

# MATLAB

## 学习笔记 v0.1.3

*MATLAB* 学习之路：功能与效率

陈磊

August 17, 2016

email [chenlei.here@gmail.com](mailto:chenlei.here@gmail.com)

blog <http://herechen.github.io>

source <https://github.com/HereChen/TheWayMATLABLearning>

这份文档于 2013 年的夏天开始撰写. 不同于开始, 后来我是这样打算的, 以简洁的方式呈现自己所学的 MATLAB 知识. 一则温故知新, 二则留存一份历史, 如是而已.

文中写作上既不全面描述, 也不过于细致描述, 简单描述的目的在于“知”, 知道一些功能, 知道一些技巧, 然后可以探究.

## Contents

<b>1 学途伊始</b>	<b>1</b>
1.1 窗口	1
1.2 脚本	2
1.3 函数	2
<b>2 一些功能</b>	<b>2</b>
2.1 快捷键	2
2.2 MATLAB 启动	3
2.3 路径设置	3
2.4 startup	4
2.5 计时器	5
2.6 代码单元	5
2.7 断点调试	6
2.8 代码发布	6
2.9 代码保护	7
2.10 应用发布	8
2.11 警告和错误	8
2.12 显示格式	8
<b>3 一些函数</b>	<b>9</b>
<b>4 一些技巧</b>	<b>10</b>
4.1 内存溢出及解决方案	10
4.2 自定义函数帮助	11
4.3 图像导出	11

4.4 公式打印	11
4.5 重复向量以构造矩阵	12
<b>5 高效编程</b>	<b>13</b>
5.1 分析代码性能	13
5.2 预分配内存	13
5.3 临时变量	14
5.4 MATLAB 建议	14
5.5 矩阵存储方式	14
5.6 函数类型	15
5.7 大数据处理	15
5.8 并行计算	15
<b>6 额外兴趣</b>	<b>16</b>
6.1 拍照	16
<b>A Appendix</b>	<b>16</b>
A.1 MATLAB 资源	16

## List of Tables

1 快捷键	3
-------	---

## List of Figures

1 添加路径	4
2 断点调试	6

3	代码发布 . . . . .	7	10	警告输出 . . . . .	8
4	内存溢出和内存查看 . . . . .	10	11	错误输出 . . . . .	8
			12	Command 窗口显示格式 . . . . .	8
			13	生成随机矩阵 . . . . .	10
			14	自定义帮助 . . . . .	11
			15	图像中的公式输出 . . . . .	11
			16	重复向量以构造矩阵 . . . . .	12
			17	用 Profiler 分析代码性能 . . . . .	13
			18	是否预分配内存效率对比 . . . . .	13
			19	矩阵不同存取方式效率对比 . . . . .	14
			20	不同函数类型效率对比 . . . . .	15
			21	并行计算 parfor 与 for 效率对比 . . . . .	15
			22	拍照 . . . . .	16

## List of Examples

1	第一个脚本 . . . . .	2
2	第一个函数 . . . . .	2
3	我的 startup.m . . . . .	4
4	计时 1 . . . . .	5
5	计时 2 . . . . .	5
6	代码单元 . . . . .	5
7	代码发布 . . . . .	7
8	代码保护 . . . . .	7
9	红色字符串输出 . . . . .	8

## 1 学途伊始

最开始学习 MATLAB 是从脚本开始的, 写一段然后就选中代码按 F9 这种. 后来逐渐需要把一些通用或常用的脚本写成函数, 需要的时候一条语句就可以实现. 这大概也就是整体上我的学途历程了.

再往后, 就开始考虑效率, 接触一些方便好用的功能, 考虑算法设计和 MATLAB 编程特性的结合, 如何去调试程序等等.

### 1.1 窗口

MATLAB 有几个基本的窗口: Command Window 用于输出结果以及执行命令之用; Editor 则作为编辑代码之用; 在 Workspace 可以看到当前保存的变量; Command History 保存了执行过的命令的历史记录; Current Folder 是当前目录, 方便了文件和资源管理.

## 1.2 脚本

在 MATLAB 里面 Ctrl+N 新建一个后缀为 m 的文件, 也就是通常说的 M-file. 写入要实现的功能, 选中要执行的代码然后按 F9 就可以在 Command 窗口查看输出结果, 或者在 Workspace 中查看保存的变量. 比如, 下面给出的例子就包含了注释、一个输出语句以及一个矩阵的创建.

Example 1: 第一个脚本

---

```
disp('Hello MATLAB') % 我是注释,前面一条语句是输出
A = [1 2; 3 4]; % 创建一个矩阵,若句尾不加分号执行时会直接输出
```

---

## 1.3 函数

把常用到的功能或者具有一定功能的单独拿出来, 写成一个函数, 在需要的时候直接调用. 这一方便避免的代码的重复书写, 又方便了程序的调试. 函数不能在当前文件执行, 需要通过调用的形式来执行. 比如下面的这个函数保存为文件之后, 可以直接在 Command 窗口调用.

Example 2: 第一个函数

---

```
function sm = twonosum(no1,no2)
% 两个数之和
sm = no1 + no2;
end
```

---

- 保存时需要和函数名一致, 上面的例子应保存为 twonosum.m;
- Command 窗口或脚本调用格式为 var1 = twonosum(var2, var3);
- 文件 twonosum.m 所在路径需要添加, 这在文中的路径设置中提到;

Note 1.1. 最后的 end 并非是必须的, 可以删除.

## 2 一些功能

### 2.1 快捷键

Table 1: 快捷键

功能	快捷键	功能	快捷键
注释	Ctrl + R	切换到 Command Window	Ctrl + 0
自动排版	Ctrl + I	切换到 Command History	Ctrl + 1
取消注释	Ctrl + T	切换到 Current Folder	Ctrl + 2
向右缩进	Ctrl + [	切换到 Workspace	Ctrl + 3
向左缩进	Ctrl + ]	切换到 Editor	Ctrl + Shift + 0
执行当前单元	Shift + Ctrl + Enter	Editor 之间的切换	Ctrl + PgDn/PgUp
终止程序	Ctrl + C	添加函数	Ctrl + J
上一个单元	Ctrl + Down	下一个单元	Ctrl + Up

## 2.2 MATLAB 启动

这里不描述通常的点击图标启动. 有这样两种启动方式:

- matlab, 完全启动, 即正常的启动;
- matlab -nojvm, 只启动 Command 窗口, 这将禁用与 java 相关的功能, 启动速度较快.

在 Windows 中, 可以命令方式来启动, 即在 cmd 或 Ctrl+R 调出的“运行”中执行. 命令启动有个前提, MATLAB 的可执行程序路径已经在环境变量中.

在图中直接输入以 matlab 可实现完全启动. 若是期望轻量级编程, 可以“no java”方式启动, 并配合其他编辑器 (比如 Sublime Text), 可以实现脚本或函数的编辑及执行. 对两种方式做个对比:

- 完全启动, 速度慢, 功能齐全, 占用内存大 (相对);
- “no java”启动, 速度快, 功能少一部分, 占用内存小.

*Note 2.1.* MATLAB 环境变量的设置. 假设 MATLAB 安装在 C:\MATLAB 目录下, 那么, matlab.exe 应该在 C:\MATLAB\bin 目录下. 接着设置环境变量, 电脑属性 → 高级系统设置 → 环境变量 → 用户变量下选中 PATH 并点击编辑按钮 → 添加 C:\MATLAB\bin(在最后加入分号添加) → 一路保存退出.

## 2.3 路径设置

设置路径的方法:

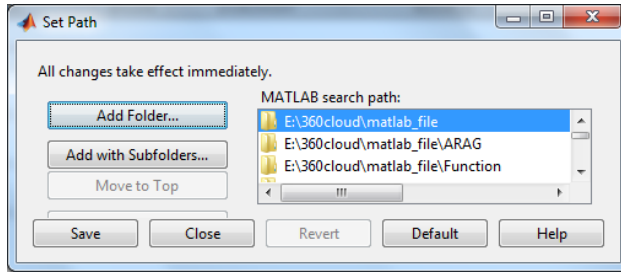


Figure 1: 添加路径

- 鼠标点击实现. Home→set path. Add Folder... 只添加选定文件夹的路径, Add with Subfolders... 添加指定文件夹及其所有子文件夹路径.
- 通过命令添加, 比如:

---

```
addpath('D:\matlab_file')
```

---

- 直接通过路径文件设置.

---

```
>> which pathdef
C:\Program\MATLAB\R2013a\toolbox\local\pathdef.m
>> edit('C:\Program\MATLAB\R2013a\toolbox\local\pathdef.m')
```

---

- 设置工作目录. MATLAB 默认的工作目录可以通过输入 userpath 来查看, 更改则通过 userpath(mypath) 来实现, 其中 mypath 是自定义的路径字符串. 设置工作目录的意义在于, 将需要的文件放在放在工作目录, 就无需额外添加路径了.

添加路径的意义在于, 这使得我们期待调用的脚本、函数或资源等能够被找到.

*Note 2.2.* 如果路径中包含空格, 用 ' ' 来表示空格, 即两个单引号, 中间包含一个空格.

## 2.4 startup

startup.m 是在 MATLAB 启动时就执行的脚本. 若是需要它, 我们需要做三件事情:

- 新建 startup.m;
- 确保 startup.m 所在文件夹的路径已经添加;
- 编辑需要 MATLAB 启动就执行的内容.

### Example 3: 我的 startup.m

---

```
clc; % 清除Command窗口显示的内容
```

---

```
cd D:\matlabfile;      % 定位到matlabfile文件夹
addpath(genpath(pwd)); % 将当前文件夹的所有文件夹及子文件夹添加到路径中
% pwd为获得当前文件夹路径,即D:\matlabfile
```

---

这段代码执行 3 个命令,主要目的还是在于进入到我的工作目录,并添加可能的新的路径(因而用了 `genpath`).

## 2.5 计时器

`tic` 为计时开始, `toc` 为计时结束. 通常用以查看代码/算法执行的时间.

---

### Example 4: 计时 1

---

```
tic
disp('Time recorder...');
toc
```

---

---

### Example 5: 计时 2

---

```
t1 = tic;
disp('Time recorder...');
toc(t1)
```

---

```
Time recorder...
Elapsed time is 0.007663 seconds.
```

---

两个例子单独的执行效果如上. 在使用时可以注意以下几点:

- 若是不需要输出时间, 在 `toc` 后加分号即可;
- 若需要保存记录的时间, 可用变量存储, 比如 `usetime = toc` 或 `usetime = toc(t1)`.
- 计时方案 2 适合有多个计时需求的应用.

*Note 2.3.* 另一种计时 `etime`, 但这并不是推荐的方式, [help etime](#) 查看.

## 2.6 代码单元

`%` 是注释, `%%` 则是一个单元 (同时作为注释).

---

### Example 6: 代码单元

---

```
%% display cell 1
disp('the first cell.');
```

```
%% display cell 2
disp('the second cell.');
```

% 两个百分号与注释之间有空格.

---

代码单元可在不选中代码的情况下, 可以快捷键 `Ctrl+Shift+Enter` 连续执行. 并还有以下好处:

- 一个单元一个单元的执行, 方便检错/调试;
- 方便查看过程数据;
- 把具有特定功能的一段代码作为一个单元, 使得代码结构清晰.

## 2.7 断点调试

设置断点是调试程序常用的方法, 可以用这种方法来找出错误所在, 或者了解代码执行过程. 在需要暂停的一行或多行前用鼠标点击设置断点.

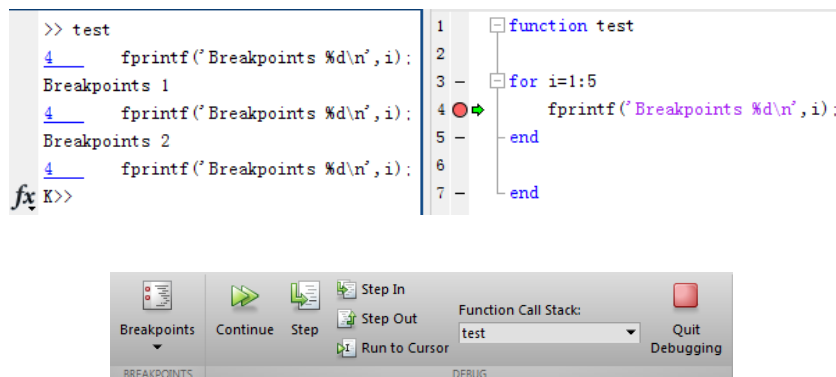


Figure 2: 断点调试

相对于通过执行选中代码段来调试, 这种方式更加便捷, 并且适合批量操作. 断点调试常用的两个快捷键:

- `F5`, 继续执行;
- `Shift+F5`, 停止继续执行 (Quit Debugging).

## 2.8 代码发布

代码发布, 就功能而言, 即可用其他文档格式展现代码. 发布的格式有多种可选, 比如: latex、html、doc、ppt 等等. 发布方式:

- 在菜单栏 **Publish** 下设置和发布;
- 通过函数 `publish` 发布, 查看 [help publish](#). 比如将 `test.m` 发布为 pdf.



## Example 7: 代码发布

```
publish('test.m','pdf');
```

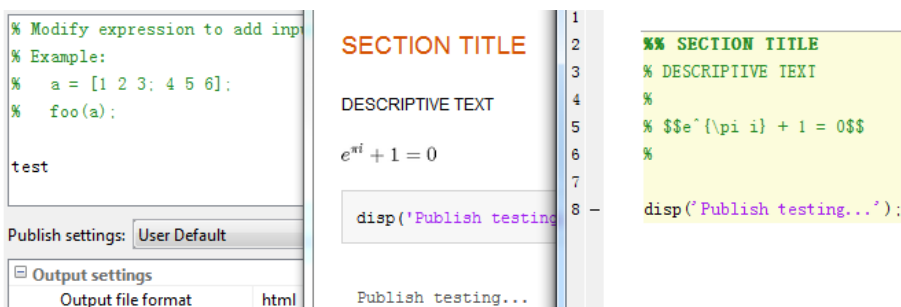


Figure 3: 代码发布

发布代码的同时, 结果也会一同发布, 这包括输出和图像等. 代码发布还有这样一些细节:

- 添加章节, 使代码更加直观. 这可以通过代码单元来实现;
- 注释设置, 一则可以设置字体, 二则可以设置列表 (list);
- 添加其他元素, 比如: 图片、公式、超链接.

*Note 2.4.* 就我当前所用版本, 要发布含数学公式的建议用  $\text{\LaTeX}$ , 其他几种公式转换为图片发布, 效果并不好. 或可用其他方式弥补也是可以的 (比如先发布为  $\text{\LaTeX}$ , 再生成 pdf). 也许, 这仅仅是因为对  $\text{\LaTeX}$  的钟爱.

## 2.9 代码保护

代码保护 (protect function) 从功能上来理解, 有两点:

- 代码加密 (用这种方式生成的文件是被加密的), 同时又能像其他文件一样调用;
- 预解析文件, 直接调用解析过的文件, 可以提高执行速度.

首先, 通过 `pcode` 将 `m` 文件解析为 `p` 后缀文件; 然后直接调用该文件. 这里给出了针对单个文件的例子, 但也可一次解析多个文件.

## Example 8: 代码保护

```
pcode filename.m
```

*Note 2.5.* 生成解析文件后, 路径下就有两个同名, 但是后缀不同的文件. 对 `m` 文件修改后需要重新解析, 不然, 会调用较新的 `m` 文件, 而不是 `p` 文件. 另外, “解析” 一词在这里更多的是指 `pcode` 的函数作用.

## 2.10 应用发布

应用发布是指把脚本或者函数编译成可执行的 `exe` 文件. 应用发布包含两个步骤:

- `mbuild -setup` 选择和配置编译器;
- `mcc -m myfile.m` 将脚本编译成可执行文件.

注意上面命令之间的空格. 编译得到的同名 `exe` 文件在 MATLAB 安装目录的 `bin` 文件夹下, 或者在所编译文件所在的文件夹下.

## 2.11 警告和错误

在编写程序时, 会用一些输出来标识警告或者错误, 以下是针对这个问题的方法.

Example 9: 红色字符串输出

---

```
fprintf(2,'edit your text here.');
```

---

Example 10: 警告输出

---

```
warning('edit your text here.');
```

---

Example 11: 错误输出

---

```
error('edit your text here.');
```

---

*Note 2.6.* `warning` 和 `error` 可以像 `fprintf` 一样输出字符串或者数值. 例如:

```
str = 'I am a string'; warning('%s',str)
```

## 2.12 显示格式

Command 窗口的默认显示格式是 `short`, 此时显示小数时最多显示 4 位小数. 常用的切换有显示更多的小数, 以及将小数作为分数显示. 这里介绍三种格式切换, 更多的查看 [help format](#).

Example 12: Command 窗口显示格式

---

<code>format short</code>	% 默认格式
<code>format long</code>	% 显示更多小数位
<code>format rat</code>	% 以分数形式显示

---

### 3 一些函数

对于要介绍的函数或命令, 在每个说明后面紧接着给出示例, 或者是使用是说明.

- `eval` – 将字符串作为 MATLAB 语句执行. `eval('1 2'*3')`
- `help` – 查看帮助文档. 查看 `size` 函数帮助, `help size`
- `prod` – 向量元素的连乘. 矩阵同行元素相乘, `prod([1 2;3 4],2)`
- `pause` – 暂停. 暂停 5 秒, `pause(5)`
- `nnz` – 矩阵非零元素的个数. `nnz([0 0 0 1 2])`
- `nonzeros` – 矩阵非零元素. `nonzeros([0 0 0 1 2])`
- `xlsread` – Excel 文件读取. 调用格式 `[NUM,TXT,RAW]=xlsread(FILE,SHEET,RANGE)`
- `xlswrite` – Excel 文件写入. 如果不加 `SHEET` 参数则存储到默认的表格, 若是自定义则会在 Excel 中添加新的表格. 调用格式 `xlswrite(FILE,ARRAY,SHEET)`
- `clc` – 清屏. 清除 Command 窗口显示内容. `clc`
- `clear` – 清除变量, 清楚工作空间存储的变量. 其后加变量名可以只清除指定变量, 清除全部变量直接使用即可, 清除某个变量 `vari`, `clear vari`
- `save` – 保存工作空间变量. 保存全部变量, `test.mat`; 保存某几个变量, `p = 1; q = [1 2]; save('data.mat', 'p', 'q');`
- `load` – 从硬盘加载数据. 用法和 `save` 类似.
- `diary` – 保存 Command 屏幕输出, 由于屏幕输出延时, 可能并没有保存数据, 因此有两个方案来应对; 其一, 加入 `pause`; 其二, 判断是否已保存数据, 直到有数据才执行 `diary off`
- `which` – 查看函数的文件路径. 查看函数 `fmincon` 的位置, `which fmincon`
- `whos` – 查看当前工作空间变量的属性. 直接使用为查看全部变量属性, 查看 `var` 变量的属性, `whos var`
- `saveas` – 保存图像 (Figure 或仿真框图). 保存当前 figure 为 jpg 图像, `saveas(gcf, 'output', 'jpg')`

- textscan – 格式化读取文件或字符串.
- regexp – 用正则表达式匹配内容.
- deal – 多个变量同时赋值,  
 $C = \{\text{rand}(3) \text{ ones}(3,1) \text{ eye}(3) \text{ zeros}(3,1)\}; [a,b,c,d] = \text{deal}(C\{:\});$

## 4 一些技巧

这里所指技巧具有专题性的意义, 是面向解决方案的, 而不是面向功能的讲解.

### 4.1 内存溢出及解决方案

**内存溢出** 所谓内存溢出, 也就是计算时出现的 Out of memory.

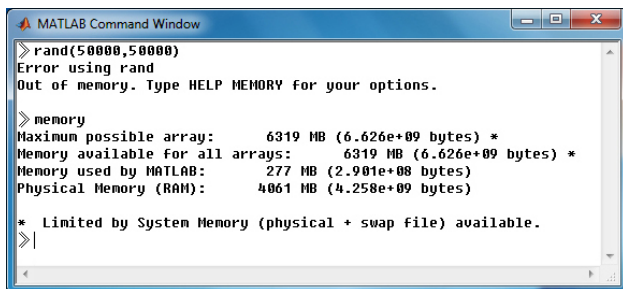


Figure 4: 内存溢出和内存查看

#### Example 13: 生成随机矩阵

---

```
matrix = rand(2e+5,2e+5);
```

---

*Note 4.1.* 逐渐增大或减小维数, 这样可以查看设备能存储最大矩阵的维数.

**内存溢出解决** 简而言之, 开源节流.

- 开源方法
  - 增加 RAM. 买个大点的内存条装上;
  - 以 “no java” 方式启动 MATLAB.(这是在节 MATLAB 的流, 开程序执行的源)
- 节流方法

- 数据本地存储. 将数据存储到本地, 需要时再导入;
- 使用已有的变量, 即时删除临时变量 (不再需要的变量);
- 以函数封装. 将程序中某几部分的代码封装为 `function` 调用, `function` 调用只输出最后需要的数据, 其间的临时变量在每次调用完 `function` 之后都会自动删除.

## 4.2 自定义函数帮助

在 MATLAB 中可以用输入 `help` 加函数名来获得熟悉使用方法及例子. 我们期望对自己编写的函数也实现这样的功能, 那么有两个关键点, 一是参照 MATLAB 的内置函数注释格式写好注释 (或者参照下面的示例), 二是确保此函数路径已经添加. 接下来就可以通过 `help myfunction` 这样的形式来查看了.

Example 14: 自定义帮助

---

```
function outputs = myfunction(inputs)
%MYFUNCTION this function is bulabula...
% decrible your function here
% see also function1 function2

% Reversion: 22-Aug-2013
% 2013/08/22 16:58:27
```

---

## 4.3 图像导出

除了一般的截图方法, 有这样几种方法.

- `print('-dpdf', pdf_name);`, 生成图像为 pdf 格式.
- 直接设置导出, File - Export Setup - Export, 通过 Rendering - Resolution 可以设置图像质量.
- `saveas(gcf, 'output', 'jpg')`, 导出当前 figure 为名为 output 的 jpg 格式图像.

## 4.4 公式打印

MATLAB 支持  $\text{\LaTeX}$  的公式输出, 在两个地方可能用到公式输出.

- 图片上的公式
- 代码发布注释中的公式

Example 15: 图像中的公式输出

---

```
y = title('$$y = \sqrt{x}$$'); % 标题
set(y, 'interpreter', 'latex'); % 用LaTeX翻译标题
```

---

## 4.5 重复向量以构造矩阵

首先解释一下什么是向量重复构造矩阵, 比如有一个向量  $[1 \ 2 \ 3]$ , 现在我打算将它构造成  $[1 \ 2 \ 3; 1 \ 2 \ 3; 1 \ 2 \ 3]$ , 这就是此小节要解决的内容.

Example 16: 重复向量以构造矩阵

---

```
%% repmat 函数方法
clear
x = 1:1:20000; N = 30000;
tic
X = repmat(x, N, 1);
toc
```

```
%% 矩阵乘法方法
clear
x = 1:1:20000; N = 30000;
tic
Y = ones(N,1)*x;
toc
```

```
%% 下标方法
clear
x = 1:1:20000; N = 30000;
tic
Z = x(ones(N, 1), :);
toc
```

---

```
Elapsed time is 25.953179 seconds.
Elapsed time is 217.866385 seconds.
Elapsed time is 13.400210 seconds.
```

---

实际上, `repmat` 用到了第三种, 在其源码中可以看的到, 但它更加通用. 如果是数值, 首推第三者. `repmat` 的源码可以通过 [which repmat](#) 查看其位置. 另外需要提及, 测试时每段前面当加 `clear`, 这是要消除由于内存的原因影响后面程序的执行效率.

还有一种方式, 这种方式可以列重复挨着:

---

```
>> blkproc([1 2 3 4 5], [1 1], @(x)repmat(x, 3, 2))

ans =

1    1    2    2    3    3    4    4    5    5
1    1    2    2    3    3    4    4    5    5
1    1    2    2    3    3    4    4    5    5
```

---

*Note 4.2.* 可以考虑用上面的某种方法用向量重复来构造三维的矩阵.

# 5 高效编程

## 5.1 分析代码性能

通过 Profiler 这个工具来查看执行程序中占用时间比例, 并作有针对性的优化这样可以.

Example 17: 用 Profiler 分析代码性能

---

profile on	% 打开profile
plot(magic(35))	% 要执行的代码
profile viewer	% 查看分析结果

---

分析结果包括每部分的占用时间比例, 没个函数或语句的执行时间, 点击链接可以进入到下一层, 以此来逐层查看.

*Note 5.1.* 也可以通过在 Profiler 中的 Run this code 输入要分析的文件名或代码, 点击 Start Profiling 按钮开始分析. 打开 Profiler 的方式除了执行 profile viewer, 还可以通过在 HOME 下的 Run and Time 按钮来打开.

## 5.2 预分配内存

为矩阵（或向量）预分配内存, 并尽量避免改变矩阵的大小. 为什么预分配内存呢? 实际上, 申请新的内存本身就是一种耗费. 每当申请需要新的内存存储变量时, MATLAB 都会先查找是否有足够并且逻辑上是连续的内存空间来存储（若没有则会内存溢出, 无法继续执行程序）. 对于矩阵, 每次对它添加元素时, 其所占内存就在改变, 这种改变在地址上的本质是, 它不是在原有的基础上添加, 而是找到一块合适的地址之后, 存储到新的空间上, 并删除原来空间上的数据.

Example 18: 是否预分配内存效率对比

---

n = 1e+7;	
%% 未初始化赋值	
tic	
for i=1:n	
a(i) = i;	
end	
toc	
%% 初始化后赋值	
tic	
b = zeros(1,n,'double');	
for i=1:n	
b(i) = i;	
end	

---

toc

---

Elapsed time is 4.193847 seconds.

Elapsed time is 0.177962 seconds.

---

## 5.3 临时变量

这里所谓临时变量,是指那些并不是关键的要存储的变量,而只是临时使用.尽量少使用临时变量,或者使用少量的临时变量,毕竟申请内存是一件耗费的事情.

## 5.4 MATLAB 建议

根据 MATLAB 建议编写程序(有时候编程会出现黄色下划线).但不更改也并不会影响程序的执行.这些建议包括一些函数使用的提醒,比如将来会移除这个函数,也包括联系上下文实现某一功能时的参数建议,还包括变量的使用建议.

## 5.5 矩阵存储方式

矩阵的存储方式以列优先存储,即我们在计算时尽量采用列优先的方式.

Example 19: 矩阵不同存取方式效率对比

---

```
n = 10000;
A = rand(n , n);
B = zeros(n , n);

%% 以行存取
tic
for i = 1:n
    B(i , :) = A(i , :);    % 以行方式赋值
end
toc

%% 以列存取
tic
for i = 1:n
    B(:, i) = A(:, i);    % 以列方式赋值
end
toc
```

---

Elapsed time is 24.323625 seconds.

Elapsed time is 0.621799 seconds.

---



## 5.6 函数类型

不同函数类型的执行效率并不一样, 在此仅以符号函数和匿名函数作为对比对象.

Example 20: 不同函数类型效率对比

---

```
% 符号函数
syms fun1(x);
fun1(x) = x^3 + x^2 + x + 1;
tic
fun1(5);
toc

% 匿名函数
fun2 = @(x)x^3 + x^2 + x + 1;
tic
fun2(5);
toc
```

---

```
Elapsed time is 0.006139 seconds.
Elapsed time is 0.000074 seconds.
```

---

## 5.7 大数据处理

直接的描述就是提前给大数据分配内存. 过多小数据分配内存后形成的内存零碎化, 这将在后面为大数据寻找合适的连续内存地址消耗更多时间.

## 5.8 并行计算

这里所指的并行计算是针对一台多核的电脑. 所谓并行计算, 是指让多个工作同时进行, 从而节省计算时间. 这里以 `parfor` 作为解释对象, 要用到的是 `matlabpool`.

`parfor` 即 `parallel for` (并行的 `for` 语句), 可以看作是把 `for` 循环执行的内容改造成了同时执行的内容. `matlabpool` 用于打开多核, 默认是并没有打开的.

实现并行计算包括两个操作, 一是打开多核 (或者 `worker`), 二是要保证循环计算之间并不存在关联性.

Example 21: 并行计算 `parfor` 与 `for` 效率对比

---

```
matlabpool open 2          % 打开两个worker
M = 200; a = zeros(M,1);   % 200 次计算

% for
tic
for i = 1:M
    a(i) = max(eig(rand(M))); % 取随机矩阵最大特征值
end
```

---

```

toc

%% parfor
tic
parfor i = 1:M
    a(i) = max(eig(rand(M)));
end
toc
matlabpool close          % 关闭所有worker

```

---

```

Elapsed time is 11.092022 seconds.
Elapsed time is 6.325050 seconds.

```

---

尽管通常不是用循环来实现的内容,通过转化为不相关的循环计算,就可以用并行计算来实现.

*Note 5.2.* 在 MATLAB 右下角可以看到打开 worker 的数量.

## 6 额外兴趣

### 6.1 拍照

用 MATLAB 调用硬件拍照简单示例 (确保有可用的摄像头, 并一段一段执行):

#### Example 22: 拍照

---

```

%% 打开摄像头
vid = videoinput('winvideo',1); % 默认格式
preview(vid);

%% 拍照并预览
data = getsnapshot(vid);
image(data);

%% 保存照片为 photo.jpg
imwrite(data,'photo.jpg');

```

---

## A Appendix

### A.1 MATLAB 资源

- MATLAB 中文论坛 [www.ilovematlab.com](http://www.ilovematlab.com)

在这里可以提出自己的问题, 也可以看看别人的问题, 通过问问题和浏览来开阔思维 and 了解.

- MATLAB File Exchange <http://www.mathworks.com/matlabcentral/fileexchange/>  
通过搜索可以找到已经实现了一些打包函数, 比如三维插值、自然排序的实现等等.
- cheat-sheets <http://www.cheat-sheets.org/>  
这里有很多快速了解各种语言的文档, 其中就有 MATLAB 的表格.