2. Основы работы в **Octave**

Умение выполнять элементарные математические операции, определять переменные, использовать встроенные функции системы и создавать собственные, работать с массивами данных — это азы работы в системе Octave.

2.1 Элементарные математические выражения в *Octave*

Простейшие арифметические операции в Octave выполняют с помощью следующих операторов:

\+ *сложение*;

− *вычитание*;

* *умножение*;

/ *деление слева направо*;

\ *деление справа налево*;

^ *возведение в степень*.

Вычислить значение арифметического выражения можно, если ввести его в командную строку и нажать клавишу ENTER:

```
>>> 13.5/(0.2+4.2)^2-2.3
ans = -1.6027
```

Листинг 2.1

Обратите внимание, что при вводе вещественных чисел для отделения дробной части используется точка.

Если вычисляемое выражение слишком длинное, то перед нажатием клавиши ENTER следует набрать три или более точек. Это будет означать продолжение командной строки:

```
>>> 1+2*3-4....
>>> +5/6+7....
>>> -8+9
ans = 11.833
```

Листинг 2.2

В первой главе мы уже говорили о значении символа точки с запятой «`;`» в конце выражения. Напомним, что если он указан в конце выражения, то результат вычислений не выводится, а активизируется следующая командная строка:

```
>> 2-1;
>> 2-1
ans = 1
```

Листинг 2.3

2.2 Текстовые комментарии

Правилом хорошего тона в программировании всегда считался «читаемый» программный код, снабженный большим количеством комментарием. Поскольку далее речь пойдет о достаточно сложных вычислениях, для наглядности мы так же будем применять текстовые комментарии.

Текстовый комментарий в Octave это строка, начинающаяся с символа `%`. Использовать текстовые комментарии можно как в командной строке, так и в тексте программы. Строка после символа `%` не воспринимается как команда и нажатие клавиши Enter приводит к активизации следующей командной строки.

Примеры использования текстовых комментариев присутствуют уже в следующем параграфе.

2.3 Представление вещественного числа в *Octave*

Числовые результаты могут быть представлены с плавающей (например, –3.2E-6, 6.42E+2), или с фиксированной (например, 4.12, 6.05, –17.5489) точкой. Числа в формате с плавающей точкой представлены в экспоненциальной форме `mE±p`, где `m` – мантисса (целое или дробное число с десятичной точкой), `p` – порядок (целое число). Для того, чтобы перевести число в экспоненциальной форме к обычному представлению с фиксированной точкой, необходимо мантиссу умножить на десять в степени порядок. Например,

6.42E+2 = 6.42.102 = 642

-3.2E-6 = -3.2.10-6 =-0.0000032

В листинге 2.4 приведен пример ввода вещественного числа.

```
>>> 0.987654321
ans =  0.98765
```

Листинг 2.4

Не трудно заметить, что число знаков в дробной части числа в строке ввода больше чем в строке вывода. Происходит это потому, что результат вычислений выводится в виде, который определяется предварительно установленным форматом представления чисел.

Команда, с помощью которой можно установить формат числа имеет вид:

```
format формат числа;
```

В Octave предусмотрены следующие форматы чисел:

- `Short` – краткая запись, применяется по умолчанию;
- `Long` – длинная запись;

```
>>> format short
>>> pi
ans =  3.1416
>>> format long
>>> pi
ans =  3.14159265358979
```

Листинг 2.5

- `Short E` (`Short e`) – краткая запись в формате с плавающей точкой;
- `Long E` (`Long e`) – длинная запись в формате с плавающей точкой;

```
>>> format short E
>>> pi
ans =  3.1416E+00
>>> format long E
>>> pi
ans =  3.14159265358979E+00
```

Листинг 2.6

- `Short G` (`Short g`) – вторая форма краткой записи в формате с плавающей точкой;
- `Long G` (`Long g`) – вторая форма длинной записи в формате с плавающей точкой;

```
>>> format short G
>>> pi
ans =     3.1416
>>> format long G
>>> pi
ans =     3.14159265358979
```

Листинг 2.7

- `Hex` – запись в виде шестнадцатеричного числа;
- `native-Hex` – запись в виде шестнадцатеричного числа, в таком виде, в каком оно хранится в памяти компьютера;
- `Bit` – запись в виде двоичного числа;
- `native-Bit` – запись в виде двоичного числа, в таком виде, в каком оно хранится в памяти компьютера;

```
>>> format native-hex
>>> pi
ans = 182d4454fb210940
>>> format hex
>>> pi
ans = 400921fb54442d18
>>> format bit
>>> pi
ans =
0100000000001001001000011111101101010100010001000010110100011000
>>> format native-bit
>>> pi
ans =
0001100010110100001000100010101011011111100001001001000000000010
```

Листинг 2.8

- `Bank` – запись до сотых долей;
- `Plus` – записывается только знак числа;

```
>>> format bank
>>> pi
ans = 3.14
>>> format +
>>> pi
ans = +
>>> -pi
ans =
```

Листинг 2.9

- `Free` – запись без форматирования, чаще всего этот формат применяют для представления комплексного числа (подробно о комплексных числах см. п. 2.5.2);

```
>>> format short
>>> 3.1234+2.9876*i
ans = 3.1234 + 2.9876i
>>> format free
>>> 3.1234+2.9876*i
ans = (3.123,2.988)
```

Листинг 2.10

- `Compact` – запись в формате не превышающем шесть позиций, включая десятичную точку, если целая часть числа превышает четыре знака, число будет записано в экспоненциальной форме.

```
>>> format compact
>>> 123.123456
```

```
ans =  123.12
>>> 1234.12345
ans =  1234.1
>>> 12345.123
ans =  1.2345e+04
```

Листинг 2.11

Обратите внимание, что формат `Short` установлен по умолчанию. Вызов команды `format` с другим числовым форматом означает, что теперь вывод чисел будет осуществляться в установленном формате.

2.4 Переменные в *Octave*

В Octave можно определять переменные и использовать их в выражениях. Для определения переменной необходимо набрать `имя переменной`, символ «=» и `значение переменной`. Здесь знак равенства – это *оператор присваивания*, действие которого не отличается от аналогичных операторов языков программирования. То есть, если в общем виде оператор присваивания записать как

`имя переменной = значение выражения`,

то в переменную, имя которой указано слева, будет записано значение выражения, указанного справа.

`Имя переменной` не должно совпадать с именами встроенных процедур, функций и встроенных переменных системы. Система различает большие и малые буквы в именах переменных. То есть `ABC`, `abc`, `Abc`, `aBc` – это имена разных переменных.

Выражение в правой части оператора присваивания может быть числом, арифметическим выражением, строкой символов или символьным выражением. Если речь идет о символьной или строковой переменной, то выражение в правой части оператора присваивания следует брать в одинарные кавычки.

```
>>>%Определение переменной
>>> x=1.2
x = 1.2000
>>> y=3.14
y = 3.1400
>>>%--------------------------------
>>>%Использование переменных в арифметическом выражении
>>> z=x+y
z = 4.3400
>>>%--------------------------------
>>>%Определение символьной переменной
>>> s='a'
s = a
>>>%--------------------------------
>>>%Определение строковой переменной
>>> str='Посадил дед репку. Выросла репка большая, пребольшая'
str = Посадил дед репку. Выросла репка большая, пребольшая
```

Листинг 2.12

Несколько примеров присвоения значений переменным приведено в листинге 2.13 Обратите внимание, что если символ «`;`» в конце выражения отсутствует, то в качествен результата выводится имя переменной и ее значение. Наличие символа «`;`» передает управление следующей командной строке. Это позволяет использовать имена переменных для записи промежуточных результатов в память компьютера.

```
>>> a=1;b=2;c=a*b;d=c^2
d =  4
```

Листинг 2.13

Листинг 2.14 содержит пример использования в выражении ранее неопределенной переменной.

```
>>>%Сообщение об ошибке. Переменная не определена.
>>> t/(a+b) error: 't' undefined near line 37 column 1
```

Листинг 2.14

Если команда не содержи знака присваивания, то по умолчанию вычисленное значение присваивается специальной системной переменной `ans`. Причем полученное значение можно использовать в последующих вычислениях, но важно помнить, что значение `ans` изменяется после каждого вызова команды без оператора присваивания. Примеры использования системной переменной `ans` можно увидеть в листинге 2.15.

```
>> >25-3
ans = 22
>>> %Значение системной переменной равно 22
>>> 2*ans
ans = 44
>> >%Значение системной переменной равно 44
>> x=ans^0.5
x =    6.6332
>> %Значение системной переменной не изменилось и равно 44
>> ans
ans =   44
```

Листинг 2.15

Кроме переменной `ans` в Octave существуют и другие системные переменные:

`ans` – результат последней операции без знака присваивания;

`i, j` – мнимая единица ($\sqrt{-1}$);

`pi` – число π (3.141592653589793);

`e` – число *e* (экспонента 2.71828183)

`inf` – машинный символ бесконечности (∞);

`NaN` – неопределенный результат ($\frac{0}{0}$, $\frac{\infty}{\infty}$, 1^{∞} и т.п.);

`realmin` – наименьшее число с плавающей точкой (2.2251e-308);

`realmax` – наибольшее число с плавающей точкой (1.7977e+308);

Все перечисленные переменные можно использовать в математических выражениях.

Если речь идет об уничтожении определения одной или нескольких переменных, то можно применить команду:

```
clear имя переменной
```

Пример применения команды `clear` приведен с листинге 2.16.

```
 >>> a=5.2;b=a^2;
 >>> a,b
 a = 5.2000
 b = 27.040
 >>> clear a
 >>> a,b
error: 'a' undefined near line 126 column 1
 >>> b
 b = 27.040
```

Листинг 2.16

2.5 Функции в *Octave*

Все функции, используемые в Octave, можно разделить на два класса *встроенные* и *определенные пользователем*.

В общем виде обращение к функции в Octave имеет вид:

```
имя переменной = имя функции(аргумент)
```

или

```
имя функции(аргумент)
```

Если `имя переменной` указано, то ей будет присвоен результат работы функции. Если же оно отсутствует, то значение вычисленного функцией результата присваивается системной переменной `ans`.

Например,

```
>>> x=pi/2; %Определение значения аргумента
>>> y=sin(x)%Вызов функции
y =  1
>>> cos(pi/3)%Вызов функции
ans =  0.50000
```

Листинг 2.17

Рассмотрим элементарные встроенные функции Octave. С остальными будем знакомиться по мере изучения материала.

2.5.1 Элементарные математические функции

Далее приведены элементарные математические функции Octave.

Табл. *2.1.* Тригонометрические функции

Функция	Описание функции
`sin(x)`	синус числа x
`cos(x)`	косинус числа x
`tan(x)`	тангенс числа x
`cot(x)`	котангенс числа x
`sec(x)`	секанс числа x
`csc(x)`	косеканс числа x
`asin(x)`	арксинус числа x
`acos(x)`	арккосинус числа x
`atan(x)`	арктангенс числа x
`acot(x)`	арккотангенс числа x
`asec(x)`	арксеканс числа x
`acsc(x)`	арккосеканс числа x

Примеры работы с тригонометрическими функциями показаны в листинге 2.18.

```
>>> x=pi/7
x = 0.44880
>>> sin(x)
ans = 0.43388
>>> (1-cos(x)^2)^0.5
ans = 0.43388
>>> tan(x)/(1+tan(x)^2)^0.5
ans = 0.43388
>>> (sec(x)^2-1)^0.5/sec(x)
ans = 0.43388
>>> 1/csc(x)
```

```
ans = 0.43388
>>> asin(x)
ans = 0.46542
>>> acos((1-x^2)^0.5)
ans = 0.46542
>>> atan(x/((1-x^2)^0.5))
ans = 0.46542
```

Листинг 2.18

Табл. *2.2.* Экспоненциальные функции

Функция	Описание функции
`exp(x)`	Экспонента числа x
`log(x)`	Натуральный логарифм числа x

Применение экспоненциальных функций:

```
>>> x=1
x =  1
>>> exp(x)
ans =  2.7183
>>> log(x)
ans = 0
>>> log(e^2)
ans =  2
```

Листинг 2.19

Табл. *2.3.* Гиперболические функции

Функция	Описание функции
`sinh(x)`	гиперболический синус числа x
`cosh(x)`	гиперболический косинус числа x
`tanh(x)`	гиперболический тангенс числа x
`coth(x)`	гиперболический котангенс числа x
`sech(x)`	гиперболический секанс числа x
`csch(x)`	гиперболический косеканс числа x

Листинг 2.20 содержит примеры работы с гиперболическими функциями.

```
>>> cosh(x)^2-sinh(x)^2
ans =  1
>>> tanh(x)*coth(x)
ans =  1
```

Листинг 2.20

Табл. *2.4.* Целочисленные функции

Функция	Описание функции
`fix(x)`	округление числа x до ближайшего целого в сторону нуля
`floor(x)`	округление числа x до ближайшего целого в сторону отрицательной бесконечности
`ceil(x)`	округление числа x до ближайшего целого в сторону положительной бесконечности
`round(x)`	обычное округление числа x до ближайшего целого
`rem(x, y)`	вычисление остатка от деления x на y
`sign(x)`	сигнум–функция числа x, выдает 0, если x=0 , –1 при x<0 и 1 при x>0

Примеры работы с тригонометрическими функциями показаны в листинге 2.21.

```
>>> pi
ans =  3.1416
>>> fix(pi)
ans =  3
>>> floor(pi)
ans =  3
>>> floor(-pi)
ans = -4
>>> ceil(pi)
ans =  4
>>> ceil(-pi)
ans = -3
>>> round(pi)
ans =  3
>>> pi/2
ans =  1.5708
>>> round(pi/2)
ans =  2
>>> rem(5,2)
ans =  1
>>> sign(0)
ans = 0
>>> sign(pi)
ans =  1
>>> sign(-pi)
ans = -1
```

Листинг 2.21

Табл. *2.5.* Другие элементарные функции

Функция	Описание функции
`sqrt(x)`	корень квадратный из числа x
`abs(x)`	модуль числа x
`log10(x)`	десятичный логарифм от числа x
`log2(x)`	логарифм по основанию два от числа x
`pow2(x)`	возведение двойки в степень x
`gcd(x, y)`	наибольший общий делитель чисел x и y
`lcm(x, y)`	наименьшее общее кратное чисел x и y
`rats(x)`	представление числа x в виде рациональной дроби

Далее приведены примеры работы с функциям из таблицы 2.5.

```
>>> x=9;
>>> sqrt(x)
ans =  3
>>> abs(-x)
ans =  9
>>> abs(x)
ans =  9
>>> x=10;
>>> log10(x)
ans =  1
>>> log10(10*x)
ans =  2
```

```
>>> x=4;
>>> log2(x)
ans =  2
>>> pow2(x)
ans =  16
>>> x=8;y=24;
>>> gcd(x,y)
ans =  8
>>> lcm(x,y)
ans =  24
>>> rats(pi)
ans =   355/113
>>> rats(e)
ans = 2721/1001
```

Листинг 2.22

2.5.2 Комплексные числа. Функции комплексного аргумента

Рассмотрим реализацию комплексной арифметики в Octave. Как было отмечено выше, для обозначения мнимой единицы зарезервировано два имени – `i`, `j`, поэтому ввод комплексного числа производится в формате:

```
действительная часть + i * мнимая часть
```

или

```
действительная часть + j * мнимая часть
```

Пример ввода и вывода комплексного числа приведен в листинге 2.23.

```
>>> 3+i*5
ans =  3 + 5i
>>> -2+3*i
ans = -2 + 3i
>>> 7+2*j
ans =  7 + 2i
>>> 0+7i
ans =  0 + 7i
>>> 6+0*j
ans =  6
```

Листинг 2.23

Кроме того, к комплексным числам применимы элементарные арифметические операции: +, –, *, \, /, ^ (листинг 2.24).

```
>>> a=-5+2i;b=3-5*i;
>>> a+b
ans = -2 - 3i
>>> a-b
ans = -8 + 7i
>>> a*b
ans =  -5 + 31i
>>> a/b
ans = -0.73529 - 0.55882i
>>> a^2+b^2
ans =   5 - 50i
```

Листинг 2.24

Функции для работы с комплексными числами приведены в таблице 2.6.

Табл. *2.6.* Функции работы с комплексными числами

Функция	Описание функции
`real(Z)`	выдает действительную часть комплексного аргумента Z
`imag(Z)`	выдает мнимую часть комплексного аргумента Z
`angle(Z)`	вычисляет значение аргумента комплексного числа Z в радианах от $-\pi$ до π
`conj(Z)`	Выдает число комплексно сопряженное Z

Примеры использования функций из табл. 2.6 приведены в листинге 2.25.

```
>>> a=-3;b=4;Z=a+b*i
Z = -3 + 4i
>>> real(Z)
ans = -3
>>> imag(Z)
ans =  4
>>> angle(Z)
ans =  2.2143
>>> conj(Z)
ans = -3 - 4i
```

Листинг 2.25

Обратите внимание, что большая часть математических функций, описанных в п. 2.5.1 работают с комплексным аргументом:

```
>>> a=-3;b=4;Z=a+b*i
Z = -3 + 4i
>>> sin(Z)
ans =  -3.8537 - 27.0168i
>>> exp(Z)
ans = -0.032543 - 0.037679i
>>> sqrt(Z)
ans =  1 + 2i
>>> abs(Z)
ans =  5
```

Листинг 2.26

2.5.3 Операции отношения в **Octave**

Операции отношения выполняют сравнение двух операндов и определяют, истинно выражение или ложно (таблица 2.7). Результат операции отношения – логическое значение. В качестве *логических значений* в Octave используются 1 («истина») и 0 («ложь»).

Табл. *2.7.* Операции отношения.

Операция	Описание операции
<	меньше
>	больше
= =	равно
~=	не равно
<=	меньше или равно
>=	больше или равно

2.5.4 Логические выражения в **Octave**

Логическое выражение может быть составлено из операций отношения и логических операций (операторов). Логические выражения выполняются над логическими данными. В

таблице 2.8 представлены основные логические выражения: «и», «или», «не».

Табл. *2.8.* Логические операции

A	B	«не» A	A «и» B	A «или» B	A «исключающее или» B
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	0	1	0

В Octave существует возможность представления логических выражений в виде логических операторов и логических операций (таблица 2.9).

Табл. *2.9.* Виды логических выражений в *Octave*

Тип выражения	Выражение	Логический оператор	Логическая операция
Логическое «и»	`A and B`	`and(A, B)`	`A & B`
Логическое «или»	`A or B`	`or(A, B)`	`A | B`
Исключающее «или»	`A xor B`	`xor(A,B)`	
Отрицание	`not A`	`not (A)`	`~ A`

В таблице 2.9 `A` и `B` – логические выражения (или целочисленные значения 1 и 0).

Логические операторы (выражения) определены и над массивами (матрицами) одинаковой размерности и выполняются поэлементно над элементами. В итоге формируется результирующий массив (матрица) логических значений, каждый элемент которого вычисляется путем выполнения логической операции над соответствующими элементами исходных массивов.

При одновременном использовании в выражении логических и арифметических операций возникает проблема последовательности их выполнения. В Octave принят следующий *приоритет операций*.

1. Логические операторы.
2. Логическая операция ~.
3. Транспонирование матриц, операции возведения в степень, унарный + и −.
4. Умножение, деление
5. Сложение, вычитание.
6. Операции отношения.
7. Логическая операция «и» – `&`
8. Логическая операция «или» – `|`

В листинге 2.27 представлены примеры использования логических операций над скалярными и матричными значениями.

```
>>> A=3;B=4;C=2*pi;
>>> P=~(((A+B)*C)>A^B)|(A+B)==((B-A)+A^2)
P =  1
>>> X=[2 1 6];Y=[1 3 5];
>>> Z=~or((X>2*Y),(X>Y))|and((X>2*Y),(X<Y))
Z =
   0   1   0
```

Листинг 2.27

2.5.5 Функции, определенные пользователем

В первой главе мы уже создавали небольшую программу, которая решала конкретное квадратное уравнение. В этой программе отсутствовал заголовок (первая строка определенного вида) и в нее невозможно было передать входные параметры, то есть это был

обычный список команд, воспринимаемый системой как единый оператор.

Функция, как и программа, предназначена для неоднократного использования, но она имеет входные параметры и не выполняется без их предварительного задания. Функция имеет заголовок вида

```
function name1 [, name2, …] = fun(var1 [, var2, …])
```

где `name1` [, `name2`, …] – список выходных параметров, то есть переменных, которым будет присвоен конечный результат вычислений, `fun` – имя функции, `var1` [, `var2`, …] – входные параметры. Таким образом, простейший заголовок функции выглядит так:

```
function name = fun(var)
```

Все имена переменных внутри функции, а так же имена из списка входных и выходных параметров воспринимаются системой как локальные, то есть эти переменные считаются определенными только внутри функции.

Программы и функции в Octave могут быть созданы при помощи текстового редактора и сохранены в виде файла с расширением .m или .M. Но при создании и сохранении функции следует помнить, что ее имя должно совпадать с именем файла.

Вызов программ в Octave осуществляется из командной строки. Программу можно запустить на выполнение, указав имя файла, в котором она сохранена.

Обращение к функции осуществляется так же, как и к любой другой встроенной функции системы, то есть с указанием входных и выходных параметров. Вызвать функцию можно из командной строки или использовать ее как один из операторов программы.

ПРИМЕР 2.1. Создать функцию для решения кубического уравнения.

Кубическое уравнение

$$ax^3+bx^2+cx+d=0 \qquad (2.1)$$

после деления на *a* принимает канонический вид:

$$x^3+rx^2+sx+t=0 \quad , \qquad (2.2)$$

где

$$r=\frac{b}{a} \quad , \quad s=\frac{c}{a} \quad , \quad t=\frac{d}{a} \quad .$$

В уравнении (2.2) сделаем замену

$$x=y-\frac{r}{3}$$

и получим следующее приведенное уравнение:

$$y^3+py+q=0 \quad , \qquad (2.3)$$

где

$$p=\frac{(3s-r^2)}{3} \quad , \quad q=\frac{2r^3}{27}-\frac{rs}{3}+t \quad .$$

Число действительных корней приведенного уравнения (2.3) зависит от знака дискриминанта $D=(\frac{p}{3})^3+(\frac{q}{2})^3$ (табл. 2.10).

Таблица *2.10.* Количество корней кубического уравнения

Дискриминант	**Количество действительных корней**	**Количество комплексных корней**
D≥0	1	2
D<0	3	-

Корни приведенного уравнения могут быть рассчитаны по формулам Кардано:

$$y_1=u+v \, , \, y_2=\frac{-(u+v)}{2}+\frac{(u-v)}{2}i\sqrt{3} \, , \, y_3=\frac{-(u+v)}{2}-\frac{(u-v)}{2}i\sqrt{3} \qquad (2.4)$$

Здесь

$$u=\sqrt[3]{\frac{-q}{2}+\sqrt{(D)}}, v=\sqrt[3]{\frac{-q}{2}-\sqrt{(D)}} \quad .$$

Далее представлен список команд, реализующий описанный выше способ решения кубического уравнения:

```
function [x1,x2,x3]=cub(a,b,c,d)
r=b/a;
s=c/a;
t=d/a;
p=(3*s-r^2)/3;
q=2*r^3/27-r*s/3+t;
D=(p/3)^3+(q/2)^2;
u=(-q/2+sqrt(D))^(1/3);
v=(-q/2-sqrt(D))^(1/3);
y1=u+v;
y2=-(u+v)/2+(u-v)/2*i*sqrt(3);
y3=-(u+v)/2-(u-v)/2*i*sqrt(3);
x1=y1-r/3;
x2=y2-r/3;
x3=y3-r/3;
```

Листинг 2.28

```
>>> [x1,x2,x3]=cub(3,-2,-1,-4)
x1 =  1.4905
x2 = -0.41191 + 0.85141i
x3 = -0.41191 - 0.85141i
```

Листинг 2.29

Редактирование и отладка файлов описана в первой главе.

2.6 Массивы в *Octave*

Как правило массивы используют для хранения и обработки множества однотипных данных. Вместо создания переменной, для хранения каждого данного, создают один массив, где каждый элемент имеет порядковый номер. Таким образом, *массив* – множественный тип данных, состоящий из фиксированного числа элементов одного типа. Как и любой другой переменной, массиву должно быть присвоено *имя*.

Переменную, представляющую собой просто список данных, называют *одномерным массивом* или *вектором*. Для доступа к данным, хранящимся в определенном элементе массива, необходимо указать *имя массива* и *порядковый номер* этого элемента, называемый индексом.

Если возникает необходимость хранения данных в виде таблиц, в формате строк и столбцов, то необходимо использовать *двумерные массивы* (*матрицы*). Для доступа к данным, хранящимся в таком массиве, необходимо указать *имя массива* и *два индекса*, первый должен соответствовать *номеру строки*, а второй *номеру столбца* в которых хранится необходимый элемент.

Значение *нижней границы индексации* в Octave равно единице. Индексы могут быть только целыми положительными числами или нулем.

Самый простой способ задать одномерный массив в Octave имеет вид

```
имя массива = Xn:dX:Xk
```

где `Xn` – значение первого элемента массива, `Xk` – значение последнего элемента массива, `dX` – шаг, с помощью которого формируется каждый следующий элемент массива, то есть значение второго элемента составит `Xn+ dX`, третьего `Xn+ dX+ dX` и так далее до `Xk`.

Если параметр `dX` в конструкции отсутствует:

```
имя массива = Xn:Xk
```

это означает, что по умолчанию он принимает значение равное единице, то есть каждый следующий элемент массива равен значению предыдущего плюс один.

Примеры создания массивов приведены в листинге 2.31.

```
>>> A=1:5
A =
1 2 3 4 5
>>> B=2:2:10
B =
2 4 6 8 10
>>> xn=-3.5;xk=3.5;dx=0.5;
>>> X=xn:dx:xk
X =
Columns 1 through 8:
-3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0
Columns 9 through 15:
0.5 1.0 1.5 2.0 2.5 3.0 3.5
```

Листинг 2.30

Переменную заданную как массив можно использовать в арифметических выражениях и в качестве аргумента математических функций. Результатом работы таких операторов являются массивы (листинг2.31).

```
>>> xn=-3.5;xk=3.5;dx=0.5;
>>> X=xn:dx:xk;
>>> Y=cos(X/2)
Y =
 Columns 1 through 7:
-0.1782 0.0707 0.3153 0.5403 0.7316 0.8775   0.9689
 Columns 8 through 14:
1.0 0.9689 0.8775 0.7316 0.5403 0.3153 0.0707 -0.1782
>>> B=2:2:10;C=sqrt(B)
C =
   1.4142   2.0000   2.4495   2.8284   3.1623
>>> -2:2
ans =
  -2  -1   0   1   2
>>> ans*2-pi/2
ans =
   -5.5708   -3.5708   -1.5708    0.4292    2.4292
```

Листинг 2.31

Векторы и матрицы в Octave можно вводить поэлементно. Так для определения *вектора–строки* следует ввести имя массива, а затем после знака присваивания, в квадратных скобках через пробел или запятую перечислить элементы массива:

```
>>> x=[2 4 6 8 10]
x =     2    4    6    8   10
>>> y=[-1.2 3.4 -0.8 9.1 5.6 -7.3]
y = -1.20000   3.40000  -0.80000   9.10000   5.60000  -7.30000
```

Листинг 2.32

Элементы *вектора–столбца* вводятся через точку с запятой:

```
>>> X=[1;3;5;7;9]
X =
   1
```

```
    3
    5
    7
    9
```

Листинг 2.33

Обратиться к элементу вектора можно, указав имя массива и порядковый номер элемента в круглых скобках:

```
>>> x=[2 4 6 8 10];
>>> y=[-1.2 3.4 -0.8 9.1 5.6 -7.3];
>>> x(1)%значение первого элемента массива x
ans =  2
>>> y(5)%значение пятого элемента массива y
ans =  5.6000
>>> x(1)/2+y(3)^2-x(4)/y(5)
ans =  0.21143
```

Листинг 2.34

Ввод элементов матрицы так же осуществляется в квадратных скобках, при этом элементы строки отделяются друг от друга пробелом или запятой, а строки разделяются между собой точкой с запятой. *Обратиться к элементу матрицы* можно, указав после имени матрицы, в круглых скобках, через запятую, номер строки и номер столбца на пересечении которых элемент расположен. Примеры задания матриц и обращение к их элементам показаны в листинге 2.35.

```
>>> M=[2 4 6;1 3 5;7 8 9]
M =
   2   4   6
   1   3   5
   7   8   9
>>> M(1,2)
ans =  4
>>> M(3,1)
ans =  7
>>> M(2,2)/2+M(3,3)^0.5-M(1,1)*5
ans = -5.5000
```

Листинг 2.35

Подробно работа с векторами и матрицами описана в пятой главе.

2.7 Символьные вычисления в *Octave*

Символьные вычисления в Octave поддерживает специальный пакет расширений **octave-symbolic**. Процедура установки пакетов расширений описана в первой главе. Если пакет уже установлен, то пред началом работы его нужно загрузить командой

```
pkg load symbolic
```

Теперь можно использовать любые функции из пакета **symbolic**.

Оператор `symbols` *инициализирует символические операции*, с этого оператора должны начинаться любые действия в символьных переменных.

Работа с символьными переменными в Octave требует их специального *объявления*:

```
sym('имя переменной')
```

Например, команда `x = sym ("x")` объявляет символьную переменную `x`.

ПРИМЕР 2.2. Выполнить арифметические операции с символьными переменными $z = x \cdot y, t = \frac{x^3}{z}$, где $x = a + b, y = a^2 - b^2$ (листинг 2.36).

```
>>>%Объявление символьных переменных
>>> x = sym ("x");
>>> y = sym ("y");
>>> z = sym ("z");
>>> t = sym ("t");
>>> a = sym ("a");
>>> b = sym ("b");
%Вычисление символьных выражений
>>> x=a+b
x =
a+b
>>> y=a^2-b^2
y =
-b^(2.0)+a^(2.0)
>>> z=x*y
z =
-(b^(2.0)-a^(2.0))*(a+b)
>>> t=x^3/z
t =
-(b^(2.0)-a^(2.0))^(-1)*(a+b)^(2.0)
```

Листинг 2.36

Символьные вычисления в Octave предусматривают работу с элементарными математическими функциями (таблица 2.11).

Табл. *2.11.* Функции в символьных вычислениях

Функция	Описание функции
Sin(x)	синус числа x
Cos(x)	косинус числа x
Tan(x)	тангенс числа x
aSin(x)	арксинус числа x
aCos(x)	арккосинус числа x
aTan(x)	арктангенс числа x
Log(x)	натуральный логарифм числа x
Exp(x)	экспонента числа x (e^x)
Sqrt(x)	корень квадратный из числа x ($\sqrt{x}$)
Pi	число π

Вычислить *значение символьного выражения* при заданном значении переменной можно с помощью функции

```
subs(выражение,имя переменной,значение переменной)
```

ПРИМЕР 2.3. Вычислить значение выражения $y=\sin(\alpha)^2-\cos(\alpha)^2$, при $\alpha_1=\frac{\pi}{3}, \alpha_2=\frac{\pi}{6}$ (листинг 2.37).

```
>>> x = sym ("x") ;
>>> y = sym ("y") ;
>>> y=Sin(x)^2-Cos(x)^2
y =
-cos(x)^(2.0)+sin(x)^(2.0)
%Значение выражения при заданном значении переменной
>>> subs(y,x,Pi/3)
```

```
ans =
0.50000000000000001
>>> subs(y,x,Pi/6)
ans =
-0.49999999999999999
```

Листинг 2.37

Преобразовать символьное выражение, представить его в виде элементарных функций, возможно командой

```
expand(выражение)
```

ПРИМЕР 2.4. Раскрыть скобки в выражении $y=(\sqrt{x}+1)(\sqrt{x}-1)+(x-1)^3$ (листинг 2.28).

```
>>> y=(Sqrt(x)+1)*(Sqrt(x)-1)+(x-1)*(x-1)*(x-1)
y =
(-1.0+x)^3+(-1.0+sqrt(x))*(1.0+sqrt(x))
>>> expand(y)
ans =
-2.0+(4.0)*x+x^3-(3.0)*x^2
```

Листинг 2.38

Далее, по ходу изложения материала, будут рассмотрены операции с матрицами символов, решение систем линейных уравнений в символьных переменных (п. 5.9), решение нелинейных уравнений и систем (п. 7.4), дифференцирование (п. 8.1).