n", i, sqrt(i));
endfor
```

Альтернативным способом вывести данные является функция `disp(x)`, которая печатает значение аргумента `x`

```
octave:6> disp("Сгенерирована случайная матрица"), disp(rand(2))
Сгенерирована случайная матрица
   0.18649   0.26070
   0.22552   0.73934
```

1.7 Практические задания

Для закрепления пройденного материала рекомендуется самостоятельно выполнить следующие задания.

1. Сгенерировать случайную целочисленную матрицу размерности 5×8 с элементами не меньше 10.
2. Сгенерировать диагональную матрицу с элементами [2 5 8] на второй диагонали сверху и снизу относительно главной диагонали.
3. Сгенерировать квадратную блочную матрицу, состоящую из четырех квадратных блоков девяти случайных чисел от 0 до 1.
4. Сгенерировать случайный вектор столбец из десяти положительных и отрицательных элементов.
5. Сгенерировать вектор-строку состоящую из пятнадцати двоек.
6. Сгенерировать единичный вектор-столбец размерности 8.
7. Сгенерировать нулевой вектор-строку размерности 10
8. Найти максимальный элемент матрицы.
9. Вычислить натуральный логарифм случайной целочисленной матрицы.
10. За одну команду определить является ли некоторая заданная матрица A нулевой.
11. Сгенерировать произвольную матрицу A размерности 15×10 Найти определитель элементов стоящих в последних 10 строках и во всех столбцах.
12. Найти сумму чисел кратных либо только 3, либо только 5 натурального ряда, члены которого не превышают 1000. Программа не должна содержать ни одного цикла и ни одного условного оператора.
13. В одну строчку вычислений найти разность между суммой квадратов первых ста натуральных чисел и квадратом их суммы.

14. В одну строку вычислений найти положительную срезку[2] матрицы A (сгенерировать самостоятельно функцией `randn`).

15. Найти сумму всех четных чисел Фибоначчи[3], не превышающих 4 миллионов. Допустимо использование не более одного цикла, программа не должна содержать ни одного условного оператора.

16. В матрице A найти сумму всех коэффициентов стоящих в четных строках и нечетных столбцах. Программа не должна содержать ни одного цикла и ни одного условного оператора.

17. Последовательность треугольных чисел строится по правилу: член последовательности с номером $p_n = \sum_{i=1}^{n} i$. Например, седьмое треугольное число равно $1+2+3+4+5+6+7 = 28$. Первые десять членов такой последовательности это $1, 3, 6, 10, 15, 21, 28, 36, 45, 55 \ldots$. Найти произведение первых 100 членов треугольной последовательности кратных 77.

18. Построить вектор v, с компонентами v_i такими, что

$$v_i = \begin{cases} n/2, & \text{если } n \text{ – четное число;} \\ 3n+1, & \text{если } n \text{ – нечетное;} \end{cases}$$

где n пробегает значения от 13 до 23. Программа не должна содержать ни одного цикла и условного оператора.

19. Привести квадратную матрицу к нижней треугольной. Программа должна содержать не более одного цикла.

20. Вычислить все диагональные миноры квадратной матрицы A.

21. Написать функцию сложения булевого вектора и 1. На входе — вектор слагаемое, на выходе результирующий вектор. Например на вход 0 0 1, на выход 0 1 0.

22. Используя функцию булевого сложения вычислить все главные миноры[4] произвольной матрицы A. Программа должна содержать не более одного цикла.

Список литературы

[1] GNU Octave: [Электронный ресурс]. Режим доступа: http://www.gnu.org/software/octave/

[2] Числовой инструментарий: [Электронный ресурс]. Режим доступа: http://www.zabaykalsk.ru/docs/lib/other/numeric1.htm

[2] Положительная срезка матрицы $A = (a_{ij} : i = 1, \ldots, m,\ j = 1, \ldots, n)$, это матрица $P = (p_{ij} : i = 1, \ldots, m,\ j = 1, \ldots, n)$, где $p_{ij} = a_{ij}$, если $a_{ij} > 0$ и $p_{ij} = 0$, если $a_{ij} \leq 0$.

[3] Каждый член последовательности Фибоначчи равен сумме двух предыдущих. $F_1 = 1,\ F_2 = 1,\ F_i = F_{i-2} + F_{i-1},\ i = 3, 4, \ldots$

[4] Главный минор, это определитель подматрицы, стоящей на пересечении строк и столбцов с одинаковыми номерами в исходной матрице

2 Построение научной графики в Gnuplot

Пакет Gnuplot представляет собой интерактивную программу, предназначенную для создания графических изображений, задаваемых в виде функций или файлов данных. Управление работой программы осуществляется при помощи команд на специальном языке. Эти команды можно либо вызывать в командной строке Gnuplot, либо загружать из предварительно подготовленного файла. Gnuplot является некомерческим продуктом, работающим на самых разных платформах (например, UNIX, IBM OS/2, MS Windows, DOS, Macintosh, VMS, Atari).

Перечень возможностей Gnuplot весьма широк:

- графики строятся для функций одной или двух переменных;
- функция может быть задана в любой из трех форм: аналитической (формулой), параметрической или табличной (файл данных);
- кривые можно рисовать в декартовой или полярной системе координат;
- графики можно доплнять надписями или вспомогательными линиями (указатели, стрелки и т.п.);
- поддерживаются различные стили линий и шрифтов;
- вывод графика возможен на экран или в файлы различного типа;
- программа позволяет аппроксимировать экспериментальный набор точек кривыми заданного вида.

В данном руководстве предполагается, что работа с пакетом Gnuplot ведется на базе платформы Linux.

Вызвать Gnuplot можно непосредственно из консоли командой `gnuplot`, после чего пользователю будет предложена командная строка, начинающаяся с промпта (приглашения к действию)

```
gnuplot>
```

Теперь можно приступать к созданию графиков функций.

Основными командами начертания графиков в Gnuplot являются `plot` и `splot`. Команда `plot` рисует 2D-графики, то есть графики функций одной переменной. Команда `splot` предназначена для изображения 3D-графиков — графиков функций от двух переменных. Между синтаксисом, правилами вызова и свойствами этих двух команд много общего.

В простейшем случае вызов команды рисования сопровождается заданием функции, график которой необходимо построить

```
plot sin(x)
splot x**2 + y**2
```

По умолчанию в качестве аргумента функции одной переменной выступает переменная x, для функций двух переменных — x и y. При вызове команд `plot` и `splot` можно указать диапазон изменения аргументов функций

```
plot [-2*pi:2*pi] sin(x)
splot [-15:15] x**2 + y**2
splot [-15:15][-5:5] x**2 - y**2
```

Здесь, в первом случае переменная x изменяется в интервале от -2π до 2π; во втором случае переменная x изменяется от -15 до 15, диапозон изменения y выставляется по умолчанию самой программой; в третьем случае переменная x изменяется от -15 до 15, а y — от -5 до 5.

Можно также указать диапозон изменения возможных значений функции. Для команды `plot` это будет всегда второй из интервалов

```
plot [-5:5][0:10] x**2
```

Для команды `splot` диапазон изменения значений функции всегда задает третий интервал

```
splot [-15:15][-5:5][-100:100] x**2 - y**2
```

На одной картинке сразу можно изображать несколько графиков, для этого при вызове команд рисования функции перечисляются через запятую

```
plot [-5*pi:5*pi][-2:2] sin(x)/x, 1/x
```

Сложные функции задаются при помощи применения математических операторов к элементарным функциям. Списки допустимых операторов и некоторых вещественных функций Gnuplot приведены в таблицах 3 и 4 соответственно.

Таблица 3: Операторы Gnuplot

Символ	Пример	Название	Символ	Пример	Название
Унарные операторы					
`-`	`-a`	унапный минус	`+`	`+a`	унарный плюс
`!`	`!a`	логическое отрицание	`!`	`a!`	факториал
Бинарные операторы					
`**`	`a**b`	возведение в спепень	`!=`	`a!=b`	неравенство
`*`	`a*b`	умножение	`<`	`a<b`	меньше
`/`	`a/b`	деление	`<=`	`a<=b`	меньше или равно
`/+`	`a+b`	сложение	`>=`	`a>=b`	больше или равно
`-`	`a-b`	вычитание	`&&`	`a&&b`	логическое И
`==`	`a==b`	равенство	`||`	`a||b`	логическое ИЛИ
Тернарный оператор					
`?:`	`a?b:c`	если `a` истино, то вернуть `b`, иначе вернуть `c`			

В качестве примера сложной функции рассмотрим

$$f(x) = \begin{cases} \sin(x), & \text{при } 0 \le x < 1, \\ \frac{1}{x}, & \text{при } 1 \le x < 2, \\ \text{неопределена}, & \text{в противном случае.} \end{cases}$$

Для построения графика $f(x)$ необходимо набрать следующие команды

```
f(x) = 0<=x && x<1 ? sin(x) : 1<=x && x<2 ? 1/x : 1/0
plot [0:3] f(x)
```

Таблица 4: Библиотека вещестенных математических функций

Мат. функция	Функция Gnuplot	Мат. функция	Функция Gnuplot
$\|x\|$	`abs(x)`	$\sin(x)$	`sin(x)`
округление до целого вверх	`ceil(x)`	$\arcsin(x)$	`asin(x)`
округление до целого вниз	`floor(x)`	$\mathrm{sh}(x)$	`sinh(x)`
целая часть числа	`int(x)`	arcsh(x)	`asinh(x)`
x^a, a – заданное число	`x**a`	e^x	`exp(x)`
$\ln(x)$	`log(x)`	$\lg(x)$	`log10(x)`
$\cos(x)$	`cos(x)`	$\mathrm{tg}(x)$	`tan(x)`
$\arccos(x)$	`acos(x)`	$\mathrm{arctg}(x)$	`atan(x)`
$\mathrm{ch}(x)$	`cosh(x)`	th(x)	`tanh(x)`
arcch(x)	`acosh(x)`	arcth(x)	`atanh(x)`
$\mathrm{arctg}(y/x)$	`atan2(y,x)`		

Отметим, что Gnuplot игнорирует неопределенные значения, то есть последняя ветвь функции $(1/0)$ не даст точек для построения. Также необходимо обратить внимание, что $f(x)$ будет построена как непрерывная функция на разрыве. Для построения разрыва придется создать отдельные функции для двух кусков

...

или использовать параметрическое задание функции

...

Команды `plot` и `splot` строят графики функций заданных таблично. Таблица должна быть сохранена в виде текстового файла, каждая строка которого представляет наборы точек, разделенных пробелами. Вызов команд сопровождается именем файла, заключеным к ковычки (одинарные или двойные). По умолчанию значения в первых трех столбцах такого файла понимаются Gnuplot как координаты (x, y, z) соответственно.

Пусть каждая строка файла `data.1` содержит набор из четырех координат

```
-10.00000000   0.54402111  -0.83907153 100.00000000
 -9.90000000   0.45753589  -0.88919115  98.01000000
 -9.80000000   0.36647913  -0.93042627  96.04000000
 -9.70000000   0.27176063  -0.96236488  94.09000000
 -9.60000000   0.17432678  -0.98468786  92.16000000
 -9.50000000   0.07515112  -0.99717216  90.25000000
 -9.40000000  -0.02477543  -0.99969304  88.36000000
```

и так далее. Команда

```
plot 'data.1'
```

строит график функции $f(x)$, где значение аргумента x берется из первого столбца, а значение функции $f(x)$ — из второго столбца файла `data.1`. Вызов команды

```
splot 'data.1'
```

построит трехмерный график функции $f(x, y)$, где значения аргументов x и y берутся из первого и второго столбцов соответственно, значение функции $f(x, y)$ — из третьего столбца файла `data.1`.

Построить график функциональной зависимости между любыми столбцами файла данных поможет модификатор `using`, после которого указываются номера рассматриваемых столбцов через двоеточие (:)

```
plot "data.1" using 1:4
plot "data.1" using 4:1
splot "data.1" using 4:3:2
```

В первом примере аргумент функции берется из первого солбца файла `data.1`, а значение — из четвертого. Во втором примере наоборот, аргумент — из четвертого столбца, а значение — из первого. В третьем примере значения двух аргументов берутся из четвертого и третьего столбцов, значение функции — из второго. Важно помнить, что при построении двумерного графика (команда `plot`) в модификаторе `using` надо указать ровно два столбца, а при построении трехмерного графика (команда `splot`) — ровно три столбца.

Для каждой функции после ключевого слова with можно указать стиль графика. Это могут быть, например, линии (lines), точки (points), или точки с обозначением ошибки измерения (yerrorbars). Есть и другие стили. Далее можно указать тип линий или точек (lt, pt), толщину линий (lw) и размер точек (ps). Эти же команды позволяют рисовать графики функций, заданных наборами точек, записанных в файл. plot [-5*pi:5*pi] sin(x)/x with lines lt 1 lw 5, 1/x with lines lt 2 lw 1