Оглавление

Глава 1

Основы

1.1 Форматы отображения

Для начала, стоит отметить, что по умолчанию включен формат short В Octave есть следующие форматы отображения чисел:

- short

```
octave:1> format short
octave:2> pi
ans = 3.1416
```

- long

```
octave:3> format long
octave:4> pi
ans = 3.141592653589793
```

- short e — краткая запись в формате с плавающей точкой

```
octave:5> format short e
octave:6> pi
ans = 3.1416e+00
```

- long e — длинная запись в формате с плавающей точкой

```
octave:7> format long e
octave:8> pi
ans = 3.141592653589793e+00
```

- short g — вторая форма записи в формате с плавающей точкой

```
octave:10> format short g
octave:11> pi
ans = 3.1416
```

- long g — вторая форма записи в формате с плавающей точкой

```
octave:12> format long g
octave:13> pi
ans = 3.141592653589793
```

- Hex — запись в виде шестнадцетиричного числа

```
octave:14> format hex
octave:15> pi
ans = 400921fb54442d18
```

- native-hex — запись в шестнадцетиричного числа, в таком виде, в каком оно хранится в памяти компьютера

```
octave:16> format native-hex
octave:17> pi
ans = 182d4454fb210940
```

- bit — запись в виде двочиного числа

```
octave:18> format bit
octave:19> pi
ans = 01000000000100100100001111111011010101...
```

- native-bit — запись в виде двочиного числа, в таком виде, в каком оно хранится в памяти компьютера

```
octave:21> format native-bit
octave:22> pi
ans = 00011000101101000010001000101010110...
```

- bank — запись до сотых долей

```
octave:23> format bank
octave:24> pi
ans = 3.14
```

- plus — записывается только знак числа

```
octave:25> format plus
octave:26> pi
ans = +
```

- free — запись без форматирования, чаще всего этот формат применяют для представления комплексного числа

```
octave:27> format short
octave:28> 3.1234+2.9876*i
ans =  3.1234 + 2.9876i
octave:29> format free
octave:30> 3.1234+2.9876*i
ans = (3.1234,2.9876)
```

- 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 comact
octave:35> 123.123456
ans = 123.123
octave:36> 1234.12345
ans = 1234.12
octave:37> 12345.123
ans = 12345.1
```

1.2 Элементарные функции

1.2.1 Целочисленные функции

- $fix(x)$ — округление числа x до ближайщего целого в сторону нуля
- $floor(x)$ — округление числа x до ближайщего целогов сторону отрицательной бесконечности
- $ceil(x)$ — округление числа x до ближайщего целого в сторону положительной бесконечности
- $round(x)$ — округление числа x до ближайщего целого
- $rem(x, y)$ — вычисление остатка от деления x на y
- $sign(x)$ — возвращает 0, если $x = 0$, -1 при $x < 0$ b 1, если $x > 0$

1.2.2 Тригонометрические функции

- $sin[d](x)$ — синус числа x в радианах
- $cos[d](x)$ — косинус числа x
- $tan[d](x)$ — тангенс числа
- $cot[d](x)$ — котангенс числа
- $sec[d](x)$ — секанс числа
- $csc[d](x)$ — косеканс числа
- $asin[d](x)$ — арксинус числа
- $acos[d](x)$ — арккосинус числа
- $atan[d](x)$ — арктангенс числа
- $acot[d](x)$ — арккотангенс числа
- $asec[d](x)$ — арксеканс числа
- $acsc[d](x)$ —арккосеканс числа

1.2.3 Экспоненциальные функции

- $exp(x)$ — экспонент числа x
- $log(x)$ — НАТУРАЛЬНЫЙ логарифм числа x

1.2.4 Гиперболические функции

- $sinh(x)$ — гиперболический синус x
- $cosh(x)$ — гиперболический косинус x
- $tanh(x)$ — гиперболический тангенс x
- $coth(x)$ — гиперболический котангенс x
- $sech(x)$ — гиперболический секанс x
- $csch(x)$ — гиперболический косеканс числа x

1.2.5 Другие полезные и простые элементарные функции

- $sqrt(x)$ — квадратный корень из числа x
- $abs(x)$ — модуль числа x
- $log10(x)$ — десятичный логарифм от числа x
- $log2(x)$ — логарифм по основанию два из 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

1.2.6 Обработка строк

- $char(code)$ — возвращает символ по его коду (Например у буквы d код равен 100). Можно подставить промежуток кодов и получить буквы с кодами этого промежутка
- $deblank(s)$ — возвращает новую строку удаляя пробелы в конце старой строки
- $int2str(x)$ — возвращает строку из числа x или, если x – массив, то строку составленную из чисел, хранящихся в x (числа будут преобразованы к целому типы)
- $findstr(str, substr)$ — возвращает номер позиции, начиная с которой подстрока $substr$ входит в строку str
- $lower(s)$ — возвращает строку, где все буквы строки s заменены на строчные
- $upper(s)$ — возвращает строку s, преобразованную к прописным буквам
- $mat2str(x, n)$ — преобразовывает числовую матрицу x в строку; если присутствует необязательный параметр n, то перед преобразованием в строку все элементы округляются до n значащих цифр
- $num2str(x, n)$ — преобразовывает матрицу (массив) x в массив символов, если присутствует необязательный параметр n, то перед преобразованием в строку все элементы матрицы округлятся до n значащих цифр

- $sprintf(format, x)$ — формирует строку в соотвествии с заданным $format$, одно и то же что и сишный sprintf
- $sscanf(s, format)$ — возвращает из строки s числовое значение в соотвествии с форматом. Тот же сишный sscanf
- $str2double(s)$ — формирует числа из строки s, если это возможно
- $str2num(s)$ — формирование массива числе из строки, если это возможно
- $strcat(s1, s2, ..., sn)$ — объединяет строки
- $strcmp(s1, s2)$ — возвращает 1, если строки одинковы, 0 – в противном случае
- $strcpmi(s1, s2)$ — сравнение строк, не различая строчные и прописные буквы
- $strjust(s, direction)$ — выравенивание строки s в соотвествии с напрвлением $direct$: $right$ – по правому краю, $left$ – по левому краю, $center$ – по центру
- $strncmp(s1, s2, n)$ — сравнение первых n символов строк. Возвращает 1, если первые n сиволов строк совпадают, 0 – в противном случае
- $strrep(s, subs, subsnew)$ — формирует новую строку из строки s путем замены подстрок $subs$ на подстроки $subsnew$
- $strtok(s, delimiter)$ — поиск первой подстроки в строке s, отделенной пробелом или символом табуляции (при отсутствии параметра $delimiter$) или превой подстроки отделенной от s одним из символов, входящих в $delimiter$. Функция может возвращать 2 параметра: первый – найденная подстрока, второй – содержит остаток строки s после $strtok$

1.3 Комплексные числа. Функции комплексного аргумента

Форма записи:

1. действительная часть $+ i *$ мнимая часть
2. действительная часть $+ j *$ мнимая часть

Пример:

```
>> 3+i*5
ans =  3 + 5i
>> 7+2j
ans =  7 + 2i
>> 7+0j
ans = 7
>> 5+j3
error: 'j3' undefined near line 1, column 1
```

Также, можно применять элементарные арифметические операции $+, -, *, backslash, /, \hat{}$.

```
>> a = -5+2i;
>> b = 3-5i;
>> a+b
ans = -2 - 3i
>> a-b
ans = -8 + 7i
>> a*b
ans =  -5 + 31i
>> a/b
ans = -0.7353 - 0.5588i
>> a^2+b^2
ans =   5 - 50i
```

1.3.1 Функции для работы с комплексными числами

- $real(z)$ — возвращает действительную часть комплексного аргумента z
- $imag(z)$ — возвращает мнимую часть комплексного аргумента z
- $angle(z)$ — вычисляет значение аргумента комплексного числа z в радианах от $-\pi$ до π
- $conj(z)$ — возвращает числа, комплексно сопряженное z

Пример:

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

1.4 Логические операторы

- $<$ — меньше
- $>$ — больше
- $==$ — равно
- $=$ — не равно (также есть $!=$, но в Matlab оно не работает)
- $<=$ — меньше или равно
- $>=$ — больше или равно

Логичесие операции и отношения:

- $and(a, b)$ или $a\&b$ — логическое и
- $or(a, b)$ или $a|b$ — логическое или
- $xor(a, b)$ — исключающее или
- $not(a)$ или $\ a$ — отрицание

1.5 Функции определенные пользователем

1.5.1 Базовые формы записи функции

Функция, которая ничего не возвращает и не получает никаких аргументов

```
function <function-name>
  <body>
end[function]
```

Квадратные скобки обозначают, что можно написать как и $endfunction$, так и просто end. Дальше, для краткости я буду писать просто end. Стоит отметить, что в Octave принимается как и $endfunction$ ($endfor$ и др.), так и end, но в Matlab синтаксически верно только end.

Для того чтобы определить функцию, которая принимает некоторые аргументы, есть такая запись:

```
function <function-name>(<arg-list>)
  <body>
end
```

Для того чтобы функция возвращала, некоторое значение существует такая форма:

```
function <ret-value> = <function-name>(<arg-list>)
  <body>
end
```

1.5.2 Продвинутые техники

Функция, которая принимает переменное кол-во аргументов

Для того чтобы определить функцию, которая получает переменное кол-во аргументов, стоит использовать специальный параметр *varargin*:

```
function <ret-value> = <function-name>(varargin)
  <body>
end
```

Пример:

```
function print_arguments(varargin)
  for i = 1 : length(varargin)
      printf("Input argument %d: ", i);
      disp(varargin{i});
  end
end
```

Эта функция может такое:

```
print_arguments(1, [1, 2], "hi")
Input argument 1: 1
Input argument 2:    1   2
Input argument 3: hi
```

Функция, которая возвращает несколько значений

```
function [<ret-list>] = <function-name> (<arg-list>)
  body
end
```

В этом, примере квадратные скобки обозначают, массив возвращаемых значений, они обязательны! Пример:

```
function [max, idx] = vmax (v)
  idx = 1;
  max = v(idx);
  for i = 2 :length(v)
      if(v(i) > max)
          max = v(i);
          idx = i;
      endif
  endfor
endfunction
```

Функции с переменным числом возвращаемых значений

Для того чтобы функция имела переменная кол-во возвращаемых значений, надо применить специальный параметр - *varargout*

```
function varargout = <function-name>(<arg-list>)
  <body>
  end
```

Пример:

```
function varargout = one_to_n()
  for i = 1 : nargout
      varargout{i} = i;
  endfor
endfunction
```

Используем:

```
>> [a,b,c] = one_to_n()
a = 1
b = 2
c = 3
>> [a,b,c,d] = one_to_n()
a = 1
b = 2
c = 3
d = 4
```

1.5.3 Общая форма записи функции

Если обобщить всё, то в общем объявление функции выглядит так:

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

1.6 Массивы и матрицы

Самый простой способ создать массив это задание промежутка: $start : step : end$, где *start* – начало промежутка, *step* – шаг, *end* – конец промежутка.

```
>> A = 1 : 5
A =
1   2   3   4   5

>> A = 1 : 0.5 : 3
A =
1.0000    1.5000    2.0000    2.5000    3.0000
```

1.7 Символьные вычисления

Для начала нужно установить пакет symbolic для Octave, дальше загрузить его командой *pkg load symbolic*. Для того чтобы создать символьную переменную надо их объявить, например *x = sym('x')* или *syms x*. Пример:

```
>> x = sym("x")
x = (sym) x
>> y = sin(x)^2 - cos(x)^2
y = (sym)

 2         2
sin (x) - cos (x)

>> subs(y, x, pi)
ans = (sym) -1
>> y = x^2
```

```
y = (sym)

2
x

>> subs(y, 10)
ans = (sym) 100
```

Стоить заметить, что пакет symbolic после subs возвращает значения типа sym, это значит, что его в дальнейшим надо привести к численному типу, для этого можно использовать eval или double

Глава 2

Полезные функции

2.1 Символьные вычисления

2.1.1 @sym/double

Ссылка на страницу в документации: https://octave.sourceforge.io/symbolic/function/@sym/double.html
Конвертирует символьное выражение в double

```
x = sym(1) / 3
⇒ x = (sym) 1/3
double (x)
⇒ ans =  0.3333
```

Не смотря на название, может конвертировать комплексное символьное выражение в число:

```
z = sym(4i) - 3;
double (z)
⇒ ans = -3 + 4i
```

2.1.2 @sym/eval

Ссылка на страницу в документации: https://octave.sourceforge.io/symbolic/function/@sym/eval.html
sym: eval — Приводит символьное выражение к double, может брать переменные из рабочего простанства. Примеры: Без выражений, которые не содержат символов, *eval* делает то же самое что и *double*

```
f = 2*sin(sym(3))
⇒ f = (sym) 2·sin(3)
eval(f)
⇒ ans = 0.2822
double(f)
⇒ ans = 0.2822
```

```
syms x y
f = x*sin(y)
⇒ f = (sym) x·sin(y)
```

```
x = 2.1
⇒ x = 2.1000
y = 2.9
⇒ y = 2.9000

f
⇒ f = (sym) x·sin(y)

eval(f)
⇒ ans = 0.5024
```

Глава 3

Ссылки

Откуда была взята информация:

- https://intuit.ru/studies/courses/3677/919/info
- https://octave.org/doc/v6.2.0/
- https://octave.sourceforge.io/symbolic/overview.html