Matlab (II)

张晓平

武汉大学

2018 年 9 月 10 日

目录 I

1. 脚本文件

脚本文件

matlab 程序保存在 m 文件中，以".m" 结尾。m 文件分为两种：

- 脚本文件
- 函数文件

脚本文件

对于脚本文件：

- 首行不以function开头；
- 可直接复制到命令行运行，也可保存成 m 文件执行；
- 必须都是脚本，不能包含函数定义，或者说不能出现function关键词。

脚本文件

对于函数文件：

- 首行以function开头；
- 必须保存成 m 文件执行，然后在脚本里或者其他函数里调用；
- 函数文件可以包含子函数，如果一个 m 文件有多个子函数，那么只有第一个函数为主函数，后面的都是子函数。

脚本文件

总而言之，脚本文件有以下特点：

- 多条命令的综合体
- 没有输入、输出变量
- 使用 matlab 基本 workspace
- 没有函数声明行

脚本文件的编写

问题　编写程序，绘制 $y=\tan\theta$ 在 $(1.6,4.6)$ 间的图像。设该程序放在目录 `D:\NA\OP2` 中。

脚本文件的编写

问题　编写程序，绘制 $y=\tan\theta$ 在 $(1.6,4.6)$ 间的图像。设该程序放在目录 `D:\NA\OP2` 中。

1、进入目录

```
>> cd D:\NA\OP2
```

脚本文件的编写

问题　编写程序，绘制 $y=\tan\theta$ 在 $(1.6,4.6)$ 间的图像。设该程序放在目录 `D:\NA\OP2` 中。

1、进入目录

```
>> cd D:\NA\OP2
```

2、在命令行中输入

```
>> edit tanplot.m
```

脚本文件的编写

3、在编辑器中输入以下代码:

```
theta = linspace(1.6, 4.6);
tandata = tan(theta);
plot(theta, tandata);
xlabel('\theta (radians)');
ylabel('tan(\theta)');
grid on;
axis([min(theta) max(theta) -5 5])
```

脚本文件的编写

4、按 Ctrl+S 保存;

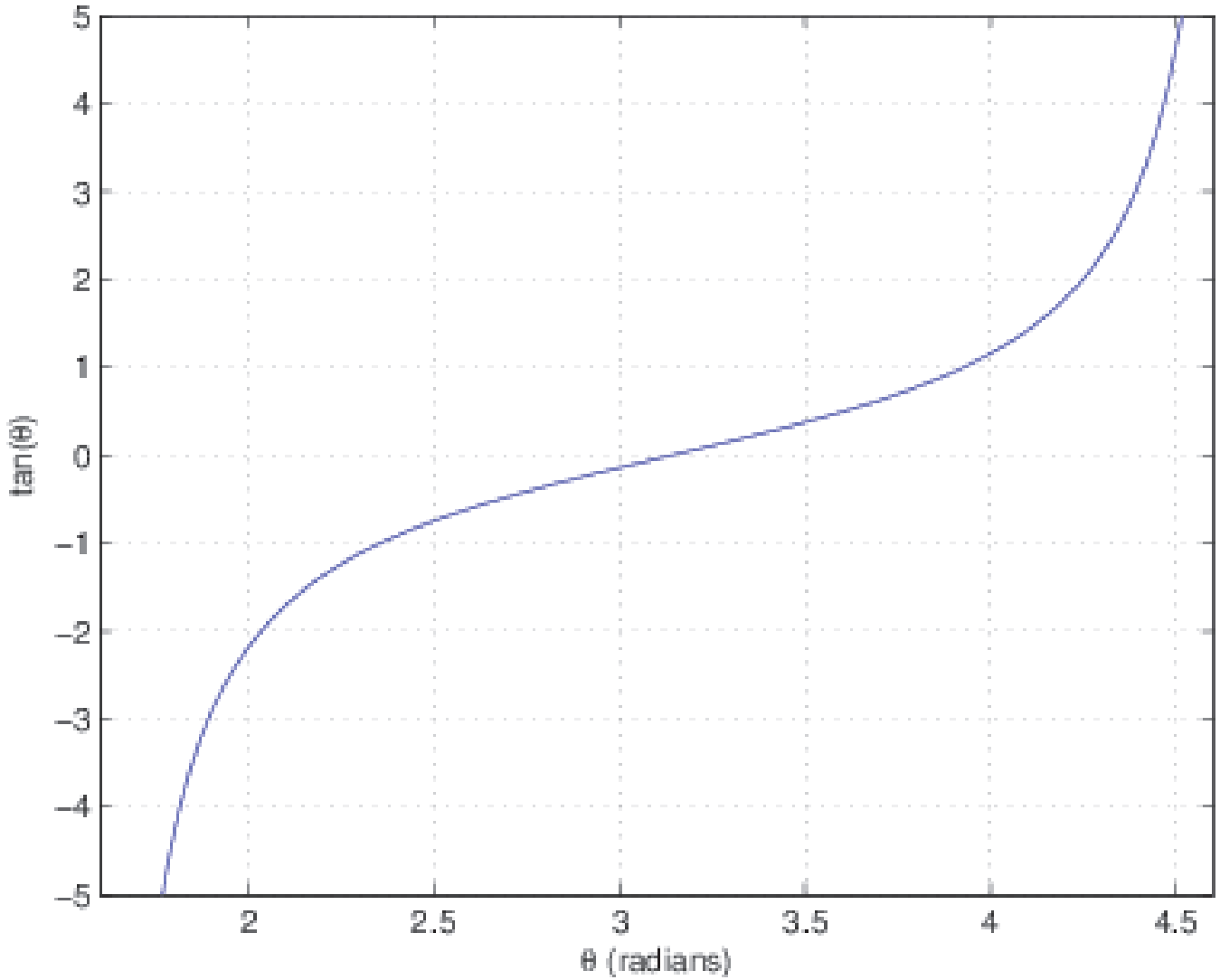
脚本文件的编写

4、按 Ctrl+S 保存；

5、在命令行中输入

```
>> tanplot
```

脚本文件的编写



脚本文件的编写

若需修改绘图，只要编辑脚本文件然后重新运行即可，这就可以避免重复键入命令。

脚本的副作用

脚本文件中的所有变量会被添加到 workspace，这可能导致一些不可预料的结果。其原因在于

- workspace 已经存在的变量可能会被重写；
- workspace 中的状态变量可能会影响脚本的运行。

脚本文件

easyplot.m

```
D = load('xy.dat');          % D 是一个含两列的矩阵
x = D(:, 1);                 % x 为第一列
y = D(:, 2);                 % y 为第二列
plot(x, y);
xlabel('x axis')
ylabel('y axis')
title('Plot of generic x-y data set')
```

脚本文件

easyplot.m 影响了 workspace，它创建了三个变量：

```
>> clear
>> who

>> easyplot
>> who
Your variables are:
D     x     y
```

脚本文件

通常，脚本的副作用

- 可能导致难以追踪的 bug
- 难以避免
- 创建和改变 workspace 中的变量
- workspace 中的变量发生改变时，不会给出任何警告

由于脚本有副作用，更好的办法是将复杂的计算封装在函数式 m 文件中

2. 函数文件

函数文件

- 函数是子程序
  - 函数使用输入输出变量来与其他函数或命令窗口进行通信
  - 函数使用的是临时变量，它们只在函数运行时存在。临时变量有别于工作空间或其他函数中的同名变量。

函数文件

- 输入变量的使用使得函数可以使用不同的数据进行相同的操作，亦即函数式 m 文件是可重用的。
- 函数可以调用其它函数。
- 指定的任务可封装在函数中，这种模块化的操作使得解决复杂问题的结构化成为可能。

函数文件

```
function [outArgs] = funName(inArgs)
```

- `outArgs` 放在 `[]` 中
  - 若 `outArgs` 有多个变量，用逗号隔开
  - 若只有一个输出变量，则 `[]` 是可选的
  - 允许没有 `outArgs`
- `inArgs` 放在 `()` 中
  - 若 `inArgs` 有多个变量，用逗号隔开
  - 不允许没有 `inArgs`

函数文件

```
function twosum(x, y)  %two inputs and no output
x + y

function s = threesum(x, y, z) %three inputs and
 one output
s = x + y + z;

function [s, p] = addmult(x, y) %two input and
two output
s = x + y;
p = x * y;
```

函数文件

```
>> twosum(2, 2)
ans =
     4

>> x = [1 2]; y = [3 4];
>> twosum(x, y)
ans =
     4     6
```

函数文件

```
>> A = [1 2; 3 4]; B = [5, 6; 7 8];
>> twosum(A, B);
ans =
     6     8
    10    12

>> twosum('one', 'two')
ans =
    227    229    212
```

函数文件

```
>> clear
>> x = 4; y = -2;
>> twosum(1, 2)
ans =
     3
>> x + y
ans =
     2
```

函数文件

```
>> disp([x y])
    4    -2
>> who
Your variables are:
ans      x        y
```

函数文件

```
>> a = threesum(1, 2, 3)
a =
     6
>> threesum(4, 5, 6)
ans =
     15
>> b = threesum(7, 8, 9);
```

函数文件

```
>> [a, b] = addmult(3, 2)
a =
     5
b =
     6
>> addmult(3, 2)
ans =
     5
>> v = addmult(3, 2)
v =
     5
```

函数文件

`addmult` 需要两个返回变量。调用 `addmult` 时若无返回变量或只有一个返回变量会导致不恰当的行为。

函数文件

- 函数通过输入输出变量赋值;
- 函数中定义的变量对该函数来说是局部的，局部变量对其他函数或命令环境不可见;
- 返回变量的个数应该与函数提供的输出变量的个数相匹配。

函数文件

通常，需要把结果输出在屏幕或者文件中。还有些时候，需要给用户提供一些提示说明。

- 函数的输入
  - 使用 `input` 函数
  - 更好的方式是使用函数的输入参数
- 从函数中输出文字
  - 使用 `disp` 函数
  - 使用 `fprintf` 函数进行格式化输出

input 函数

可用于提示用户输入数值或文字

```
>> x = input('Enter a value for x');
>> yourName = input('Enter your name', 's');
```

应该避免使用 input 函数，它不是必要的。函数的所有输入应该通过输入变量提供。

disp 和 fprintf

命令窗口中的输出可以通过 `disp` 或者 `fprintf` 函数实现。而输出到文件则需要使用 `fprintf` 函数。

- `disp` 容易使用，但对输出格式的控制有限；
- `fprintf` 比 `disp` 复杂，但方便对输出格式的控制。

disp

```
disp(outMatrix)
```

disp 输出数值

```
>> disp(5)
     5
>> x = 1:3; disp(x)
     1     2     3
>> y = 3-x; disp([x; y])
     1     2     3
     2     1     0
>> disp([x y])
     1     2     3     2     1     0
>> disp([x' y])
??? Error using ==> horzcat
CAT arguments dimensions are not consistent.
```

注意：最后一条语句表明 disp 函数必须输入合法的矩阵

disp 输出字符串

```
>> disp('Hello world!')
Hello, world!

>> s='MATLAB is built with LAPACK';
>> t='Earlier version used LINPACK and EISPACK';
>> disp([s; t])
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.

>> disp(char(s, t))
MATLAB is built with LAPACK
Earlier version used LINPACK and EISPACK
```

disp 输出字符串

`disp[s; t]` 会引起错误，因为 `s` 的元素个数小于 `t`。

char 函数

char 函数构造一个字符串矩阵，它将每个输出放在分隔的行上，如有必要将行以空字符填充。

```
>> S = char(s, t);
>> length(s), length(t), length(S(1, :))
ans =
    29
ans =
    41
ans =
    41
```

disp 与 num2str 搭配使用

```
stringValue = num2str(numericValue)
```

num2str 通常与 disp 一起用于创建数值的字符输出。

disp 与 num2str 搭配使用

```
>> num2str(pi)
ans =
    3.1416
>> A = eye(3)
A =
     1     0     0
     0     1     0
     0     0     1
>> S = num2str(A)
S =
1  0  0
0  1  0
0  0  1
```

disp 与 num2str 搭配使用

```
>> clear
>> A = eye(3); S = num2str(A); B = str2num(S);
>> A - S
>> A - S
??? Error using ==> minus
Matrix dimensions must agree.
>> A - B
ans =
0     0     0
0     0     0
0     0     0
```

disp 与 num2str 搭配使用

```
>> whos
Name        Size                    Bytes  Class
Attributes

A           3x3                        72  double
B           3x3                        72  double
S           3x7                        42  char
```

disp 与 num2str 搭配使用

num2str 与 disp 一起可用于数值的文本输出。

```
>> x = sqrt(2);
>> outString = ['x␣=␣', num2str(x)];
>> disp(outString)
x = 1.4142
```

或

```
>> disp(['x␣=␣', num2str(x)]);
x = 1.4142
```

disp 与 num2str 搭配使用

```
disp(['x = ', num2str(x)]); % x is a row vector
```

```
>> y = [1 2 3 4]; z = y';
>> disp(['z = ', num2str(z)])
??? Error using ==> horzcat CAT arguments
dimensions are not consistent.
```

```
>> disp('z = '); disp(z)
z =
     1
     2
     3
     4
```

format 函数

format 函数用于控制 disp 输出的精度。

```
>> format short
>> disp(pi)
    3.1416
>> format long
>> disp(pi)
    3.141592653589793
```

format 函数

num2str 函数的第二个参数也可以用于控制输出的精度。

```
>> disp(['pi = ', num2str(pi, 2)])
pi = 3.1
>> disp(['pi = ', num2str(pi, 4)])
pi = 3.142
>> disp(['pi = ', num2str(pi, 8)])
pi = 3.1415927
```

fprintf 函数

```
fprintf(outFormat, outVariables)
fprintf(fileHandle, outFormat, outVariables)
```

1. `outFormat` 为格式化字符串，提供各种占位符；
2. 第一种形式将输出显示在命令窗口；
3. 第二种形式将输出写入由 `fileHandle` 指定的文件中。

fprintf 函数

C 程序员需注意:

- matlab 中的 `fprintf` 函数使用单引号' 来定义格式化字符串;
- `fprintf` 函数是向量化的。

```
>> x = 3;
>> fprintf('Square root of %g is %8.6f\n', x,
sqrt(x));
Square root of 3 is 1.732051
```

fprintf 函数

表: 格式化字符串

字符	说明
%s	字符串
%d	整型
%f	浮点型
%e	浮点型 (科学计数法)
%g	%f 或%e 的紧凑形式
\n	换行符号
\t	插入 tab 键

fprintf 函数

除了可以指定转换类型 (如%d，%f，%e) 外，还可指定转换宽度与精度。

```
%wd
%w.pf
%w.pe
```

w 表示打印结果的字符个数，p 表示小数点后的位数。

fprintf 函数

格式化字符串	含义
%14.5f	浮点型，共 14 个字符，其中小数点后 5 位
%12.3e	科学计数法，共 12 个字符，小数点后 3 位，包含字符串 e+00 或 e-00

fprintf 函数

Value	%8.4f	%12.3e	%10g
2	**2.0000	***2.000e+00	*********2
sqrt(2)	**1.4142	***1.414e+00	***1.41421
sqrt(2e-11)	0.0000	***4.472e-06	4.47214e-06
sqrt(2e11)	4.4721e+05	***4.472e+05	****447214

fprintf 函数

fprintf 是向量化的，这使得可用紧凑的形式打印向量或者矩阵。

```
>> x = 1:4; y = sqrt(x);
>> fprintf('%9.4f\n', y)
1.0000
1.4142
1.7321
2.0000
```

fprintf 函数

```
>> x = 1:4; y = sqrt(x);
>> fprintf('y␣=␣%9.4f\n', y)
y =      1.0000
y =      1.4142
y =      1.7321
y =      2.0000
```

%9.4f 重复用于 y 的每个元素。

fprintf 函数

向量化的 fprintf 逐列打印，但这也可能导致一些意外的结果：

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
```

fprintf 函数

```
>> fprintf('%8.2f %8.2f %8.2f\n', A)
    1.00    4.00    7.00
    2.00    5.00    8.00
    3.00    6.00    9.00

>> fprintf('%8.2f %8.2f %8.2f\n', A')
    1.00    2.00    3.00
    4.00    5.00    6.00
    7.00    8.00    9.00
```

fprintf 打印表格 I

```
function boxSizeTable
label = char('small', 'medium', 'large', 'jumbo'
);
width = [5; 5; 10; 15];
height = [5; 8; 15; 25];
depth = [15; 15; 20; 35];
vol = width .* height .* depth / 10000;
fprintf('\n␣Sizes␣of␣boxes␣used␣by␣ACME␣Delivery
␣Service\n\n');
fprintf('size␣␣␣␣␣width␣␣height␣␣depth␣␣␣volume\
n');
fprintf('␣␣␣␣␣␣␣␣␣␣␣(cm)␣␣␣␣(cm)␣␣␣(cm)␣␣␣␣(m^3)\
n');
for i = 1:length(width)
    fprintf('%-8s␣%5d␣␣␣%5d␣␣%5d␣%8.5f\n', ...
```

fprintf 打印表格 II

```
        label(i,:), width(i), height(i), ...
        depth(i), vol(i))
end
```

fprintf 打印表格

```
>> boxSizeTable
 Sizes of boxes used by ACME Delivery Service

size        width   height   depth    volume
             (cm)     (cm)    (cm)      (m^3)
small           5        5      15   0.03750
medium          5        8      15   0.06000
large          10       15      20   0.30000
jumbo          15       25      35   1.31250
```

利用 fprintf 写文件

利用 fprintf 函数输出到文件需要通过 fopen 函数创建文件句柄。之前讨论的格式化与向量化仍然适用。

利用 fprintf 写文件

举例：将向量写入文件

```
x = ...
fout = fopen('myfile.dat', 'wt');
for k = 1:length(x)
    fprintf(fout, '%4d␣␣␣%5.2f\n', k, x(k));
end
fclose(fout)
```

3. 程序流程控制

程序流程控制

三种基本流程结构

- 顺序结构
- 选择结构（分支）
  - `if`
  - `if ... else`
  - `if ... elseif`
  - `switch`
- 循环结构
  - `for`
  - `while`

3.1 关系运算符与逻辑运算符

关系运算符

表: 关系运算符

运算符	含义
<	小于
<=	小于等于
>	大于
>=	大于等于
~=	不等于

关系运算符

- 关系运算符用于比较两个值，常用于选择结构与循环结构
- 关系运算式的结果为逻辑值，即 `true` 或 `false`。
- 在 matlab 中，任何非零值，包括非空字符串，都等价于 `true`。只有 0 等价于 `false`。

关系运算符

关系运算符可用于标量的比较:

```
>> a = 2; b = 4;
>> a < b
ans =
     1

>> b < a
ans =
     0
```

关系运算符

也可用于相同尺寸矩阵的比较:

```
>> x = 1:5; y = 5:-1:1
>> x > y
ans =
     0     0     0     1     1
```

逻辑运算

表: 逻辑运算符

运算符	含义
&	逻辑与
|	逻辑或
~	逻辑非

逻辑运算符用于连接逻辑表达式，或改变逻辑值。

逻辑运算

```
>> a = 2; b = 4;
>> aIsSmaller = a < b; bIsSmaller = b < a;
>> bothTrue = aIsSmaller & bIsSamller
bothTrue =
     0
>> eitherTrue = aIsSmaller | bIsSmaller
eitherTrue =
     1
>> ~eitherTrue
ans =
     0
```

关系运算符与逻辑运算符

- 关系运算是关于两个值的比较，其结果是逻辑值 (`true` 或 `false`);
- 逻辑运算生成新的逻辑值;
- 相同的比较通常有很多种表达方式。

3.2 分支结构

分支结构

- `if`
  - `if`
  - `if ... else`
  - `if ... elseif`
- `switch`

if 结构

语法

```
if expression
    block of statements
end
```

举例

```
if a < 0
    disp('a is negative');
end
```

if 结构

在 if 表达式后加上逗号，可简写成一行：

```
if a < 0, disp('a is negative'); end
```

if...else... 结构

若遇到多重分支，可用 if...else... 结构

```
if x < 0
    error('x is negative; sqrt(x) is imaginary')
    ;
else
    r = sqrt(x);
end
```

if...elseif...else... 结构

```
if x > 0
    disp('x is positive');
elseif x < 0
    disp('x is negative');
else
    disp('x is exactly zero');
end
```

switch 结构

语法

```
switch expression
case value1,
    block of statements
case value2,
    block of statements
...
otherwise,
    block of statements
end
```

switch 结构

举例

```
color = '...';          %  color is a string
switch expression
case 'red',
    disp('Color is red');
case 'blue',
    disp('Color is blue');
case 'green',
    disp('Color is green');
otherwise,
    disp('Color is not red, blue or green');
end
```

3.3 循环结构

循环结构

循环结构有两种:

- `for` 循环
- `while` 循环

for 循环

语法

```
for index = expression
    block of statements
end
```

for 循环通常用于遍历矩阵:

```
x = 1:5;
sumx = 0;
for k = 1:length(x)
    sumx = sumx + x(k);
end
```

for 循环

指标以 2 递增

```
for k = 1:2:n
    ...
end
```

for 循环

指标递减

```
for k = n:-1:1
    ...
end
```

for 循环

步长为非整数

```
for k = 0:pi/15:pi
    fprintf('%8.2f␣%8.5f\n', x, sin(x));
end
```

while 循环

语法

```
while expression
    block of statements
end
```

while 循环

用牛顿法求 $\sqrt{x}$ 的值：$r_k = \frac{1}{2}(r_{k-1} + \frac{x}{r_{k-1}})$

```
r = ...
rold = ...
while abs(rold-r) > delta
    rold = r;
    r = 0.5 * (rold + x / rold);
end
```

while 循环

一个好的习惯是在 while 循环中设置迭代次数的上限。

```
maxit = 25;
it = 0;
while abs(rold -r) > dalta & it < maxit
    rold = r;
    r = 0.5*(rold + x / rold);
    it = it + 1;
end
```

break 和 return 语句

break 和 return 语句用于跳出循环结构。

- break 用于跳出 while 或 for 循环，但在循环结束的位置继续执行；
- return 用于强制退出函数，任何函数中的循环语句都会被跳过。

break 语句

```
function k = breakdemo(n)
x = rand(1, n);
k = 1;
while k <= n
    if x(k) > 0.8
        break;
    end
    k = k + 1;
end
fprintf('x(k) = %f for k = %d (n = %d)', x(k), k
, n);
```

return 语句

```
function k = returndemo(n)
x = rand(1, n);
k = 1;
while k <= n
    if x(k) > 0.8
        return;
    end
    k = k + 1;
end
```

break 与 return 的差别

break 用于跳出当前 while 或 for 循环，return 用于跳出当前函数。

4. 向量化操作

向量化操作

matlab 向量化编程，是 matlab 语言的精髓所在。向量化编程运用得好，可以从代码的运行效率明显改善中获得成功的快乐。

- 尽量避免循环的使用，多使用 matlab 的内置函数
- 使用变量前养成预分配内存的习惯
- 向量化计算代替逐点计算
- 如果矩阵含有大量 0 元素，尽量采用稀疏矩阵来提高运算速度和减少存储空间

向量化代替循环

标量代码

```
for k = 1:length(x)
    y(k) = sin(x(k))
end
```

向量化代码

```
y = sin(x)
```

预分配内存

```
y = ...
for j = 1:length(y)
    if y(j) > 0
        s(j) = sqrt(y(j));
    else
        s(j) = 0;
    end
end
```

预分配内存

在给 s 赋值时预分配内存

```
y = ...
s = zero(size(y));
for j = 1:length(y)
    if y(j) > 0
        s(j) = sqrt(y(j));
    end
end
```

向量化索引与逻辑函数

代码的向量化需要用到

1. 数组索引
   将向量或矩阵作为另一个矩阵的“下标”
2. 逻辑索引

数组索引

将向量或矩阵作为另一个矩阵的“下标”:

```
>> x = sqrt(0:4:16)
x =
    0    2.0000    2.8284    3.4641    4.0000
>> i = [1 2 5];
>> y = x(i)
y =
    0    2    4
```

数组索引

等价于

```
k = 0;
for i = [1 2 5]
    k = k + 1;
    y(k) = x(i);
end
```

逻辑索引

```
>> x = sqrt(0:4:16)
x =
     0     2.0000     2.8284     3.4641     4.0000
>> j = find(rem(x, 2)==0)
j =
     1     2     5
>> z = x(j)
z =
     0     2     4
```

逻辑索引

请将以下代码向量化:

```
y = ...
s = zero(size(y));
for j = 1:length(y)
    if y(j) > 0
        s(j) = sqrt(y(j));
    end
end
```

逻辑索引

方法 1:

```
y = ...
s = zeros(size(y));
i = find(y>0);
s(i) = sqrt(y(i))
```

逻辑索引

方法 2:

```
y = ...
s = zeros(size(y));
s(y>0) = sqrt(y(y>0))
```

向量化操作

复制整行或整列 (标量代码)

```
[m, n] = size(A)
for i = 1:m
    B(i, 1) = A(i, 1)
end
```

复制整行或整列 (向量化代码)

```
B(:, 1) = A(:, 1)
```

向量化操作

赋值及转换子矩阵 (标量代码)

```
for j = 2:3
    B(1, j) = A(j, 3);
end
```

赋值及转换子矩阵 (向量化代码)

```
B(1, 2:3) = A(2:3, 3)
```