Математические пакеты

Программирование в GNU Octave

Сучков Андрей Игоревич

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

12 сентября 2020 г.

В предыдущей серии...

- Введение в GNU Octave
- Философия GNU Octave
- Синтаксис
- Работа с векторами и матрицами
- Индексация
- ...

Структура как тип данных

Определение

Структура – это совокупность переменных, объединенных одним именем

Листинг 1: Создание структуры

```
z = struct ("Re", 4, "Im", -3);
z.Re = 3;

## Another way
x.a = 1;
x.b = [1, 2; 3, 4];
x.c = "string";
x.("blah blah") = false (1, 3);
```

Массив ячеек (списки)

Определение

Списки – это структура, позволяющая хранить элементы различных типов

Листинг 2: Создание списков

```
c = {5, [-6, 9], "Hello!", {-5:0, true, "Octave"}};

c1 = cell (1, 3);  # return empty list 1x3

c{1}  # return 5
c(1)  # return cell {5}

[a, b] = c{[1, 3]}  # return:
                    #    a = 5
                    #    b = "Hello!"

x = [c{[1, 2, 1, 1]}]  # return [5, -6, 9, 5, 5]
```

Программирование

- Программирование осуществляется не только в консоли, но и в M-файлах (файлы с расширением `.m`)
- M-файлы в Octave бывают двух типов: скрипты (Script M-Files) и функции, определенные пользователем (Function M-Files)
- Скрипт – это файл-программа, содержащая последовательность команд

Программирование

Пример скрипта

Листинг 3: Файл example1.m

```
1 a = 5; b = 7;
2 c = a + b;
3 a++;
4 lcm (a, c)
```

Командное окно

```
>> example1
ans =  12
>>
```

Функции, определённые пользователем

Всякая функция, определяемая пользователем имеет вид:

```
function [ret_var] = func_name (arg_list)
  # function-body
endfunction
```

При этом входные параметры `arg_list` и возвращаемые параметры `ret_var` могут отсутствовать.

Важно!

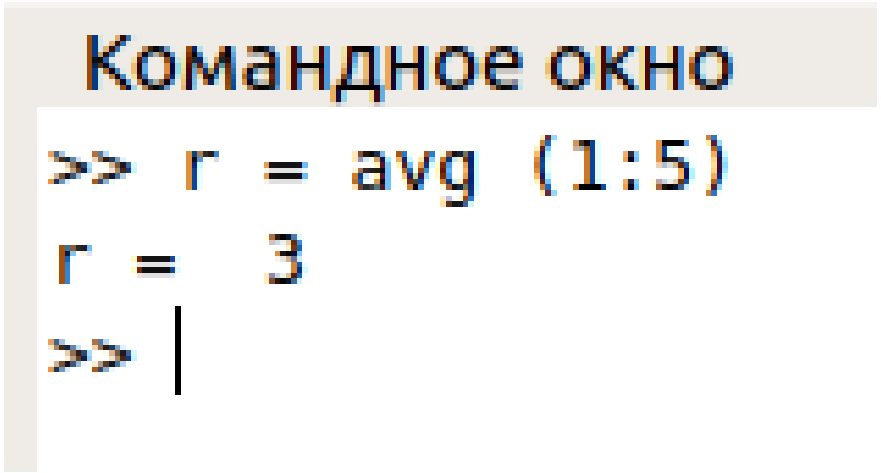
Имя function-файла должно совпадать с именем основной функции!

Функции, определённые пользователем

Пример стандартной функции

Листинг 4: Файл avg.m

```
1 function retval = avg (v)
2   retval = sum (v) / length (v);
3 endfunction
```



Функции, определённые пользователем

Пример функции с несколькими возвращаемыми параметрами

Листинг 5: Файл squ.m

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

Командное окно

```
>> [x1, x2] = squ (1, 1, -2)
x1 =  1
x2 = -2
>> [x1, x2] = squ (1, 2, 2)
x1 = -1 + 1i
x2 = -1 - 1i
>>
```

Условный оператор

Листинг 6: Стандартный оператор

```
if (condition)
  # then-body
else
  # else-body
endif
```

Листинг 7: Сокращённый оператор

```
if (condition)
  # then-body
endif
```

Условный оператор

Продолжение

Листинг 8: Расширенный оператор

```
if (condition)
  # then-body
elseif (condition)
  # elseif-body
else
  # else-body
endif
```

Листинг 9: Тернарный оператор

```
merge (mask, tval, fval);
ifelse (mask, tval, fval);  # equivalent
```

Оператор альтернативного выбора

```
switch (expression)
  case label1
    # command-list
  case label2
    # command-list
  case {label31, label32, ..., label3n}
    # command-list
  ...
  otherwise
    # command-list
endswitch
```

Оператор альтернативного выбора

Преимущество switch перед if-else

Листинг 10: Сравнение строк (опасный вариант)

```
## This is NOT valid
if (str == "GNU Octave")
  f1 ();
elseif (str == "R")
  f2 ();
elseif (str == "SageMath")
  f3 ();
else
  f4 ();
endif
```

Листинг 11: Сравнение строк (лучший вариант)

```
switch (str)
  case "GNU Octave"
    f1 ();
  case "R"
    f2 ();
  case "SageMath"
    f3 ();
  otherwise
    f4 ();
endswitch
```

Цикл с предусловием и постусловием

Листинг 12: Цикл с предусловием

```
while (condition)
  # loop-body
endwhile
```

Листинг 13: Цикл с постусловием

```
do
  # loop-body
until (condition)
```

Оператор цикла с известным числом повторений

Листинг 14: Общий вид

```
for var = expression
  # loop-body
endfor
```

Листинг 15: Часто используемый вариант

```
for i = i1:di:im
  # loop-body
endfor
```

Оператор цикла с известным числом повторений

Цикл for для матриц и списков

Листинг 16: Цикл for для матриц

```
## Loop over a matrix
for i = [1, 3; 2, 4]
  disp ("i = ")
  disp (i)
endfor
## Print columns
```

Листинг 17: Цикл for для списков

```
## Loop over a cell array
for i = {1, "two"; "three", 4}
  disp ("i = ")
  disp (i)
endfor
## Also print columns
```

Оператор цикла с известным числом повторений

Цикл for для структур

Листинг 18: Общий вид

```
for [val, key] = expression
  # loop-body
endfor
```

Листинг 19: Пример

```
x = struct ("a", pi,
            "b", 1:5,
            "c", "GNU's Not UNIX");
for [val, key] = x
  printf ("x.%s = ", key)
  disp (val)
endfor
```

Операторы прерывания итераций в циклах

Оператор break

Листинг 20: Оператор break

```
num = 103;
div = 2;

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

if (rem (num, div) == 0)
  printf ("Smallest divisor of %d is %d\n", num, div)
else
  printf ("%d is prime\n", num)
endif
```

Операторы прерывания итераций в циклах

Оператор continue

Листинг 21: Оператор continue

```
## Print elements of a vector of random
## integers that are even.

## First, create a row vector of 10 random
## integers with values between 0 and 100:

vec = round (rand (1, 10) * 100);

## Print what we're interested in:

for x = vec
  if (rem (x, 2) != 0)
    continue
  endif
  printf ("%d\n", x)
endfor
```

Функции с неограниченным числом входных элементов

Листинг 22: Использование varargin

```
1 function val = func_name (varargin)
2   # func-body
3 endfunction
```

Листинг 23: Файл smallest.m

```
1 function val = smallest (varargin)
2   val = min ([varargin{:}]);
3 endfunction
```

Командное окно

```
>> s = smallest (1, 2, 3, 4, 5, -4, 2)
s = -4
>>
```

Функции с неограниченным числом входных элементов

Продолжение

Листинг 24: Файл print_arguments.m

```
1 function print_arguments (varargin)
2   for i = 1:nargin  # or: length (varargin)
3     printf ("Input argument %d: ", i)
4     disp (varargin{i})
5   endfor
6 endfunction
```

Командное окно

```
>> print_arguments (1:5, 8, 10, -3)
Input argument 1:    1   2   3   4   5
Input argument 2:  8
Input argument 3:  10
Input argument 4: -3
>>
```

Функции с неограниченным числом возвращаемых элементов

Листинг 25: Файл one_to_n.m

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

Командное окно

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

Игнорирование аргументов

Листинг 26: Игнорирование входного аргумента

```
1 function val = pick2nd (~, arg2)
2   val = arg2;
3 endfunction
```

Игнорирование возвращаемых аргументов

Командное окно

```
>> x = squ (1, 1, -2)
x =  1
>> [~, y] = squ (1, 1, -2)
y = -2
>>
```

Оператор return

Листинг 27: Использование оператора return

```
1 function [a, b, c] = foo ()
2   a = 1;
3   c = 3;
4   return
5   b = 2;
6 endfunction
```

Командное окно

```
>> [a, b, c] = foo ()
warning: foo: some elements in list of return values are undefined
warning: called from
    foo at line 4 column 3
a =  1
b = [](0x0)
c =  3
>> [a, ~, c] = foo ()
a =  1
c =  3
>>
```

Параметры по умолчанию

Листинг 28: Обновлённый файл squ.m

```
function [x1, x2] = squ (a = 1, b, c = 0)
  D = b ^ 2 - 4 * a * c;

  if (isargout (1))
    x1 = (-b + sqrt (D)) / (2 * a);
  endif

  if (isargout (2))
    x2 = (-b - sqrt (D)) / (2 * a);
  endif
endfunction
```

Параметры по умолчанию

Результат работы программы

Командное окно

```
>> [x1, x2] = squ (2, 4)
x1 = 0
x2 = -2
>> [x1, x2] = squ (:, 4, :)
x1 = 0
x2 = -4
>>
```

Подфункции

- Помимо основной функции, function-файл может содержать *подфункции* (subfunction)
- Они имеют такой же синтаксис, как и основные функции
- Подфункции могут быть вызваны из основной функции или из других подфункций, но не за пределами function-файла
- Подфункции располагаются после основной функции

Подфункции

Пример подфункции

Листинг 29: Файл f.m

```
function f ()
## Main function
  printf ("in f, calling g\n")
  g ()
endfunction

function g ()
## First subfunction
  printf ("in g, calling h\n")
  h ()
endfunction

function h ()
## Second subfunction
  printf ("in h\n")
endfunction
```

Подфункции

Подфункции в script-файлах

Листинг 30: Использование подфункций в script-файлах

```
## Prevent Octave from thinking that this
## is a function file:
1;

function foo ()
  do_something ();
endfunction

function do_something ()
  do_something_else ();
endfunction

## Script body
printf ("Hello, world!\n")
x = 1:10;
foo ();
```

Вложенные функции

- *Вложенные функции* (nested functions) схожи с подфункциями в том, что только основная функция видна вне файла
- Однако вложенные функции имеют доступ к переменным родительской функции и могут изменять их
- По возможности рекомендуется использовать подфункции вместо вложенных функций

Видимость вложенных функций

Листинг 31: Родительская функция с дочерними функциями

```
function ex_top ()
  ## Can call: ex_top, ex_a, ex_b

  function ex_a ()
    ## Can call everything
    function ex_aa ()
      ## Can call everything
    endfunction

    function ex_ab ()
      ## Can call everything
    endfunction
  endfunction

  function ex_b ()
    ## Can call: ex_top, ex_a, and ex_b
  endfunction
endfunction
```

Дескриптор функции

- Дескриптор функции является указателем на другую функцию
- Дескрипторы функций используются для косвенного вызова других функций или в качестве аргумента другой функции
- Дескрипторы имеют тип данных *function_handle*
- Для создания дескриптора используется символ @

Дескриптор функции

Пример использования

Листинг 32: Работа с дескриптором

```
1 f = @sin;  # sine function descriptor
2
3 f (pi/6)  # return 0.5
4
5 quad (f, 0, pi)  # return 2
```

Анонимные (безымянные) функции

- Анонимная функция является функцией, которая не сохранена в программном файле, но сопоставлена с переменной
- Анонимные функции также имеют тип данных *function_handle*
- Анонимные функции могут принимать входные параметры и возвращать выходные параметры, как стандартные функции
- В отличие от функций в программном файле, анонимная функция может содержать **только один** исполняемый оператор
- Для создания анонимной функции используется символ `@`

Анонимные (безымянные) функции

Примеры

Листинг 33: Работа с анонимными функциями

```
f = @(x, y) sin (x) + cos (y) .^ 2;
f (pi/2, pi/4)  # return 1.5

quad (@(x) sin (x), 0, pi)  # return 2
quad (@sin, 0, pi)          # also return 2, but this better

a = 42;
g = @(x) x + a;
g (10)  # return 52
a = 10;
g (10)  # anyway return 52

h = {@(x) x .^ 2
     @(x, y) x * y - 10};
h{1} (5)      # return 25
h{2} (3, 2)   # return -4
```

Анонимные (безымянные) функции

Анонимная функция как входной параметр

Листинг 34: Файл newton.m

```
function x = newton (f, x0)
  dx = 1e-6;
  x = x0;
  x0++;

  while (abs (x - x0) > eps)
    x0 = x;
    df = (f (x+dx) - f (x-dx)) / (2 * dx);
    x = x0 - f (x) ./ df;
  endwhile
endfunction
```

Листинг 35: Результат работы программы

```
>> newton (@(x) x .^ 2 - 2, -2)
ans = -1.4142
>>
```