

# MATLAB 学习总结

潘忠显

[panzhongxian@126.com](mailto:panzhongxian@126.com)

# 前言

接触 MATLAB 是从大一的选修课以及线性代数上机实验开始的，真正开始用还是从大四进入实验室开始。越用越觉着 MATLAB 强大，我所了解的仅仅是九牛一毛。不做总结的工作只能叫工作，而不能成为自己的经验。因此，把之前碰到的若干问题进行总结，便于查询。如果它能帮助到别人一点，那我会非常欣慰。

另外的一个原因是若干师兄和同学都警告我，光会 MATLAB 找不到“好”工作，加个“好”字也只是委婉的说法。所以我得赶紧跟 MATLAB 先做个了结。

最了解 MATLAB 的还是软件的 Help，它知道每个函数的具体用法，而且还有丰富的例子；而对各种工具箱最了解的莫过于对应的 MATLAB User's Guide；作为补充的是 MathWork 的官网，在 File Exchange 中有各种大神编写 MATLAB 中没有但非常常用的代码。

做事情难免犯错误，如果你发现这个总结里的错误，欢迎并感谢通过邮箱或者其他方式告诉我。

# 目 录

前 言	1
<b>1 软件基本操作</b>	<b>1</b>
1.1 个性化的MATLAB	1
1.2 安装工具箱和卸载工具箱	2
1.3 常用命令	2
1.4 help 和 doc 的区别	3
1.5 快捷键	3
1.6 遇到安装到 99% 报错问题	4
<b>2 向量和矩阵操作</b>	<b>5</b>
2.1 产生向量和矩阵	5
2.2 列优先	6
2.3 矩阵的操作	6
2.4 bsxfun 函数的使用	7
2.5 排列组合的应用	7
2.6 “假、大、空”	8
<b>3 高效的程序</b>	<b>10</b>
3.1 避免使用 for	10
3.2 下标问题	11
3.3 稀疏矩阵	12
3.4 for 的层次问题	12
3.5 变量初始化	13
3.6 使用 Code Analyzer	14
3.7 使用 Profiler	15
<b>4 自己写函数</b>	<b>17</b>
4.1 函数的结构	17
4.2 全局变量	18

目 录	3
4.3 递归函数	19
4.4 提示信息	20
4.5 内嵌函数与匿名函数	20
<b>5 数据类型定义与转换</b>	<b>22</b>
5.1 数据类型	22
5.2 类型判别	23
5.3 格式转换	24
<b>6 画图</b>	<b>26</b>
6.1 General	26
6.2 figure 的属性	27
6.3 axes 属性	27
6.4 颜色	33
6.5 漂亮的注释	35
6.6 创建子图	38
6.7 其他画图函数整理（待整理）	41
<b>7 文件操作</b>	<b>43</b>
7.1 文件与路径	43
7.2 低层次文件读写	45
7.3 dlmread 和 dlmwrite 函数	46
7.4 textscan	47
7.5 xlsread 与 xlswrite 函数	48
7.6 图像、声音读写	49
<b>8 并行计算</b>	<b>50</b>
8.1 并行运算框架	50
8.2 配置 matlabpool	51
8.3 parfor 并行	52
8.4 测试并行循环	53
8.5 parfor 中的变量类型	54
8.6 两个具体问题	55
<b>9 MATLAB Coder ——生成独立的 C/C++ 代码</b>	<b>58</b>
9.1 准备工作	58
9.2 生成 C 代码基本步骤	58
9.3 保证代码生成的正确性	62
9.4 代码转换中的变量	65
9.5 不确定尺寸的输入	66

目 录	4
9.6 mxArray 数据结构体	68
9.7 其他几个问题	69
<b>10 MATLAB Compiler ——生成 dll 与 exe</b>	<b>72</b>
10.1 一些基本概念	72
10.2 创建独立应用的基本步骤	73
10.3 创建 C/C++ 共享库	74
10.4 调用动态链接库	75
10.5 mxArray 类的介绍	77
10.6 注意的问题	80
10.7 mex, mcc, mbuild	80
<b>11 人机交互</b>	<b>81</b>
11.1 一些 UI 开头的	81
11.2 一些以dlg结尾的	82
11.3 waitbar	84
11.4 figure 和命令窗中的交互	85
<b>12 简单的网络应用</b>	<b>86</b>
12.1 urlread 函数	86
12.2 sendmail 函数	89
12.3 ftp 函数	89
12.4 教务处网站的修复	90
<b>13 MATLAB 文件的发布</b>	<b>92</b>
13.1 publish 需要的文件格式	92
13.2 万事俱备，只欠 publish	94
13.3 pcode 的使用	96

# Chapter 1

## 软件基本操作

### 1.1 个性化的MATLAB

安装完 MATLAB 软件之后，我们可以做一些个性化设置，以方便不同的操作习惯。

窗口分布：MATLAB 有不同的窗口，基础的 MATLAB 教材中都会有介绍，这里只是提一下。比如：Current Folder, Command Window, Workspace, Command History, Editor 等重要窗口。通过拖拽可以改变窗口的位置，另外右上角的最小化、最大化、停靠于主界面、从主界面解锁等按钮以及 Desktop 目录下的选项，可以帮助打造适合自己编程习惯的 MATLAB。当自己搞的太乱了想恢复系统默认设置时可以点击 HOME → Layout → Default。

快捷键：每个人都有自己经常用的命令。比如我经常敲 `close all;`  
`clear;` `clc;` 三个命令，为了方便，可以将三个命令创建一个快捷键，贴到 MATLAB 工具栏上。具体操作是：将命令敲入 Command Window 中，选中命令拖到 SHOTCUTS 标签中，输入对应的 Label 和 Icon 即可；也可以点击 SHOTCUTS → New Shortcut 进行设置。

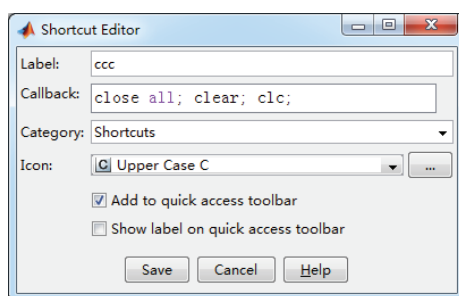


图 1.1: 自定义快捷键

字体设置: HOME → ENVIRONMENT → Preference → Fonts → Custom, 选择命令框、Editor 等位置对应的字体, 改变字体和字号。

## 1.2 安装工具箱和卸载工具箱

使用别人的工具箱, 只需要将相应的文件夹添加到MATLAB 的默认路径中即可, 可像调用普通函数一样调用工具箱函数。添加工具箱路径: File → Set Path → Add with Subfolders... → 选择工具箱文件夹 → Save。

决定不再使用工具箱之后, 可直接将文件夹删除, 但是每次打开 MATLAB 都会给 Warning, 提示没有相应的工具箱, 这时候再删除工具箱原来的路径即可: 在 /toolbox/local/pathdef.m 中直接搜索, 删除。

## 1.3 常用命令

MATLAB 有些命令很基础, 不管是哪方面的代码都会用到。下边列出了一些命令, 只是给出基本解释, 更详尽丰富的用法请参考 MATLAB 里边的 Help。

version	查看 MATLAB 版本信息
ver	查看 MATLAB 版本及工具箱版本
memory	查看可使用的内存空间
clc	清屏
close	关闭画图窗口
clear	清除变量
disp	显示字符串、变量
edit	编辑文件
delete	删除文件
pwd	当前路径
cd	改变路径
echo	回显
syms	定义符号变量
figure	开辟画图窗口
exit/quit	退出 MATLAB
dir	列出当前路径下文件
save	保存变量
format	控制输出格式

## 1.4 help 和 doc 的区别

重复一下 Help 的重要性，个人觉着查看帮助是学习 MATLAB 最重要的途径，没有之一。那么的多 MATLAB 的书都是只是翻译了 Help 而已，基础性的翻译基础性的帮助，专业性的翻译工具箱的帮助，不如自己看。MATLAB 中看帮助文档有两种方式，对应两个用法相同的函数，一个是 help，一个是 doc。

help mean 和 doc mean 都能调出一堆英文的解释。前者在命令窗中显示，显示的内容是对应函数中的函数名之后的注释，直到第一行非注释为止。介绍函数的用法，支持哪些类型输入，支持哪些类型的输出。后者在 Help 浏览器中显示帮助，内容与前者类似，但是由于是在浏览器中，所以文字字体和对齐会有更好的处理，同时可以通过图像来进行函数的展示，更加直观。所以，个人更推荐用后者，而且打开 Help browser 之后一直开着就行了，可以打开多个标签页，搜索也会保留下历史记录，同下边会提示相关的函数。

## 1.5 快捷键

通用	Ctrl + A	全选
	Ctrl + S	保存
	Ctrl + C	复制
	Ctrl + V	粘贴
注释	Ctrl + R	注释选定的内容
	Ctrl + T	取消选定的注释
	Ctrl + J	将注释自动对齐
缩进	Ctrl + [	Tab
	Ctrl + ]	Shift + Tab
	Ctrl + I	智能缩进
Cell	Ctrl + Enter	执行 Cell 的所有命令并留在本 Cell
	Ctrl+Shift+Enter	执行 Cell 的所有命令、跳到下一 Cell
	Ctrl+Up/Down	跳到上/下一个Cell
调试	F5	运行当前文件
	Shift + F5	退出调试模式
	F9	运行选中的语句
	F10	单步执行（不进入函数）
	F11	单步执行（进入函数）
	F12	加入/删除断点
其他	Tab	自动填补命令或变量名
	Ctrl + D	打开选中的函数



## 1.6 遇到安装到 99% 报错问题

问题描述：在曾安装过 MATLAB 的电脑上再重新安装时，可能会在安装到 99% 时报错。在此情况下虽然能打开 MATLAB，但是输入命令会蹦出一堆的 Warning，无法正常运行。

问题解决：

①.在 99% 报错的基础上，跳过，打开 matlab.exe，然后看到一堆 Warning;

②.恢复默认搜索路径，在 Command Window 里输入：

```
restoredefaultpath; matlabrc;
```

③.改变文件路径到安装目录：

```
cd 'D:/Program Files/MATLAB/R2011b/toolbox/local'
```

④.保存路径：savepath pathdef.m

⑤.重启 MATLAB，就一切正常了。

## Chapter 2

# 向量和矩阵操作

矩阵操作是写 MATLAB 程序的基础。大部分基础性 MATLAB 书籍对此都有非常详细讲解。这里不介绍基础，只总结点经验。

### 2.1 产生向量和矩阵

要利用向量和矩阵进行运算，首先就要先产生向量和矩阵。除了元素逐个输入之外，对一些特殊的向量和矩阵，还可以使用一些简单的表达形式。

首先，常用的就是 `1:2:200` 这种形式。这个表达式的含义是从 1 开始每隔 2 取一个数，取到不超过 200。1 一定是序列的第一个，最后的 200 可能不在结果之内，他只是设定了一个上限。另外，类似的还可以 `(1:100)*2` 的形式。前者是需要产生在范围的数，后者是为了产生固定个数的数。`1:0` 和 `1:-1:0` 的结果是不一样的，前者为空，后者是 `[1,0]`。

其次，我们可以使用 `linspace` 和 `logspace` 等特殊的函数产生一个向量，作用于 `':'` 有类似之处，但是在确定向量长度和范围之后，可以带来一些方便。比如我们要在 0 到 20 之间产生 100 个点（包含端点），那么相邻两个数之间的间隔并不恰好是  $20/100$ ，而是  $20/99$ ，我们可以使用 `linspace(0,20,100)` 命令直接得到这个向量。同样的可以使用 `logspace` 产生对数间隔的向量。

最后，当我们要组合两个或者多个向量中的元素时，可以使用 `meshgrid` 函数。假设我们有一个以  $\alpha$  和  $\beta$  作为自变量的函数  $f(\alpha, \beta) = \sin(\alpha) + 2 * \cos(\beta)$ ，我们想求出在  $\alpha \in (0, \pi)$  和  $\beta \in (-\pi, \pi)$  内的所有值，这时就可以通过 `[alpha,beta] = meshgrid(0:0.1:pi,-pi:0.1:pi)` 得到矩阵 `alpha` 和 `beta`，两个矩阵大小相同，所有相同位置的两个元素组合起来，就遍历了所有的可能组合。再通过以下两行命令即可得到 `f` 并做出图形：

```
1 f=sin(alpha)+2*cos(beta);  
2 mesh(alpha,beta,f)。
```

还有一些特殊的函数也能产生向量和矩阵，比如 `ones()`, `zeros()`, `rand()`, `randn()`, `Inf()`, `cell()` 等函数。这些函数的具体用法会在后边的章节里做介绍。

## 2.2 列优先

一些语言在存储数组时使用的行优先的原则，而 MATLAB 里存储的矩阵遵循列优先原则。所以好多函数都依据行优先的原则。

`max`, `mean`, `median`, `min`, `mode`, `std`, `var` 几个函数对矩阵做运算的时候，分别对各列取最大、均值、中位数、最小、众数、标准差、方差，得到一个行向量；如需要对行操作，需加参数指定维数。

`reshape` 是按照列来重新排列的，所以要按照行排列时，需要先转置再进行重排再进行转置。

以及以后将要介绍的循环的层次问题中，依照列优先的原则，逐个依次访问内存的效率要比按照其他规律跳来跳去的访问要高效的多。

## 2.3 矩阵的操作

常见的的矩阵的操作除了加减乘除之外，还有下边这些操作。

拼接：大部分情况下 `cat` 可以使用中括号拼接的形式代替，方便简洁，例如 `A=[B;C,D]`；

翻转：`flipud` 和 `fliplr` 能够进行矩阵的翻转；

重复：`repmat` 函数使用一个小矩阵不断重复扩展成一个大矩阵；

移位：`circshift` 能进行矩阵的循环移位，通过参数控制移动方向；

排序：`D=sortrows(A,[1 7])`；这句话的意思将 A 按照第一列排序，如果某两行第一列相同再按照第七列进行排序，默认升序，降序在列标前加负号。看起来 MATLAB 也能像 EXCEL 很方便地处理数据；

转置：符号 `'` 表示共轭转置，而符号 `.'` 才表示普通转置；

扩展：`y=rectpulse(x,nsamp)` 是通信系统工具箱中的函数，`rectpulse` 是矩形成形的意思，函数作用是将向量 `x` 中的每个元素重复 `nsamp` 次，构成一个 `length(x)*nsamp` 长的新向量；

插零：`y=upsample(x,n,phase)` 也是通信系统工具箱中的函数，`upsample` 是上采样的意思，函数作用是将向量每一个元素填零扩展成 `n` 个元素，构成一个 `length(x)*nsamp` 长的新向量，`phase` 用来控制元素插入的位置；

`i` 和 `j`：`i` 和 `j` 是虚数单位，但是有人喜欢用 `i` 和 `j` 作循环变量，当 `i` 被赋值之后就不再是虚数单位了，此时可以使用 `1i`、`1j`。编程时，循环变量最好能有清晰的含义，不选用 `i`、`j`、`k` 等；同时，虚数单位直接使用 `1i` 和 `1j` 就行了。

## 2.4 bsxfun 函数的使用

矩阵与一个数做运算，或者矩阵与另一个相同大小的矩阵做运算，都可以很方便的实现。前者直接进行加减乘除运算，后者需要进行加点运算。但我们实际操作中往往需要对矩阵行列进行操作，比如一个矩阵的每一列分别乘以一个因子，或者每一行减去改行的平均值。针对后边提出的这种需求，最原始的矩阵操作的方法是将后边的行或列向量进行扩展，扩展成为相应大小的矩阵，然后进行矩阵间的点操作。

本小节以对 3 x 3 的魔术矩阵 A 的每一列分别乘以 [1,2,3] 中每一列的元素为例，介绍 BSXFUN 函数的使用。其中 y1 是通过向量扩展成矩阵的方法得到的，y2 就是使用 bsxfun 得到的，附加的 y3 是将 A 的每一行分别减去其每一列的均值。

```
1 x = magic(3);
2 a = 1:3;
3 y1 = x.*repmat(a,[3,1]);
4 y2 = bsxfun(@times,x,a);
5 y3 = bsxfun(@minus,x,mean(x,2));
```

## 2.5 排列组合的应用

涉及随机排列、取若干组值等问题，可以考虑通过排列组合得到下标的方式来精简程序。

在对包含有特征和类型的一组数据集做训练和识别时，我们往往需要从中拿出若干组进行训练，剩下的若干组用于识别。如果通过 rand 同样大小的随机向量并设置大小比较门限得到一个 logical 型的下标向量，得到的数据个数往往不恰好是我们所期望的，因为 rand 只是产生随机数。并不能保证 rand 出的 100 个数中小于 0.2 的恰好是 20 个，所以这种方法不可行。

下面介绍通过产生随机排列下标来实现随机挑选。具体思路是使用 randperm 函数产生一个 1:n 的随机排列，其中 n 为数据总长度，用此数组将原始数据进行新的乱序排列，取出前若干组就是随机选择的若干组。

```
1 dataSet = rand([1,100]);
2 ind = randperm(length(dataSet));
3 ind1 = ind(1:80);
4 ind2 = ind(81:end);
5 trainSet = dataSet(ind1);
6 recogSet = dataSet(ind2);
```

有时我们还需要将一个数组的所有的排列顺序都列出，这就需要用到全排列函数 perms。例如将一个 5x1 的数组进行全排列得到其所有排列形式，程序如下：

```

1 data=rand([1,5]);
2 dataSet=data(perms(1:length(data)));

```

## 2.6 “假、大、空”

相声讽刺不让说“假、大、空”，但是在 MATLAB 中假、大、空还是很有作用的。

NaN 表示非数值，有些运算会得到非数值，比如取余运算除数为 0，`rem(x,y)` 就会得到 NaN。另外，在画图时遇到 NaN，则会忽略对应的点，所以当我们只需要画一部分时，就可以用 NaN。

```

1 [x,y]=meshgrid(-1:0.01:1);
2 z=-x.^2-y.^2;
3 subplot(121)
4 mesh(x,y,z)
5 subplot(122)
6 z(z<-1)=NaN;
7 mesh(x,y,z)

```

执行上述代码将得到图 2.1。

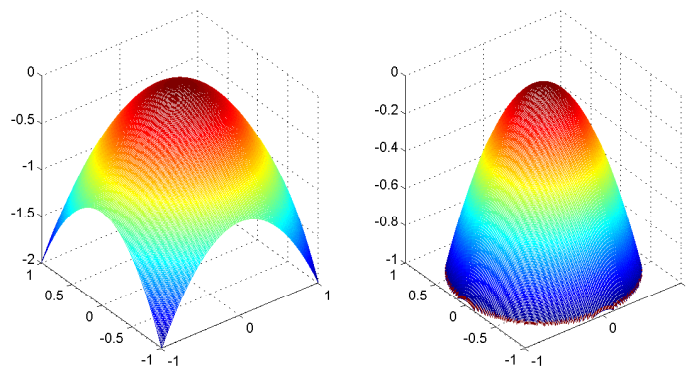


图 2.1: NaN 在画图上的应用

Inf 表示无穷大，有些运算会得到无穷大，有正负无穷大的区分，比如除数取 0 就会得到 Inf，`log(0)` 会得到 -Inf。Inf 还可以用在 `int` 函数中的作为积分限。Inf 在用于大小比较时也有用得着的地方，比如拿出一个最小值后将这个值置为 Inf 那么下一次再找最小值就找出了次最大值，等等。

[] 表示空矩阵，令矩阵的某一行/列为空就可删除改行/列；进行可变矩阵的拼接时，初始矩阵可赋值成空矩阵；

$\sim$ 在逻辑运算时表示取非， $\sim=$ 表示不等于，而当 $\sim$ 表示作为输出时，表示不输出该结果。比如 $[\sim, \text{indm}] = \max(a)$ ，只需要最大值位置，而不需要确切最大值时，就没必要浪费一个变量去保留最大值。

## Chapter 3

# 高效的程序

MATLAB 对于矩阵和向量化的操作是非常快的，但是为了对模型进行验证，我们通常需要进行大量的蒙特卡罗实验。这时候写程序就更需要注重高效性了。可能一个低效的程序需要跑两三天，而一个能得到同样结果的高效程序可能只需要一个小时甚至更短（不过有的人好像真的很享受让程序跑个两三天的过程）。本章将介绍若干种提高效率的方式，用一种提速几倍，多种方法的组合使用的加速效果是各种效果的乘积，不可小觑。帮别人加速过的几个程序中，随随便便一改就能提速几十倍。

当程序跑的很慢的时候，总有人会说，我电脑是 4 核的，可以改成并行计算。在后边的章节也会讲到并行计算的应用，但由于系统运行需要一定的资源，四个核的并行计算提速肯定不到 4 倍，很可能不如改掉一个 for 循环的效果。另外，四个核都分配给 MATLAB 的结果就是，跑一会儿电脑就会非常卡。所以为了电脑和自己，尽量多用本章的方法改进一下程序吧！

### 3.1 避免使用 for

每当跟别人讨论用向量运算代替 for 的时候，总有人说使用 for 更直观更容易理解。实际情况确实对方往往未真正理解向量化操作，直到我用代码说服他们，最后才承认向量化的运算甚至比 for 循环来的更直观。如果没有什么迭代的关系，那还是选择向量运算吧。

MATLAB 没有解释为什么它的向量化运算会比 for 循环高效的多。

问题描述：有 5 个杯子，每个杯子存了 20 个单位的水。每个杯子以一定的概率产生 0 和 1，每次从产生 1 的杯子中取出 5 个单位，平均分配到剩下的四个杯子中；如果不足 5 个单位，将剩下的所有的水倒出。执行上述操作 10000 次。当然这个问题只是个问题，不去讨论他的实际意义。

曾经有人用 for 循环写了一个解决方法，在每一次的操作中，先用 for 循环产生了一个数组，再来一个 for 循环加 if 统计要倒出的水的杯子，再来一个 for

循环加 if 将水倒入剩下的杯子中。这样真的很直观么？那就再看看去掉 for 循环的程序吧。

```

1 Water=[20 20 20 20 20];           % Initialize Water
2 P=[0.5 0.9 0.6 0.5 0.4];         % Initialize the Probability
3
4 for index=1:10000
5     tmplabel = rand([1,5])<P;      % Generate the label
6     out1 = tmplabel & Water>5;      % Cups to give 5 units
7     out2 = tmplabel & Water<5;      % Cups to give all
8     in = ~tmplabel;                % Cups to accept
9
10    if sum(in) > 0
11        exchwater = sum(out1.*5 + out2.*Water); % Water to exchange
12        Water = Water - out1.*5 - out2.*Water; % Take out
13        Water = Water + in*exchwater/sum(in); % Pour in
14    end
15 end

```

程序简洁明了，而且逻辑也非常清晰。

## 3.2 下标问题

避免使用 for 而习惯使用矩阵操作，就必须习惯巧妙地使用矩阵下标或者线性索引值。

矩阵的下标矩阵是跟该矩阵一样大的，用 logical 型的变量来标识是否选择对应位置的变量。

已知一个矩阵 A，将其中大于 0.5 的元素都取出存到 B 中，而令 A 中所有大于 0.5 的元素置零：

```

1 A = rand(10);
2 idx1 = A > 0.5;
3 B = A(idx1);
4 A(idx1) = 0;

```

其中 idx1 就是一个 10x10 logical 矩阵。

我们比较熟悉使用 A(1:3, [3,6]) 来取出 A 中第 1 到 3 行、第 3 和第 6 列的元素，并形成矩阵。那么我们能不能使用将上述两种索引变量的方式结合起来呢？答案是肯定的。

```

1 A = rand(10);
2 idx1 = logical(randi(2,[10,1])-1);
3 idx2 = 1:5;
4 B = A(idx1,idx2);

```



再次强调，0 和 1 必须是 logical 型的，所以这里需要加了一个 logical 的强制类型转换。

另外介绍有监督分类（Supervised Classification）中可以使用到的一个技巧。从文件中读取了 200 行数据，其中前 100 行是类型 1 的属性，而后 100 行是类型 2 的属性。要测试模式识别的效果，需要每次随机从各类中分别取出 90 行数据作为训练集合，而剩余的 10 行作为测试集合。

```
1 % data is a 200xN array;
2 Train_idx = [randperm(100), randperm(100)]>10;
3 Test_idx = ~Train_idx;
4 Train_data = data(Train_idx, :);
5 Test_data = data(Test_idx, :);
```

再举一个例子，一次性找到矩阵中最大值的位置。

```
1 a = rand(3);
2 [~, ind] = max(a(:));
3 [ind1, ind2] = ind2sub(size(a), ind);
```

通过这个例子，引出两个函数——ind2sub 和 sub2ind，进行 Subscripts 和 Linear index 的转换，即下标和线性索引之间的转换。下标是使用圆括号，分别输入数据的第 i 维坐标，线性索引是指按照列、行、页的顺序进行编号的一维索引值。

### 3.3 稀疏矩阵

这种方法针对稀疏矩阵，转换成稀疏矩阵形式（Sparse）进行运算，速度会得到不同程度的提升，密度越低提升越明显。具体看下面的代码。

```
1 a_sparse = sprand(3000,3000,0.0001);
2 a_full = full(a_sparse);
3
4 tic; c1 = a_full*a_full; toc;
5 tic; c2 = a_sparse*a_full; toc;
6 tic; c3 = a_sparse*a_sparse; toc;
```

三种情况耗费时间差别很大，具体如下：

Elapsed time is 3.835104 seconds.

Elapsed time is 0.076895 seconds.

Elapsed time is 0.000130 seconds.

### 3.4 for 的层次问题

如果非钟情于使用多层for时，应将最内侧的循环改成对固定一列上元素的

操作，这同样是因为“列优先原则”，对内存访问效率更高。

```
1 a=rand(10000);
2
3 tic
4 for ind1=1:10000
5     for ind2=1:10000
6         a(ind1,ind2)=1;
7     end
8 end
9 toc
10
11 tic
12 for ind1=1:10000
13     for ind2=1:10000
14         a(ind2,ind1)=1;
15     end
16 end
17 toc
```

上述例子中两个两层循环虽然完成了相同的功能，但是所耗费的时间却有较大的差别，如果层次增加，效率会相差更大：

Elapsed time is 2.420421 seconds.

Elapsed time is 0.504988 seconds.

### 3.5 变量初始化

MATLAB是支持动态开辟空间做法的，一个变量可以任意的扩充和缩减。但是动态开辟空间的做法往往使分配的内存出现不连续的状况，导致对内存访问效率底下，进而影响程序效率。所以如果能够确定变量的确实大小的变量，应该事先声明。对于大变量，尤其应该注意。

```
1 clear;
2 tic;
3 a=[];
4 for ind=1:100000;
5     a=[a,ind];
6 end
7 toc;
```

Elapsed time is 10.427830 seconds.

```
1 clear;
2 tic
3 a=zeros(1,100000);
4 for ind=1:100000
5     a(ind)=ind;
6 end
7 toc;
```

Elapsed time is 0.080073 seconds.

从上述例子可以清楚的看到，预分配空间和改变变量大小的效率差异。

## 3.6 使用 Code Analyzer

Code Analyzer 能够分析 MATLAB 程序中潜在的问题和错误，并能够提出修改意见，最大限度的帮助我们减少错误、提高效率。

通常在 Editor 窗口中我们输入若干代码之后，右侧滚动条中会出现橘黄色或红色的小横杠，将鼠标移动到上边之后，就会给出 Warning 或者是 Error，这些提示信息会告诉你对应位置的某些语句存在的警告或错误，有时右侧还有 Details 告诉我们具体的错误已经对应的处理意见。有时会有 Fix 按钮，点击该按钮 MATLAB 会自动修复问题，不过建议慎用 Fix，尽量根据错误提示及建议自己进行修改，因为 Fix 还没有那么智能，有可能会将程序搞乱。

当然除了通过右侧滚动条位置外，我们还可以专门通过 Code Analyzer 的窗口对程序进行分析，生成详细的报告。2013a 之前的版本中，Code Analyzer 在 Editor 窗口中的 Tools 标签下。从 2013a 版本开始，位置在 Home → Code → Code Analyzer。

如果要修改分析规则，比如去掉某类警告的提示，可以从 Preference 中的 code analyzer 选项中进行修改。自行修改了 Warning Found 规则之后，在上述窗口中选择保存当前设置，也可以加载其他的设置，还可以选择使用默认规则。

```
1 for ind=1:10
2     c(ind)=[ind];
3 end
```

对上边的几行程序使用 Code Analyzer 进行分析，分析结果如图 3.1。

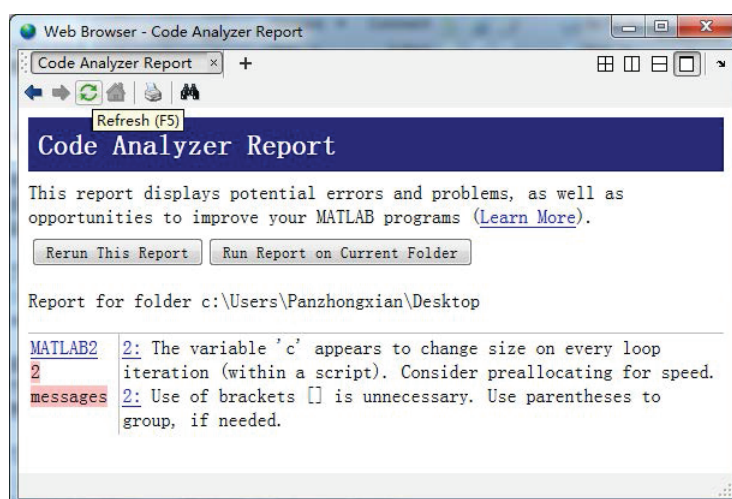


图 3.1: 使用 Code Analyzer 对代码进行分析

Code Analyzer 提示: c 随着循环的尺寸在增加, 建议进行空间预分配以提高速度; 中括号没必要使用, 如果有必要可以使用圆括号。当然这只是建议, 比如前一条建议我们可以采纳, 后一条我们只需要把中括号去掉就可以了, 而没必要使用圆括号。

### 3.7 使用 Profiler

Profiler 是一个使用 profile 函数的 GUI 界面, 通过分析程序中每句话运行多长时间以及重复多少次, 给程序员改进程序提供线索和依据。2013a 之前的版本中, Profiler 在 Desktop 标签下。从 2013a 版本开始, 位置在 Home → Code → Run and Time。

图 3.2 是我在帮别人分析一段代码时使用 Profiler 的情况。

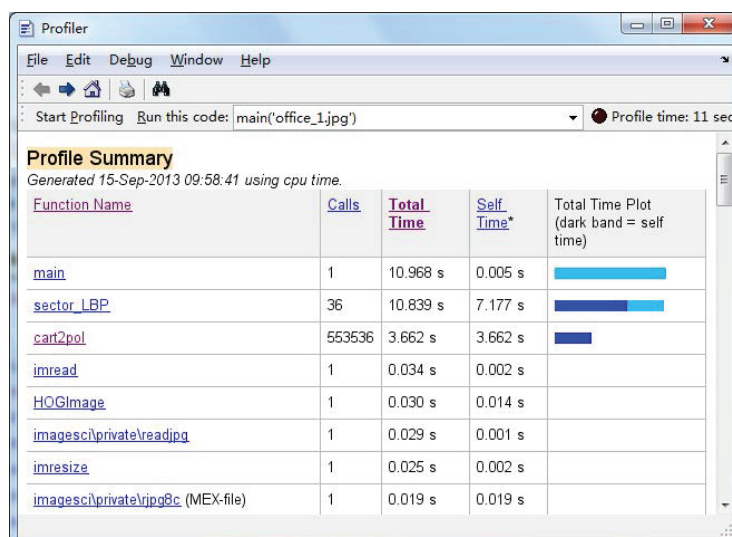


图 3.2: 使用 Profiler 对代码进行分析

上述代码分析列出了运行 main 函数时所有被调用函数的耗时。发现 sector\_LBP 和 cart2pol 占了绝大部分的时间, 而 cart2pol 也是在 sector\_LBP 中被调用。再看看 sector\_LBP 函数内部的函数运行时间分布, 如图 3.3 所示。

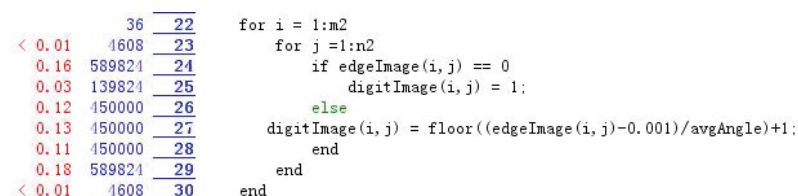


图 3.3: 使用 Profiler 对代码运行时间的分析

红色（第一列）的字表示耗费的时间，而蓝色（第二列）的字表示该语句重复的次数，两层for 循环中的语句竟然重复达50 多万次。在 MATLAB 中，要提高效率一个最基本的就是思想就是减少 for 循环，使用矩阵操作。所以提速的突破口就在于使用矩阵化操作代替嵌套 for 循环的。这一过程就是我们使用 Profile 进行分析的基本过程。

## Chapter 4

# 自己写函数

刚开始大家搞不清什么是函数（function），什么是脚本（script）。简单的分类就是，以 function 开头且文件名和函数名相同的叫做函数，不以 function 开头的叫做脚本。个人理解：脚本一条条函数的堆积，完成了一些而非一个功能，不具有通用性；而函数经常只具有一种功能且可供调用，有很强的通用性。内涵越大，外延就越小。

### 4.1 函数的结构

为了说明一下函数的结构，这里写一个简单的小函数。

```
1 function [result,n]=MySum(varargin)
2 % MySum
3 % [result,n]=MySum(nargin) Sum of two or three elements and
4 % return the number of input.
5 %
6 % Examples:
7 % [result,n]=MySum(3,4) is :
8 %         result =
9 %             7
10 %         n =
11 %             2
12 % [result,n]=MySum(3,4,4,3) is :
13 % ERROR1
14 % ERROR2
15 % Wrong input!
16 %
17 % Author: Panzhongxian@126.com
18 % Date: 2005/06/21 19:24:06
19
20 global str;
```

```
21
22 if nargin==2
23     result=varargin{1}+varargin{2};
24     n=2;
25 elseif nargin==3;
26     result=varargin{1}+varargin{2}+varargin{3};
27     n=3;
28 else
29     my_error1('ERROR1');
30     str='ERROR2';
31     my_error2;
32     error('Wrong input!');
33 end
34
35 function my_error1(str)
36 disp(str);
37
38 function my_error2
39 global str
40 disp(str)
```

第一行也不必非得是 `function`，也可以是注释，但是未注释的命令中第一行必须为 `function` 开头，并且文件名与函数名必须一致。

在 `function` 行前后的注释是为使用者提供的帮助文档。如果写在 `function` 前边，帮助文档就到 `function` 之前；如果在之后，如举得这个例子，就到第一句非注释前为止。这样我们保存之后就可以通过 `doc` 或者 `help` 查看说明了，图 4.1 就是我们通过 `doc` 查看的结果。

写的注释个人的风格不一样，但至少清晰的说明函数的用法。另外可以加上作者的信息和修改的时间，如果有多个版本，还应该注明版本号。

MATLAB 中支持函数输入个数不同、类型不同的重载，所以这里写了一个 `varargin` 就是用于这类重载的。在这个函数中，通过 `nargin` 判断函数输入个数来进行不同的操作，另外还可以用 `nargout` 根据输出的不同进行不同的操作。

两个子函数，是在这个函数文件中定义的新函数，在一个函数文件中定义的函数只能在这个函数中被使用，除了这个函数就是不可见的了。有的地方我们看到，每个 `function` 结束时都是用一个 `end`，这个 `end` 在所有函数中，要么都有，要么都没有，否则会报错。

## 4.2 全局变量

两个子函数中还展示了局部变量和全局变量的区别。

全局变量（`global variables`）区别于局部变量（`local variables`）。MATLAB 语言和其他语言一样，在调用函数之前，将变量进行保存，调用完函数之后，

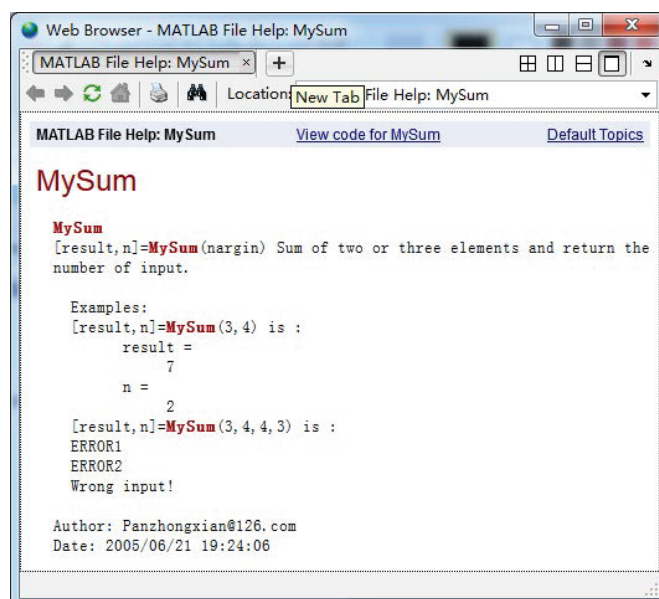


图 4.1: 自己写的函数帮助

再读取保存的变量，而局部变量仅在函数内部起作用，在函数结束时被销毁。

根据这些特点，我们分析一下使用全局变量的必要性。首先，一些程序从意义上需要全局变量，比如多线程操作时，几个线程同时需要读取和修改某个变量的当前值；其次，某变量如果作为参数传入到函数中，需要在函数中创建新的变量并对其进行赋值，而全局变量则免去了这些操作，可提高效率。

global 变量会在第一次声明而非在脚本文件中或者最外层的调用函数中，被初始化为空。每个使用 global 变量的函数或者脚本，都需要对其以 `global var1` 的形式进行声明；如果不声明，这个函数中的同名变量仍将被当做局部变量来使用。

另外，global 变量最好在函数或脚本的一开始就声明，否则会提示“如果不在函数开头声明，很可能在一些函数中不可用”。

### 4.3 递归函数

MATLAB 同样可以写递归函数，这里写了一个生成斐波那契数列的递归函数。

```
1 function result=MyFibonacci(n)
2 if n==1
3     result=1;
4 elseif n==2
5     result=1;
```



```
6 else
7     result=MyFibonacci(n-1)+MyFibonacci(n-2);
8 end
```

递归次数的默认上限是 500 次，可以使用 `set(0, 'RecursionLimit', N)` 来修改上限次数，但是需要注意的是，由于递归一次并没有结束函数，而是在函数中调用一个函数，内存没有释放，所以递归的次数不能太多以避免内存崩溃。可以思考一下如何用递归打印若干数的全排列。

## 4.4 提示信息

写函数不能指望每次调用的输入变量都符合我们的期望，而是需要想到所有可能的输入，对于不同的输入进行不同的操作，对错误的操作给出提示，以免程序卡死或者莫名的中断而不能定位错误位置。

提示信息的方式有很多。比如在本章开头时的例子中使用 `error` 函数来提示错误，也有使用 `disp` 函数。

`warnign` 和 `error` 的区别。从名字上看，前者提示警告，后者提示错误；使用格式上，两者的使用方法与 `fprintf` 相同；运行上，`warning` 不会提示完后继续运行程序，`error` 会终止程序；字体颜色上，`error` 的颜色要比 `warning` 更红更醒目。

`disp` 和 `fprintf` 的区别。为了能够了解我们程序运行到何种程度，我们需要在程序中加一些提示的语句，将运行状态打印到 Command Window 中。这是选择 `disp` 函数和 `fprintf` 函数都是可以的。两个函数也有区别：如果要打印某个矩阵，首选 `disp`，他会自动加上换行以及缩进；如果不换行打印，或者要提示和变量同时打印，则需要选择 `printf`。

`try/catch` 的使用。有些错误是我们不能预料的，那么我们就用 `try/catch`，其结构与 `if/else` 结构类似。`try` 一些操作，如果有异常则被捕获，进行 `catch` 中的操作。捕获的类型可以有 `err`, `exception` 等，在 `catch` 中常结合 `switch/case` 语句，根据每种类型的不同返回状态进行不同的操作。

## 4.5 内嵌函数与匿名函数

内嵌函数在其他语言中也有，用 `inline` 函数来表示，但是 `inline` 函数中的第一句话就是：“`inline will be removed in a future release. Use Anonymous Functions instead.`” 那这里就对它作介绍了，直接介绍匿名函数。

匿名函数只能用一条语句来表达简单的函数，不需要单独存成一个 `.m` 文件，而是存储在一个 `function_handle` 类型的变量里边。虽然是匿名，你也可以把这个变量名作为函数名，它跟标准函数一样，可以有输入和输出。`function_handle` 以 `@` 开头。

匿名函数可以有多个输入，也可以没有输入。

```
1 f1=@(x,y)x.^2+y.^2;      f1(2,3)
2 f2=@()disp(date);        f2()
```

函数中也可以使用变量作为系数。

```
1 a=1;b=2;
2 f3=@(x,y)a*x.^2+b*y.^2;    f3(2,3)
```

匿名函数中也可以嵌套匿名函数。

```
1 f4=@(c)(integral(@(x) (x.^2 + c*x + 1),0,1));
```

匿名函数的实现可以通过“变量名+括号”的形式实现，也可以使用 `feval` 函数实现。

另外，虽然不应该放在这个位置，但是还是顺便介绍分段函数和周期函数的写法吧。

```
1 f5 = @(t) (t>0)*2+(t<=0)*3;
2 f6 = @(t)mod(t,3.5)+1;
```

匿名函数可以使用 `fplot` 函数来画图，当然也可以通过 `plot` 或者其他函数画出来看看，如图 4.2。

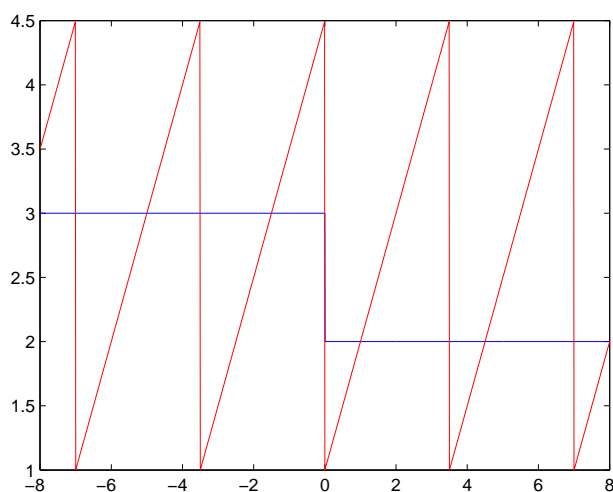


图 4.2: 分段函数与周期函数

## Chapter 5

# 数据类型定义与转换

### 5.1 数据类型

MATLAB 中到底有多少种类型的变量？我也不知道，其实有好多，其实也不多。

数值型数据的类型有 `double`, `single`, `int8`, `int16`, `int32`, `int64`, `uint8`, `uint16`, `uint32`, `uint64`, `logical` 等，这些既是数据类型，又是用户数据类型转换的函数。比如 `double(int32(pi))`。使用 `class` 函数可以查看某个变量的数值类型，比如 `class(int32(3))`。

字符型数据有 `char` 以及 `char` 组成的字符串（只是 `char` 型变量组成的矩阵）。`char` 和数值型的变量非常相似，除了显示之外，我们甚至可以把 `char` 当成一种特殊的数值。

结构体 `structure`，结构体中包含字段名（`field`）和与其对应的值（`value`）。定义结构体及赋值有两种方式，一种是通过 `struct` 函数，如 `student1 = ... struct('Name','PanZhongxian','Age',24);`；另一种方式是每次增加一部分，如 `student2.Name='Panzhongxian'; student2.Age=24;`。个人觉着前者结构清晰、可读性强、内存一次性分配完成，但是 MATLAB 帮助里边绝大多数都是使用后者。`struct` 型的变量也可以组成数组，比如下边是改自 MATLAB 中自带的一个例子。

```
1 patient.name = 'John Doe';
2 patient.billing = 127.00;
3 patient.test = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];
4 patient
5 patient(2).name = 'New Name';
6 patient(2)
```

元胞数据 `cell`，元胞数据只要一个花括号一括，就可以包含任何数据类型，其意义也正在于此。小明手里有三个数字，小红手里有四个数字，如何来表

示？用元胞数组吧。

```
1 Nums=cell(2,1);  
2 Nums(1)={2,3,4};  
3 Nums(2)={5,6,7,8};
```

C1和C(1)的区别。

C11的含义不是说 C 变量的第一行第一列的内容，而是说 C 第一个 cell 中的第一个 cell。

函数句柄 Function Handles，也就是我们前边所说的用来存放匿名函数的变量。

符号型变量 Symbol，MATLAB 中没有将它作为一种特殊的数据类型列出来，因为其本质上只是一种对象。

另外，MATLAB 中同样可以定义类（Class），对类的实现就得到了对象（Object）。因为实在是面向对象编程不了解，所以这里就不班门弄斧了。具体可以在帮助浏览器中搜索“Creating Object Arrays”。

MATLAB Help 中还介绍了两种单列出来的数据类型，说实话到我写这个总结之前，我都没有用到也没看过，希望后边能探索一下。这里只是提一下：

容器映射 containers.Map，有人说这是 MATLAB 中唯一的带一定逻辑性的数据结构。可以将一个量映射到另一个量，比如将一个字符串映射为一个数值，那个字符串就是 map 的 key，那个值就是 map 的 value。据说在组织大型数据时，使用起来非常方便。

时间序列 Time Series，其实这也是一个对象。时间序列是按时间顺序的取样。它们从形成许多其它数据进行分析的基础上，随机采样的数据区别开来。时间序列分析是关注识别模式、建模模式、预测数值，可以使用 tstool 来辅助分析。

## 5.2 类型判别

MATLAB 中自带若干判断性的函数格式为 is\*，用于判断数据类型。

判定数据类型是否是对象、元胞、字符串元胞、字符、字段名、浮点数、画图窗口句柄、整数、java 对象、逻辑性数、数值型数据、MATLAB 对象、实数、标量、字符串、结构体、向量，可以选用下列函数：isa, iscell, iscellstr, ischar, isfield, isfloat, ishghandle, isinteger, isjava, islogical, isnumeric, isobject, isreal, isscalar, isstr, isstruct, isvector 等。

除判断数据类型外，还可以判断是否是特定的应用程序定义的数据、COM 对象、文件路径、空、相等（可比较字符串、元胞等）、无穷大相等、COM 事件、有穷数、全局变量、句柄、hold on 状态、无穷大数、COM 接口、关键词、字母字符、MAC 系统、集合元素、方法、非数、Windows 系统、素数、对象属

性、已排序、空格字符、稀疏矩阵、字符串中的字母和数字、学生版本、UNIX 系统、变量名，可以选用以下函数：isappdata, iscom, isdir, isempty, isequal, isequaln, isevent, isfinite, isglobal, ishandle, ishold, isinf, isinterface, iskeyword, isletter, ismac, ismember, ismethod, isnan, isobject, ispc, isprime, isprop, issorted, isspace, issparse, isstrprop, isstudent, isunix, isvarname 等。

特别介绍一下 isequal，其实这个函数不仅仅判断两个数值或者矩阵 equal，还可以用来判断两个相同类型的数据是否完全一样。比如字符串、元胞、结构体、对象等。

```
1 syms a b
2 isequal(a+b,b+a)
3 isequal('Pan','pan')
4 isequal(rand(3),rand(3))
5 isequal(struct('a',1),struct('a',1))
```

另外，还有 is\* 之外的用于判断的函数，其它函数比如 any 用于判别是否有非零数、all 用来判别是否全为非零数。

### 5.3 格式转换

除了上述介绍的类型名加括号的形式进行数据类型转换外，还有其他一些格式的转换。

complex，可将两组相同尺寸的实数矩阵作为实部和虚部组合成复数矩阵；相反的函数 real, image 则是取出复数的实部和虚部。

num2str，将数字转换成字符串，可以类似于 fprintf 函数之类，方便的设置其输出属性；str2num，将字符串转换成数，这个函数非常方便的原因在于它可以将字符串中的几种分隔符自动的辨认出来，当然，分号表示换行。虽然很智能，但是也不是所有的输入都能转出数据来，比如每行的数据个数不一样或者有两个连续的逗号，返回就会为空，如果输出两个参数，第二个参数 status 为 0。

技巧性总结(石油价格分析部分的坐标轴中有补零操作)

str2func 和 func2str 函数用于字符串和函数句柄的转换，具体有什么用处呢？

num2cell，将一个数值型的矩阵一对一转换成 cell 型矩阵，size 是一样的。mat2cell，将数值型矩阵转换成一个 cell 矩阵，但是与 num2cell 的区别在于，他可以将矩阵进行切割，然后每个 cell 中装一个矩阵，而不是一个数。矩阵的大小可以不相同，通过后边的从参数进行控制。需要注意的是，后边的控制参数以向量形式给出了切割的行和列宽度而不是其实位置。这两个函数的逆函数是都是 cell2mat，因为 num2cell 产生的结果也就是列宽和行宽都为 1 的

mat2cell。

cell2mat 字符串转的时候应该注意什么???

```
1 a = magic(3)
2 b = num2cell(a)
3 c = mat2cell(a, [1,2], [2,1])
4 d = cell2mat(c)
```

oct2dec, dec2base, dec2bin 等函数进行不同进制之间的转换, bin, dec, hex, base 等, 发挥自己的想象力, 随意组合也能列出好多来。除了转换结果是 10 进制之外, 其他函数返回的结果都是字符串。

datenum 函数可以将各种形式的日期字符串转换成数, 公元 0 年 1 月 1 日对应为 1。当年的格式为两位数时, 一般需要加一个 PivotYear 的参数, 来控制其基数年份。其逆函数不是 numdate (因为根本没有这个函数), 可以用 datestr 函数来将数据转成固定格式的年月日。处理以天为采样单位的数据的日期 (比如股票), 可以用这两个函数非常方便的转换日期, 以便分析。

```
1 DateString = {'09/16/2007'; '05/14/1996'; '11/29/2010'};
2 formatIn = 'mm/dd/yyyy';
3 datenum(DateString, formatIn)
4 datestr(ans, 'mm/dd/yyyy')
```

struct2cell 将结构体中的 Value 值作为一个 cell, 最后组成一个 cell 数组, “Field” 就被舍掉了。cell2struct 将一个对应 Value 的 cell 数组转换成一个结构体, 由于缺少 Field, 所以我们需要在后边添加上 Field 组成的 cell。

```
1 patient.name = 'John Doe';
2 patient.billing = 127.00;
3 patient.test = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];
4 p1=struct2cell(patient);
5 p2=cell2struct(p1, {'name', 'billing', 'test'});
6 isequal(p2, patient)
```

sym 函数将 char 或者数值型变量转换成符号型变量。lower, upper 用来转换字母的大小写。blanks, deblank 函数分别用来产生和消去字符串中的空格。

# Chapter 6

## 画图

仿真结果需要用清晰的图展示出来。要画一个漂亮的图，需要注意很多细节，这一章就谈谈一些对画图的设置。

### 6.1 General

下面列出一些画图通用的命令：

<code>figure</code>	创建画图窗口
<code>axes</code>	创建画图坐标
<code>set</code>	设置图像某些属性
<code>get</code>	得到图像某些属性
<code>gcf</code>	得到当前 figure 的 handle
<code>gca</code>	得到当前 axes 的 handle
<code>clf</code>	擦除当前 figure 的内容
<code>cla</code>	擦除当前 axes 的内容
<code>axis</code>	控制坐标轴范围
<code>plot</code>	2 维线图
<code>hold</code>	一个 axes 中允许/禁止连续画图

在这里先说一下 figure 和 axes 的关系：figure 是一个窗口，而 axes 是一个坐标系，axis 是坐标轴。一个 figure 中可以包含多个 axes，一个 axes 中有多个 axis。当创建一个 axes，会自动将其贴到一个 figure 中，如果没有 figure 则创建一个 figure。

每进行一次画图的操作，都会得到一个句柄，通过操作这些句柄可以对图像进行属性的修改。想知道某个图像元素有哪些属性，可以使用 `get(h)` 获得其所有属性，再用 `set(H, 'PropertyName', PropertyValue)` 来设置其中的属性。

## 6.2 figure 的属性

之前已经介绍过 figure 和 axes 的关系，这一小节主要介绍一下 figure 一些常用属性的设置。figure 有很多属性，但是其中有一部分是用来做交互的，比如 CloseRequestFcn 属性是用来控制关闭 figure 时调用函数；有一部分属性是用来控制在 Windows 和 Unix 下显示方式的；还有一部分是控制打印格式的。所以我们常用的属性只有很少的几个。这里写了段小程序，设置了几个属性，大家可以运行一下。

```
1 scrsz = get(0, 'ScreenSize');
2 f=figure;
3 set(f, 'Position', [scrsz(3)/4 scrsz(4)/5 scrsz(3)/2 scrsz(4)/2], ...
4     'Color', 'w', ...
5     'Name', 'Pan's Figure', ...
6     'MenuBar', 'none', ...
7     'ToolBar', 'none', ...
8     'NumberTitle', 'off', ...
9     'Resize', 'off');
```

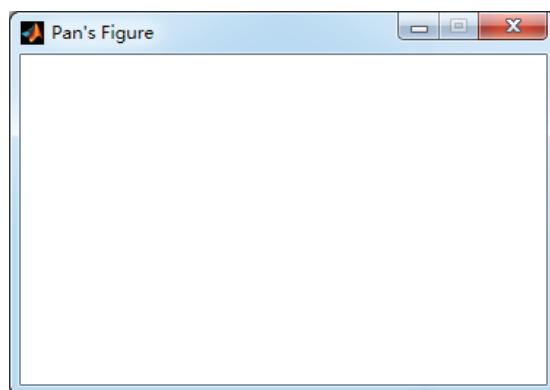


图 6.1: 创建一个 figure

figure 的一些响应函数在 GUI 中使用较多，在相关章节还会再介绍。

## 6.3 axes 属性

之前已经介绍过 figure 和 axes 的区别，这一小节主要介绍一下 axes 一些常用属性的设置。其中有一些属性不属于 axes，但为了比较或者逻辑连续性，将这些属性放到了本节。

如果有更多的需求或者要了解更多的属性，请到 MATLAB Help 中搜索 Axes Properties。



### 6.3.1 大小和位置

画一个图，首先要确定其大小和位置。当然，如果我们没有特殊的要求，将会使用 MATLAB 的默认设置。这是介绍一下三个范围：OuterPosition、Position 以及 TightInset（只读属性）。

如图 6.2，该图片来自 MATLAB Help。

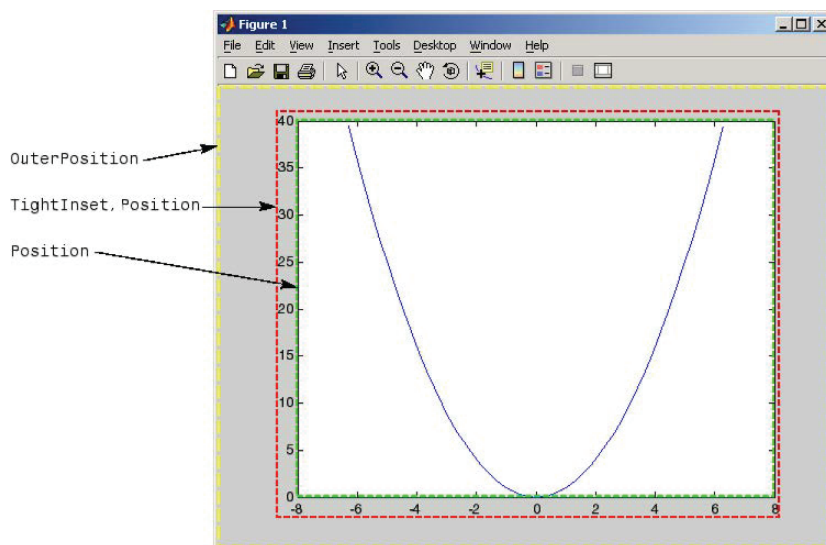


图 6.2: 图像的大小和位置

OuterPosition 和 Position 两个属性只能通过设置一个来控制 Axes 的大小，当设置一个属性时，默认该属性为 ActivePositionProperty。

### 6.3.2 坐标轴属性

本小节以 x 轴为例，讨论坐标轴（Axis）的一些属性设置。坐标轴作为坐标系的一部分，要控制坐标轴实际是控制坐标系的属性。

属性	作用
XScale	控制坐标轴尺寸是线性还是对数
XTick	用于标定 X 轴 Tick 的位置
XLim	X 轴显示的范围
XGrid	X 轴 Tick 对应位置加栅格线
XDir	X 轴的方向
XColor	X 轴的颜色
XAxisLocation	X 轴的位置是底部还是顶部

XTickLabel	X 轴 Tick 对应的字符串, 若不设置为对应的数值
Box	控制边框的开关
XLabel	用来获取 Axes 中 XLabel 的句柄, 非属性
TickDir	Tick 是在内侧还是外侧
TickLength	Tick 的长度

下面通过程序展示对 X 轴属性的设置。

```

1  t=0:11;
2  y=10.^t;
3  Months={'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',...
4         'Sep', 'Oct', 'Nov', 'Dec'};
5  plot(t,y,'.-');
6  AX=gca;
7
8  set(AX,'YScale','log');
9  set(AX,'XTick',[0:11]);
10 set(AX,'XTickLabel',Months);
11 set(AX,'XLim',[-0.5,11.5]);
12 set(AX,'XDir','reverse');
13 set(AX,'XGrid','on');
14 set(AX,'XColor','red');
15 % set(gca,'YAxisLocation','Right');
16 set(get(AX,'XLabel'),'String','Month','Color','k');
17 set(AX,'TickDir','out');
18 set(AX,'TickLength',[0.005,0.005])
19 set(AX,'Box','off');

```

执行程序可以得到图 6.3, 我们可以将每一个属性的设置, 对应到图中。比如坐标轴颜色的设置、对数间隔的设置。

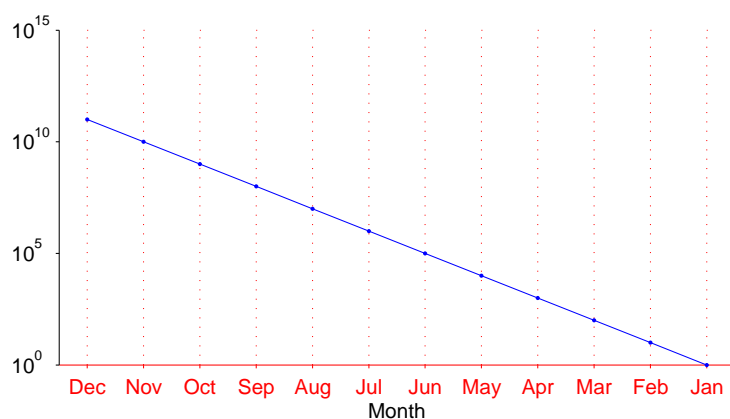


图 6.3: 坐标轴属性控制

特别的提到 Xlabel，是想说 Xlabel 不是 Axes 的属性，而是 Axes 中的一个句柄，指向 X 轴下标注的文字。当要修改 x 轴标注时，应修改 Xlabel 句柄的对应属性。而 xlabel() 则是用来直接加标注并设置属性的函数。

而其他的很多属性也可以直接用命令来控制，比如 set(AX, 'XLim', ... [-0.5, 11.5]) 和 xlim([-0.5, 11.5]) 是等价的。

### 6.3.3 plot 里边的点线属性

图上的点和线是数据的直观表现，这一小节主要说说 plot 函数中的点和线属性的设置。

属性	作用
LineStyle	线型，可选 - , -- , : , - . , none
LineWidth	线条宽度
Color	线条颜色
Marker	标记类型，可选 + , o , . , x , s 等
MarkerSize	标记符号大小
MarkerEdgeColor	标记符号边框颜色
MarkerFaceColor	标记符号面颜色

以上各个属性为最常用的属性，还可以通过 get 函数查看其他属性，再进行相应的设置。

通过一段程序来展示一下对这些属性的设置。下边的代码是对 MATLAB Help 中代码进行的改写。

另外，这里将栅格线（GridLines）的添加和线型这只也放在了这里，其设置与对 plot 类似。

```

1 figure;
2 t = (0:20)/20;
3 y = sin(2*pi*t)
4 h1 = plot(t,y);
5
6 set(h1, 'LineStyle', '-.', ...
7       'LineWidth', 2, ...
8       'Color', 'r', ...
9       'Marker', 'o', ...
10      'MarkerEdgeColor', 'k', ...
11      'MarkerFaceColor', [.49 1 .63], ...
12      'MarkerSize', 10)
13
14 grid on;
15 set(gca, 'GridLineStyle', '-');
16 set(gca, 'Layer', 'top')

```

执行代码，得到图 6.4。

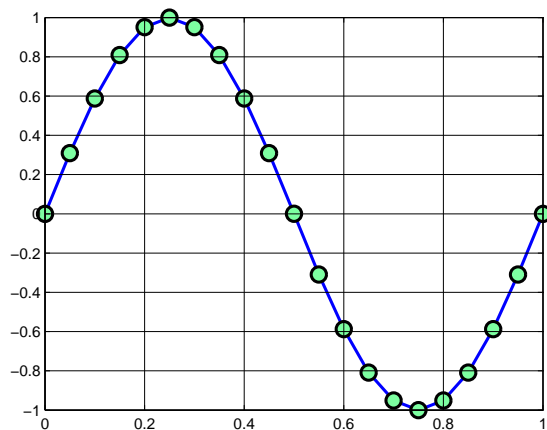


图 6.4: 线点属性设置

### 6.3.4 双 y 轴

绘制左右两侧都有 y 轴的图像时需要用到 `plotyy` 函数。虽然很多人都知道这个命令，但是想画出一副自己满意的双 y 轴图像还是需要设置好多属性的。这里以一个任务讲解需要设置的属性。任务：画一幅四线同图的图线，其中两根根在一个坐标轴重，另外两根在另外另一坐标，两坐标轴 y 轴分别位于左右两侧，在同一个 axis 中上下分布。

```

1 clear;close all;clc;
2 color_set={'c','b','m','r'};
3 t=0:0.1:10;
4 yu1=sin(t);
5 yu2=sin(t+pi*0.1);
6 yd1=2*cos(t);
7 yd2=2*cos(t+pi*0.1);
8
9 figure;
10 [AX,H(1),H(3)]=plotyy(t,yu1,t,yd1);
11 % [AX,H(1),H(3)]=plotyy(t,yu1,t,yd1,@plot,@semilogy);
12 hold(AX(1),'on'); hold(AX(2),'on');
13 set(AX(1),'Box','off');
14 set(AX(2),'Box','off');
15 % RATHER THAN 'hold on;'
16 H(2)=plot(AX(1),t,yu2);
17 H(4)=plot(AX(2),t,yd2);
18

```

```

19 for index=1:4
20     set(H(index), 'Color', color_set{index});
21 end
22
23 linkaxes(AX, 'x');
24
25 set(AX(1), 'XLim', [0,10], 'XTick', 0:1:10);
26 set(AX(2), 'XAxisLocation', 'Top')
27 set(AX(2), 'XTick', []);
28
29 set(AX(1), 'YColor', 'b', 'YLim', [-4,1.5], 'YTick', -1:0.5:1);
30 set(AX(2), 'YColor', 'r', 'YLim', [-2.5,7], 'YTick', -2:1:2);
31 set(AX, 'XGrid', 'on', 'YGrid', 'on');
32
33 legend(H, {'yu1', 'yu2', 'yd1', 'yd2'});
34 set(get(AX(1), 'Ylabel'), 'String', 'yu') ;
35 set(get(AX(2), 'Ylabel'), 'String', 'yd') ;
36 xlabel({'t'});

```

运行程序，得到的6.5，还算比较漂亮的一幅图。

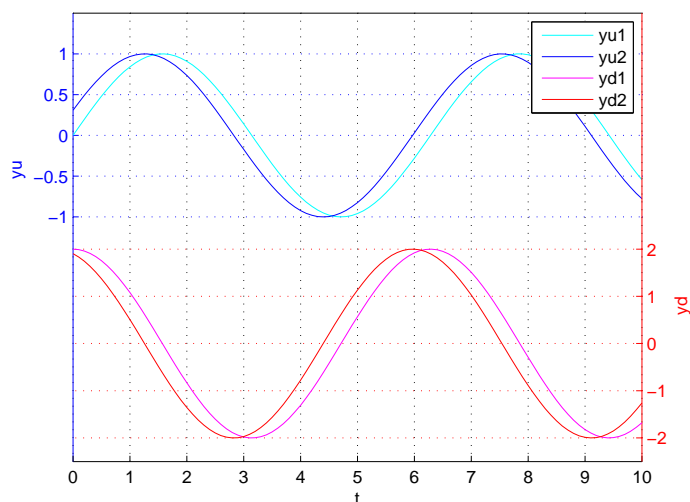


图 6.5: 双 y 轴作图

需要说明以下几个问题：

1. `[AX, H1, H2] = plotyy(...)` 函数返回三个参数，AX 是一个包含 AX(1) 和 AX(2) 两个 axes 的句柄，H1 和 H2 是两条曲线的句柄。
2. 可能有人会问，这样的图好像不如画成两个 subplots，然后 linkaxes 的效果好。我这里把两个图上下分开画主要是为了举例说明，当需要控制左右坐标轴范围时该如何设置。

3. 在第 11 行程序中的注释表示左右两个坐标轴可以是 `plot` 和 `semilogy` 型, 当然 `x` 轴也可以是对数型的。

4. 这里任务特别的强调要画四条线, 逻辑就是就是需要在产生的 `axes` 上继续画图。有好多使用类似于 `plot(ax(2), x, y)` 命令时会遇到如下的错误 “??? Error using ==> plot Parent destroyed during line creation”, 原因就是 `hold on` 命令只针对 `ax(1)`, 如果需要在 `ax(2)` 上添加, 需要对其 `hold on`。

5. 程序 13 和 14 行将两个坐标系的 `box` 取消掉, 是为了防止 `AX(1)` 两侧的颜色和 `AX(2)` 两侧的坐标轴颜色重叠。程序 26 行将 `AX(2)` 的 `x` 轴放到顶部, 第 27 行将 `AX(2)` 横轴显示置为空, 通过这种方式又构成了一个 `Box`, 将 `Axes` 给围了起来。另外, 若两个 `axes` 的两个横轴都设置在底部且显示, 则会出现重叠变粗的现象。

## 6.4 颜色

颜色在下边的哪一小节中都会用到, 随便插到哪里都不太合适, 所以就单独拿出一节介绍一下。

### 6.4.1 颜色的表示

可以用三种方式表示一个 RGB 色, 分别是短名、长名、RGB 分量值。使用长、短名的颜色较为有限, 只有 8 种, 其三种表示的对应关系如下:

RGB Value	Short Name	Long Name
[1 1 0]	y	yellow
[1 0 1]	m	magenta
[0 1 1]	c	cyan
[1 0 0]	r	red
[0 1 0]	g	green
[0 0 1]	b	blue
[1 1 1]	w	white
[0 0 0]	k	black

`plot(1:5, 'Color', [0 0 1]); plot(1:5, 'b'); plot(1:5, 'blue');` 这三条命令是等价的。

如果要使用其他颜色, 可以自己根据 RGB 的分量进行调整, R、G、B 值取值范围从 0 到 1。当然如果有自己常用的颜色, 可以通过 `colordef` 函数来定义一种新的对应关系。

### 6.4.2 colormap 与 colorbar

colormap 函数用在一些自动出现多种颜色的画图之后，用来控制图的颜色风格，比如 bar, mesh, surf, contour 等函数。

```
1 load count.dat;
2 yMat = count(1:6,:);
3 figure;
4 bar(yMat);
5 colormap('summer');
6 figure;
7 bar(yMat);
8 colormap('autumn');
```

执行程序，得到的结果如图 6.6 中的两幅图，数据完全一样，只有用以标识种类的颜色色系不同。

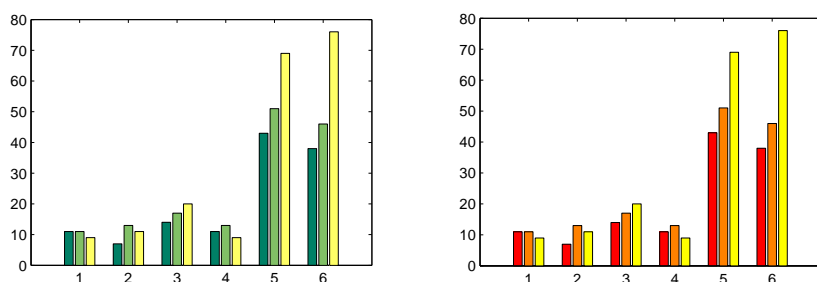


图 6.6: Colormap 示意图

有一些画图的函数用颜色表示数值大小，比如 image(), imagesc(), surf(), surfc, mesh(), meshc(), trisurf() 等。而 colorbar 函数是之后，用来显示不同颜色与数值大小的对应关系。colorbar 其实也是一个 axes，它的若干属性与坐标系的属性是类似的，但是其中 location 属性是和 legend 函数属性是类似。

```
1 close all;
2 figure;
3 n=20;
4 [X,Y] = meshgrid(linspace(0,1,2*n+1));
5 L = 40*membrane(1,n);
6 surfc(X,Y,L);
7 colorbar;
8
9 figure;
10 contourf(rand(10));
11 set(gca, 'XTick', [], 'YTick', []);
12 hbar=colorbar('location','SouthOutside');
```

```

13 set(hbar,'XTick',...
14     linspace(min(get(hbar,'Xlim')),max(get(hbar,'Xlim')),7),...
15     'XTickLabel',{'Freezing','Cold','Cool','Neutral','Warm',...
16     'Hot','Burning'});

```

上述程序得到的结果如图 6.7。

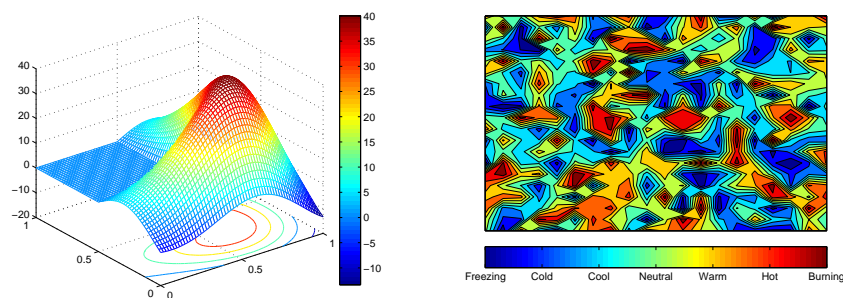


图 6.7: Colorbar 示意图

### 6.4.3 颜色填充

介绍三个用来填充颜色的函数：`patch(X,Y,C)`, `fill(X,Y,C)`, `area(Y)`。三个函数的区别在于，`patch` 是一个低层次的函数，`fill` 和 `area` 是高层次的函数，后两个函数可能用到 `patch` 函数。`fill` 的功能是给多边形填充颜色，而 `area` 的功能是将某条线与基准线之间的部分填充上颜色。

## 6.5 漂亮的注释

### 6.5.1 text, title, xlabel, legend

`text` 用来在图中加文字的，这些文字可以包含 MATLAB 自带的特殊符号，比如  $\alpha\beta \rightarrow$  等。

`text` 函数有多种输入参数的方式，最常用的是 `text(x,y,'string')`，其中 `x,y` 表示 `text` 的左侧的坐标。

常用的 `text` 属性有：

属性	控制对象
HorizontalAlignment	水平对齐方式
VerticalAlignment	垂直对齐方式
FontSize	字体大小



FontAngle	字体倾斜度
FontName	字体名称
FontWeight	字体粗细
Color	字体颜色
BackgroundColor	背景颜色
Margin	文字周围空白大小
EdgeColor	边框颜色
LineStyle	边框线性
LineWidth	边框宽度

通过一段代码来展示这些属性的控制作用。

```

1 figure;
2 plot(1:5)
3 h=text(3,4,{'\color{magenta}magenta} {\bf{text}} {\it{here}}!',...
4   '\color{black}a new line');
5 set(h,'HorizontalAlignment','center',...
6   'VerticalAlignment','middle',...
7   'FontSize',16,...
8   'BackgroundColor',[.7 .9 .7],...
9   'EdgeColor','k',...
10  'LineStyle','—',...
11  'LineWidth',1,...
12  'Margin',10,...
13  'Color','b');

```

运行代码，得到的图 6.8

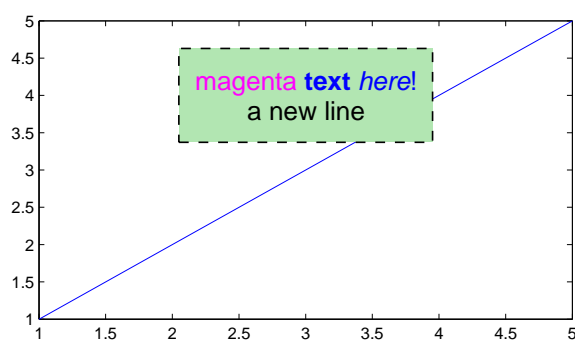


图 6.8: text函数的应用

另外需要说明几点:

1. 修改的属性是通用的属性，对于一些在 text 函数中特别写入的属性，后边是无法改变的。

2. 如果用左箭头指向某点, 则text 位置选成该点, 选择左侧对齐方式, 同理右箭头应选择右对齐的方式。当然也可以通过画好后人工拖动的方式。

3. 在text的字符串函数中使用转义字符时, 转义范围可以用大括号括起来, 大括号之外的文字不再受此转义字符的影响。

4. 如果要多行显示, 可以使用大括号产生包含字符串的CELL 向量。跟 3 中不同的是这个大括号应该放在字符串外边。

title 和 xlabel 其实是特殊的 text , 只是固定在对应的位置, 方便使用。属性基本与 text 相同, 大家可以通过 Help 自己探索。

图例 legend 也算是一种固定位置的 text, 有 Location 属性, 来控制其位置(包括 axes 内外的各个方位), 其文字的属性与text 也基本相同。不管有几个图例, legend 只能用一列; 第一次之后的图例会被忽略, 不起任何作用。

### 6.5.2 漂亮的 L<sup>A</sup>T<sub>E</sub>X 表达式

MATLAB自身使用的表达式比较有限, 想要用更漂亮的表达式进行注释, 就选择 L<sup>A</sup>T<sub>E</sub>X 吧。

latex() 命令自动将MATLAB 表达式转换成 L<sup>A</sup>T<sub>E</sub>X 表达式。

```
1 figure;
2 f1 = '\sum_{i=1}^n i \prod_{i=1}^n \lim_{x \to 0} x^2 \int_{a_i}^b ...
      x^2 dx';
3 f2 = latex(sym('x^4+3*x^2-1'));
4 set(gca, 'OuterPosition', [0.05, 0.07, 0.9, 0.86]);
5 text(.5, .5, ['$$_{f1}$'], 'interpreter', 'latex', 'FontSize', 18, ...
      'HorizontalAlignment', 'center');
6 xlabel('$\widetilde{xxx}$', 'interpreter', 'latex', 'FontSize', 16);
```

执行代码, 效果如图 6.9。

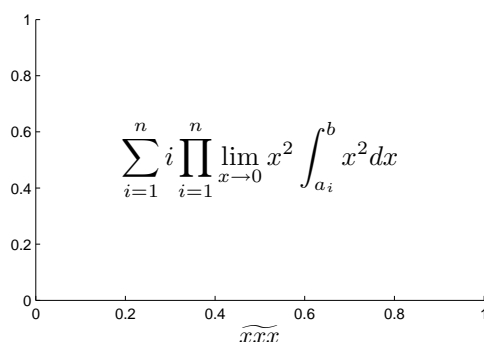


图 6.9: 注释中输入 Latex 表达式

### 6.5.3 text 与 annotation

这两个注释函数的都是为了给图像加注释的，但还是有不少区别的。

`text` 的 `Position` 是以 `axes` 为基准的，默认选择 `gca` 的句柄；输入的 `x,y,z` 坐标为一个点，是 `text` 的起点。

`annotation` 的 `Position` 是以 `figure` 为基准的，默认选择 `gcf` 的句柄；输入的 `x,y,z` 坐标的为两个点，分别是注释连接线的起始点；其支持的线、箭头、矩形、椭圆等注释方式。

下面用程序来展示一下两个函数的不同之处。

```
1 figure;
2 t=1:5;
3 plot(t,t,t,t+1,t,t+2)
4 annotation('ellipse',[.462 .297 .053 .262])
5 x = [0.5585 0.5093];
6 y = [0.2366 0.3261];
7 annotation('textarrow',x,y,...
8           'String','The first 2 lines.','FontSize',14);
9 text(1.75,5,'The third line.\rightarrow','FontSize',14);
```

执行代码，结果如图 6.10

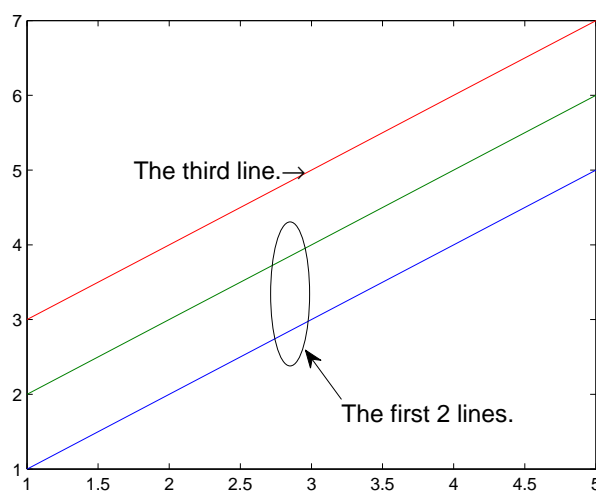


图 6.10: text 和 annotation 的功能

## 6.6 创建子图

仿真图中往往需要一个大图中包含多个子图，这就需要用到 `subplot` 函数

### 6.6.1 以编号定义子图

子图的编号顺序是按照行排序的。

1	2	3
4	5	6
7	8	9

另外，有时需要将不同大小的子图放到同一副图中，就需要将其中的若干子图进行拼接。拼接时按照最外端的子图展成的最大矩形子图，效果如图 6.11。

```
1 figure;
2 subplot(3,3,1);          subplot(3,3,[2 3]);
3 subplot(3,3,[4 7]);      subplot(3,3,[5 9]);
```

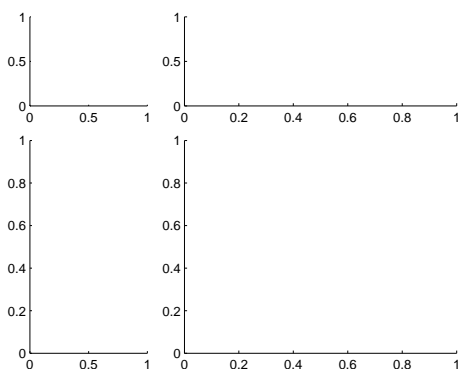


图 6.11: 以编号定义子图

### 6.6.2 以位置定义子图

除了合并子图的方法来改变各个子图大小的方法外，还可以通过确切的位置。

```
1 figure;
2 subplot('position',[.08,.08,.35,.8]);      title('subplot 1');
3 subplot('position',[.6,.08,.35,.8]);      title('subplot 2');
```

### 6.6.3 子标题与总标题

如上一小节所示，子标题可以直接用 `title` 函数写在对应的子图之下即可。总标题可以通过 `annotation` 函数作为注释加入到其中。根据标题长度大小控制 `textbox` 的位置。

接上一小节例子, 通过 `annotation` 命令即可加入总标题, 效果如图6.12。

```
1 annotation(gcf, 'textbox', 'String', {'title'}, 'FontSize', 20, ...
   'Position', [0.47 0.88 0.1 0.1], 'edgecolor', get(gcf, 'color'))
```

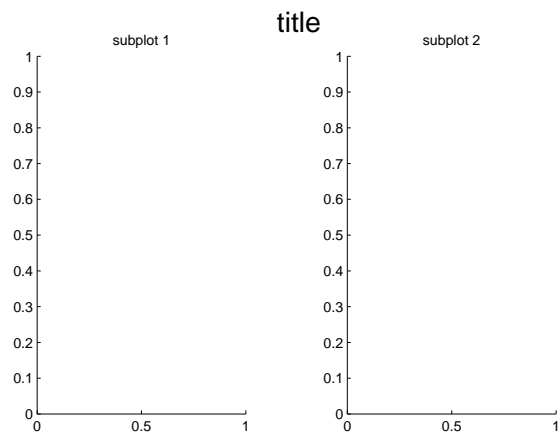


图 6.12: subplot 加总标题

#### 6.6.4 链接坐标

当两个一维信号或者两幅图像有某些联系时, 往往需要对坐标轴进行连接, 以方便分析, 这里举两个例子。

##### 只链接 x 轴

```
1 figure;
2 t = 0:0.1:20;
3 ax(1) = subplot(2,1,1);
4 plot(t,sin(t));
5 ax(2) = subplot(2,1,2);
6 plot(t,cos(t));
7 linkaxes(ax, 'x');
```

##### 同时链接 x,y 两坐标轴

```
1 I = imread('moon.tif');
2 h = fspecial('unsharp');
3 I2 = imfilter(I,h);
4 figure;
5 ax(1) = subplot(1,2,1);
6 imshow(I), title('Original Image');
7 ax(2) = subplot(1,2,2);
8 imshow(I2), title('Filtered Image');
9 linkaxes(ax, 'xy');
```

这时候使用放大选项，就可以在一副图中选择放大区域，而两个或者多个坐标系的相同区域都会被同样的放大。

## 6.7 其他画图函数整理（待整理）

area	填充2D区域
bar/bar3	条形图/3D条形图
bar3h	3D水平条形图
barh	水平条形图
pareto	排列图
pie/pie3	饼状图/3D饼状图
comet	彗星轨迹图
comet3	3D彗星轨迹图
compass	罗盘图
feather	速度向量图
quiver	2D方向箭头图
quiver3	3D方向箭头图
stairs	梯级图
stem	2D离散序列图
stem3	3D离散序列图
hist/histc	直方图
histc	直方图
rose	角度直方图
cylinder	柱面图
delaunay	德洛涅三角剖分
delaunay3	3D三角化图
ellipsoid	椭球图
inpolygon	取多边形区域内点
pcolor	伪彩色（棋盘）图
polyarea	多边形图
ribbon	带状图
slice	切片图
sphere	球形图
waterfall	瀑布图
plotmatrix	散点图矩阵
scatter	散点图
scatter3	3D散点图

### 6.7.1 其他相关函数

CameraPosition

image imagesc

# Chapter 7

## 文件操作

### 7.1 文件与路径

文件与路径常用的函数列举如下：

路径查询	dir/ls	列出文件夹中内容
	pwd	当前文件路径
	isdir	是否是路径
	.	当前文件路径
	..	父文件路径
路径修改	cd	改变文件路径
	copyfile	复制文件（夹）
	movefile	移动文件（夹）
	mkdir	创建文件夹
	rmdir	删除文件夹
	delete	删除文件/对象
	recycle	设置回收文件夹
文件查询	fullfile	填充文件路径
	what	列出所有 MATLAB 文件
	who	列出所有 WorkSpace 中的变量
	which	列出函数或文件的位置
文件操作	exist	判断文件等的存在性
	type	打印文件内容
	fileattrib	列出文件（夹）属性
	open, winopen	打开文件（夹）

exist 函数，该函数能够判断存在性，只要输入一个字符串即可。返回值为 1-8 时，分别表示（1）存在变量、（2）m 文件或 MATLAB 路径、（3）mex



或 dll 文件、(4) Simulink 模型或库文件、(5) 内嵌函数、(6) P 文件、(7) 文件夹、(8) 类；当返回值为 0 时，表示上述的所有都不存在。exist 函数也可以加上选项，判断是否存在指定的类型。

dir 函数在文件的批处理时是非常有用的。dir 中可以使用通配符，星号表示若干任意字符，问号表示单个任意字符。如果不加限制时，会有 . 和 ...，处理时应注意。返回的结果为结构体或者结构体向量，结构体具体如下：

```
name: 'PanZhx1.fig'
date: '21-十一月-2013 10:51:00'
bytes: 6011
isdir: 0
datenum: 7.3556e+05
```

利用 dir 列出的信息，我们能够对文件进行批量的操作。通过下边的操作将前边的几条命令综合起来解释一下。

```
1 newpath='subfolder';
2 if exist(newpath,'dir')==7
3     rmdir(newpath,'s');
4 end
5 mkdir(newpath);
6 filelist=dir('Pan*.txt');
7 num=0;
8 for index=1:length(filelist)
9     if filelist.datenum > datenum(date)-10
10         num=num+1;
11         newname=[num2str(num,'%03d'),'*.txt'];
12         copyfile(filelist.name,[newpath '\ ' newname]);
13     end
14 end
```

第 2 行的使用 exist 时，加了一个 ‘dir’ 选项，是为了只查找指定的文件夹；第 3 行的使用带 ‘s’ 选项的 rmdir 函数，是为了在不管权限而删除文件夹以及文件夹中的内容；第 6 行使用了通配符，列出所有以 ‘Pan’ 开头的 txt 文件；第 9 行用 datenum 作为操作依据，对 10 天之内的文件进行操作；第 11 行使用带格式的 num2str 函数，对 num 变量前补 0 补够 3 位，作为文件名；第 12 行使用更改文件名的 copyfile 函数。

另外读取文件夹中有一定规律的文件，比如从 001.txt 到 099.txt 之类的，我们也可以通过改写上边的脚本中的函数再调整一下顺序皆可实现。

MATLAB 还自带代码比较功能，visdiff 可以比较两个文件或文件夹。虽然这里的文件不光指 MATLAB 文件，但是我们最常用的还是用它进行比较代码功能。图 7.1 中展示了两个文件的比较。

比较窗口中首先给出的是文件名和修改时间等文件信息，正文部分的代码比较根据不同的相似程度，用颜色进行了标注。其中，完全相同的行用白色的

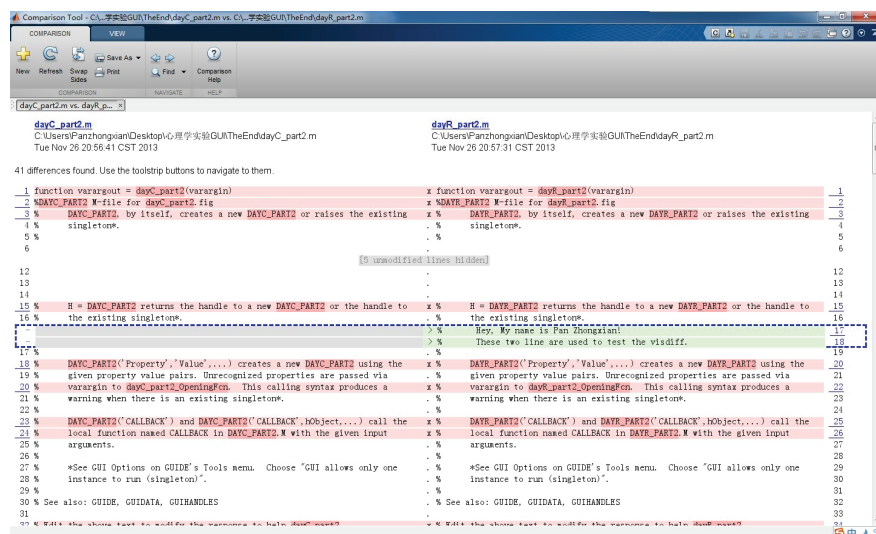


图 7.1: Comparison Tool 的使用

底，大部分相同用浅红色的底（其中不同的部分用深红色标注），多的插入行用绿色标注，对应少的行则用灰色标注，空行会隐藏并用灰色提示“未更改的行已隐藏”。这一工具也能在 Editor → FILE → Compare 中找到。

copyfile 如果不加destination，就默认复制到当前工作目录

## 7.2 低层次文件读写

MATLAB 能读入很多文件类型，除了 MATLAB 自己的数据文件类型 MAT 文件外，还可以读入 bin 文件、ASCII 文件、csv 文件、xls 文件、图像文件、视频文件、音频文件等。

首先说一下低层次的文件操作函数：fopen、fclose、fseek、ferror、fgetl、fgets、fprintf、fread、frewind、fscanf、feof、ftell、fwrite 等，不是说使用这些函数的就档次低，而是说这些函数操作层次比较低，其他的一些读写操作函数往往在内部调用这些低层次的函数。这些函数的使用和 C 语言中的同名函数几乎是相同的，不同之处在于一些细微的调用格式。对文件进行操作之前都需要使用 fopen 函数进行打开，对文件句柄进行操作完后，又需要使用 fclose 函数将文件关闭。这里就不对这些函数作具体的介绍了，下边的应用列举了对二进制文件的读写。

```
1 fid = fopen('magic5.bin', 'w');
2 fwrite(fid, magic(5));
3 fclose(fid);
4
5 fid = fopen('magic5.bin');
```

```
6 m5 = fread(fid, [5, 5], '*uint8');
7 fclose(fid);
```

`fopen(filename, 'property')` 函数中的 `property` 用来控制以何种方式打开文件，比如只读、读写等。

## 7.3 dlmread 和 dlmwrite 函数

对于一些格式简单的文件，我们可以使用 `dlmread` 函数进行读取，`dlm` 其实是 ASCII-delimited 的缩写。这里指的格式简单，一方面是指给文件格式，复杂如 `.doc` 文件；另一方面是指文件内容的格式，复杂如 “3:0,3;4”。

```
1 a=magic(4);
2 dlmwrite('Panzhx.txt',a);
3 b=dlmread('Panzhx.txt','')
4 c=dlmread('Panzhx.txt',' ',1,1)
5 d=dlmread('Panzhx.txt',' ','A1..C2')
```

`dlmread` 函数可以不读取整个文件，而是只读取一部分。如果文件内容没有对齐，则是按照最大的行和列产生矩阵，空余位置补 0。

上述第 4 行程序使用其实位置的行列偏移，读取从第 (R,C) 个数据开始的右下方的所有数据，但是这里的行列值都是从 0 开始的，也就是说，想要得到和 `dlmread('Panzhx.txt')` 得到相同的结果，需要设置其实的行列为 (0,0)。

第 5 行程序是指定一个读取范围，行和列分别用字母和数字表示，组合表示文件中元素的位置。但与 (R,C) 方式不同的是，第一行使用 1 表示的。

```
1 If=20.03mA Vf=3.216V Ir=0μ.00A
2 Luminous Intensity Diagram (unit:mcd) (CIE A)
3
4 Garma      C0-180    C30-210    C60-240    C90-270    C120-300   C150-330
5 -90.0      14.96     23.60     25.85     27.11     27.36     26.61
6 -89.8      15.40     23.91     26.23     27.48     27.80     26.92
```

**技巧：**上边列出的数据文件开头是若干数据说明，但是数据说明的格式固定。这时候直接使用 `dlmread` 函数读取就会报错，而使用设置起始行列 (R,C) 的方式则非常方便。如下的文件，我们可以使用 `c = ... dlmread('Panzhx.txt', '', 4, 0)` 直接得到其中的数据。使用 `' '` 是为了让其忽略多余空格。需要注意的就是分隔符一定要统一，尽量不适用上边的这种形式，否则读取时很容易出现行列对不齐的情况。

与之对应的写函数是 `dlmwrite`，可以带几个选项，控制分隔符、行列偏移、输出格式等。如果使用 `dlmwrite` 函数而不加选项会进行文件的覆盖写入，而我们常常用到的是追加功能，追加的选项是 `'-append'`。

```

1 dlmwrite('myfile.txt', magic(5), ' ')
2 dlmwrite('myfile.txt', rand(3), '-append', ...
3         'roffset', 1, 'coffset', 1, 'delimiter', ' ', ...
4         'precision', '%10.4f')

```

其中，行列偏移也是基于 0 的。输出格式跟之前介绍的 `num2str` 等函数的格式是一样的。

类似的一组函数 `csvread` 和 `csvwrite` 函数，csv 是 comma-separated value file 的缩写，所以这组函数只能读写逗号分割的文件。可以加偏移和范围，不同之处在于若要加范围首先需要加上偏移位置。另外 `csvwrite` 函数不能够控制输出格式。

## 7.4 textscan

一个复杂的文件，不可能通过一个通用的函数就读入，然后把你想要的变量分好类，但是通过通用的函数，设置复杂的选项倒是可能。比如 `textscan` 函数就非常给力，它应用于一个文件里边有包含许多种不同类型的数据信息，而之前介绍的 `csvread` 和 `dlmread` 函数常用于读取单一类型（尤其是数值矩阵）的文件。

以对下面的复杂文件进行操作对 `textscan` 进行说明。

```

1 ##### Copyright at Pan Zhongxian #####
2 01/10/2005 PAN Level1 12.34 1.23e10 inf Nan Yes 5.1+3i nothing
3 03/11/2005 PZX Level2 23.54 9e19 -inf 0.001 No 2.2-.5i haha
4 // comment line
5 11/12/2005 Pan Level3 34.90 2e5 10 100 No 3.1+.1i panzhx

```

处理文件的 MATLAB 代码如下：

```

1 fileID = fopen('scan1.dat');
2 C = textscan(fileID, ...
3             '%*3c %7s %*s Level%d8 %f32 %u %7.1f %*f %s %f %*[\n]', ...
4             'commentStyle', '//', ...
5             'HeaderLines', 1, ...
6             'MultipleDelimsAsOne', 1);
7 fclose(fileID);C

```

`textscan` 函数需要对 `fopen` 的函数句柄进行操作，所以 M 脚本第 1 行就是将文件打开并得到文件句柄。

第 3 行需要根据文件内容的格式设定读取的格式：'%' 用来标识格式，'\*' 表示忽略掉或者不输出；'%\*3c' 表示对前三个字符忽略掉，c 也可以换成 s；'%7s' 表示对读取七个字符的字符串；'%\*s' 表示忽略掉该字符串，不做输出；'Level%d8' 表示将 Level 字符之后的数字转成 d8 格式，也就是 int8 类型；'%f32' 表示将数转成 32 位浮点数，即单精度 single 型；'%u' 表示将数值转成

uint32 类型; '%7.1f' 表示将数转成 double 类型, 位宽 7 位, 小数点后精度 1 位; '\*[^\n]' 表示忽略掉本行剩余的信息, '[^...]' 表示一直读到中括号内的字符为止, 此处也就是直到读到换行。

第 4 行设置注释类型, 如果文件中遇到 '/' 开头的行则会自动跳过该行。

第 5 行设置将多个连续分隔符认为是一个分隔符, 比如读取的文件中有若干个连续空格, 都会被当做一个分隔符; 与之相反的还有一个选项 'EmptyValue', 当读到两个分隔符中没有内容时, 可以将空值替换成其他内容, 比如 -Inf 等。

返回的变量 C 是一个 cell 行向量, 每一个元素存放对应列内容。

textscan 还有若干选项, 基本上会被用到的都有, 可以自己查询一下 Help。

另外, 还有文件格式不统一可以灵活的结合其他函数来实现文件的读取。例如下面文件中, 冒号之前为用户名, 之后为对应的数据。我们读取的时候就可以建立结构体数组, 'usr' 和 'data' 分别存放用户名和数据。文件和程序如下:

```
1 Datafile: 'data.txt'
2 Pan:0,3,4
3 Zhao:1
4 Qian:0,3
5 Sun:2
6 Li:1,4
```

```
1 fd=fopen('data.txt');
2 filedata=textscan(fd,'%s %s','delimiter',':', 'HeaderLines',1);
3 fclose(fd);
4
5 for i=1:length(filedata{1})
6     Pan_struct(i).usr=filedata{1}(i);
7     Pan_struct(i).data=str2num(filedata{2}{i});
8 end
```

## 7.5 xlsread 与 xlswrite 函数

Excel 文件也是我们常用来存放数据的文件格式。由于这类文件跟普通的文件格式不同, 所以不能用 dlmread, csvread 或者 textscan 函数进行读取, 而可以用 xlsread 读取、用 xlswrite 进行写入。

在安装了微软的 Excel 软件的电脑上, 这两个函数可以读写任何 Excel 文件支持的文件格式; 在没有安装微软 Excel 的电脑上, 可以只用 basic 模式, 支持读写 xls, xlsx, xlsxm 等基本格式文件。

**注意事项:** 1. 每次使用 `xlsread` 和 `xlswrite` 函数, 都会打开文件, 然后读写, 最后再关闭。大部分时间花费在打开和关闭文件上, 所以尽量打开之后一次性读取文件, 之后再将不同的行列赋值给不同的变量。2. 当文件已经在 Excel 软件中被打开, 再使用这两个函数时会提示错误, 这跟使用 `text` 等函数是不同的。

## 7.6 图像、声音读写

`imfinfo` 函数用来查看图像文件的信息。可以指定文件类型, 也可以从互联网上读取图片的相关信息。

```
1 info1 = imfinfo('ngc6543a.jpg')
2 info2 = imfinfo('moon', 'tiff')
3 info3 = imfinfo(['http://hiphotos.baidu.com/zhidao/pic/', ...
4                 'item/bf096b63f6246b605e8b7190ebf81a4c500fa249.jpg']);
```

`imread` 函数和 `imwrite` 分别用于图像文件的读取和写入。支持很多格式的图片格式读取, 包括: BMP、JPEG、PNG、CUR、JPEG、PPM、GIF、PBM、RAS、HDF4、PCX、TIFF、ICO、XWD 等。对于不同的图片格式, 可以选择使用不同的参数对读写进行设置。

另外可以使用 `im2frame` 函数将图片作为帧拼接成影像, 可以使用 `movie` 函数进行播放。

`audioinfo` 函数用来查看音频文件的信息, 会得到采样率、信道数、时间长度、总采样点数、压缩方式等。

`wavread` 和 `wavwrite` 函数分别是对 '.wav' 文件的读写, 但是 MATLAB 中也提示, 这两个函数将在以后被删掉, 建议采用 `audioread` 和 `audiowrite` 函数代替。读音频函数可以读取一部分, 返回采样速率 `y` 和 `Fs`。写音频函数需要输入文件名、数据、采样率以及其他可选的参数值。

`MovieReader` 函数用来读取视频文件, 产生一个对象, 对象中有读出的视频文件的各种信息, 再将上述信息创建一个 `movie` 结构体, 通过 `movie` 函数就可以播放了。 `MovieWriter` 是函数用来创建一个用于写视频文件的结构体, 使用 `writeVideo` 函数写入视频文件。

## Chapter 8

# 并行计算

我之前都没弄过并行运算，不过在某些因素的驱动下，我就现学了一下。下边把遇到的问题和解决方法做一下简单的总结。

### 8.1 并行运算框架

MATLAB中有好几种支持并行计算的方式，矩阵的基本运算、某些工具箱都是可以直接进行多线程并行的。另外，MATLAB 可以在多核上进行多进程的并行，而且还可以在多处理器、多计算机上进行并行。我这主要是讲关于 `parfor` 的几点问题。

每个能够执行运算的 lab 都可以叫 worker，通过我们代码制定不同的方式，MATLAB 就能根据不同的情况进行的不同并行。对于底层的细节，我们不需要知道太多。我们需要了解的是何种情况下适合并行、何种情况不能并行。图 8.1来自 MATLAB 官方帮助，展示了 worker 进行并行计算的工作方式。

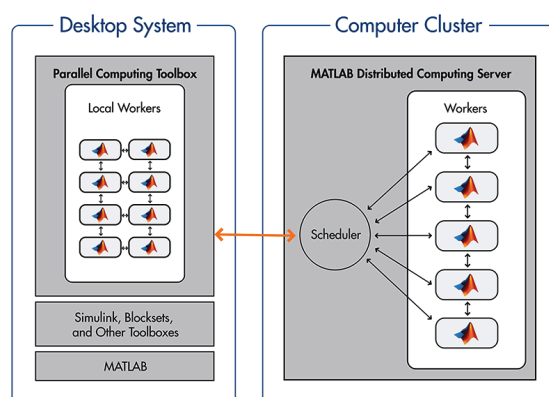


图 8.1: MATLAB 并行运算框架

## 8.2 配置 matlabpool

matlabpool 函数的作用是 ‘Open or close pool of MATLAB sessions for parallel computation’，用来分配执行多线程的 worker。

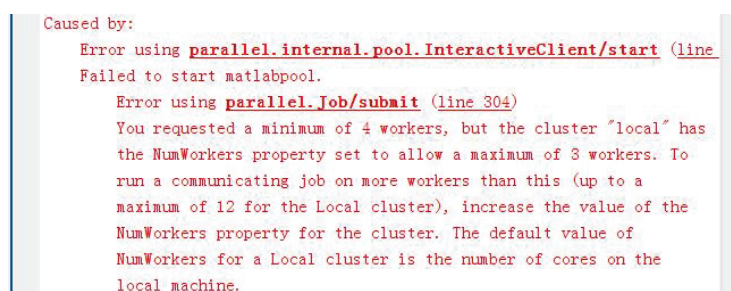
matlabpool open local n 命令表示打开本地 n 个 worker，open 可以省掉。matlabpool close 命令用来关闭所有打开的 worker，可使用 force 选项，强制关闭。

matlabpool size 命令用来检查开启 worker 的个数。如果没开，就会返回 0；如果开了，就会显示打开的 worker 个数。如果 matlabpool 已经 open，再去 open 就会出错；同样的，如果 matlabpool 已经 close，再去 close 也会出错。所以在开关 matlabpool 之前，要使用一下该命令。

matlabpool 后边加参数方式与 save 函数类似，可以直接跟着参数，也可以用括号加字符串的形式。matlabpool('size') 和 matlabpool size 功能是一样的。

```
1 if matlabpool('size') == 0
2     matlabpool open local 2;
3 end
4 if matlabpool('size') > 0
5     matlabpool close;
6 end
```

**设置 worker 个数上限：**我的电脑是 i5 的 CPU，是双核 4 线程，但我无论是打开任务管理器还是设备管理器看他都是 4 核的，也就是说系统就把他当成 4 核的了。所以我觉着我开 4 个 worker 应该没问题。使用 matlabpool ... open local 4 命令，结果报错如图 8.2，提示要求开辟的个数超过了 local worker 的数量上限：



```
Caused by:
Error using parallel.internal.pool.InteractiveClient/start (line
Failed to start matlabpool.
Error using parallel.Job/submit (line 304)
You requested a minimum of 4 workers, but the cluster "local" has
the NumWorkers property set to allow a maximum of 3 workers. To
run a communicating job on more workers than this (up to a
maximum of 12 for the Local cluster), increase the value of the
NumWorkers property for the cluster. The default value of
NumWorkers for a Local cluster is the number of cores on the
local machine.
```

图 8.2: 分配 worker 超上限出错

MATLAB 2013 之前的版本：主窗口的 Parallel 选项 → Manage Configurations 右键 → local Properties Scheduler 选项卡 → ClusterSize 改上限。为什么是 8？因为 “The ClusterSize for a local scheduler must be between 1 and 8.”，MATLAB 的 local scheduler 的 ClusterSize 最大是 8。



MATLAB 2013 之后的版本：主窗口中 Parallel 选项 → Manage Cluster Profiles... → Edit → local Properties → NumWorkers 改上限。

### 8.3 parfor 并行

parfor 函数是 parallel for 的缩写，其原理是将循环拆分后分配给不同的 worker 各自执行其中的部分。例如在进行蒙特卡洛(Monte Carlo)模拟，parfor 循环就很有用。但使用时需要注意的是：

①. 各个循环之间不能够有相互依赖的关系；下面的代码中，我们需要求 3 次 2000x2000 的 rand 矩阵的特征值，这三个次循环没有任何的关系，每次结果不相互影响，这时就可以使用 parfor 函数并行计算。

```
1 for i=1:3
2     d(:,i) = eig(rand(2000));
3 end
```

另外还有一种看似每次循环有联系的，但实际没什么联系的循环也可以使用 parfor 进行并行计算。例如：将 s 从 0 依次加到 100000，每次虽然 s 与上一次结果有关系，但是换个思路，如果我打乱了加和的顺序，结果也是一样：

```
1 s = 0;
2 for i=1:100000
3     s = s+i;
4 end
```

以下的程序就是依赖关系：

```
1 f=zeros(1,50);
2 f(1)=1;
3 f(2)=2;
4 for n=3:50;
5     f(n)=f(n-1)+f(n-2);
6 end
```

②. 不支持非透明的运算：“透明”可以理解成其中的参数都是可以判断的、执行的结果一定程度上可以预测。非透明的运算如：

```
1 save data;
2 load data;
3 eval('a=1;');
```

③. 嵌套问题：parfor 里边可以嵌套 for，但是不能嵌套 parfor，不过 parfor 内的函数可以含有 parfor。

④. parfor 循环内有通信消耗，计算量不大的程序使用 parfor 可能不会加快运算，甚至会增加程序运行时间。所以并行适合不同的情景，不能盲目的

使用。

另外还有一个重要的并行函数 `spmd`，是 `single program, multiple data` 的缩写，同时用几个 `worker` 对不同的数据使用同样的程序进行处理。这个函数在此先不做介绍。

## 8.4 测试并行循环

写一段测试并行计算的代码：

```

1  if matlabpool('size') == 0
2      matlabpool open local 4;
3  end
4
5  tic;
6  parfor i=1:20
7      d(:,i) = eig(rand(1000));
8  end
9  toc;
10
11 if matlabpool('size') > 0
12     matlabpool close;
13 end

```

下表为选择不同个数的 `worker` 时运行测试程序所需要的时间。从表中可以看出当 `worker` 的数量达到 5 个之后，再继续增加运行时间也不会减少，甚至会略有增加。

NumWorkers	1	2	3	4	5	6	10
RunTime(s)	17.6	12.3	10.8	9.06	8.80	9.08	9.39

图 8.3 和图 8.3 则显示了在 3 个和 5 个 `worker` 同时进行工作时的 CPU 占用情况。可以看出，3 个 `worker` 的 CPU 利用率已经达到 75% 了，4 个及以上就可以接近 100%。

但 5 个并没有比 3 个明显快多少。原因可能是 `worker` 多了，传递信息的开销会变大。8 个一块并行，也跟 5 个差不多的速度。



映像名称	用户名	CPU	内存 (K)	描述
MATLAB.exe...	Panzh...	25	169,552 K	MATLAB (R2013a)
MATLAB.exe...	Panzh...	25	166,740 K	MATLAB (R2013a)
MATLAB.exe...	Panzh...	25	166,792 K	MATLAB (R2013a)
QQ.exe *32	Panzh...	06	35,032 K	QQ2013

图 8.3: 3 个 `worker` 同时工作

映像名称	用户名	CPU	内存 (K)	描述
MATLAB.exe...	Panzh...	25	187,956 K	MATLAB (R2013a)
MATLAB.exe...	Panzh...	25	188,108 K	MATLAB (R2013a)
MATLAB.exe...	Panzh...	22	187,680 K	MATLAB (R2013a)
MATLAB.exe...	Panzh...	14	187,336 K	MATLAB (R2013a)
MATLAB.exe...	Panzh...	14	188,048 K	MATLAB (R2013a)

图 8.4: 5 个 worker 同时工作

通过上述测试的图表，我们发现：在未达到硬件资源上限时，占用硬件资源随 worker 的增多而成倍增加，但效率提高不到一倍；在达到了硬件资源上限后，更多的 worker 不能带来效率的提高，随着 worker 的继续增加，效率甚至有降低。个人觉着分了 worker 更多之后主从之间的通信增多、分配资源耗费更多的时间，引起了效率降低。

## 8.5 parfor 中的变量类型

变量在 parfor 的使用尤其需要注意，用不好就会碰到下面的错误：

??? Error: The variable f in a parfor cannot be classified.

See [Parallel for Loops in MATLAB, "Overview"](#).

parfor 函数中的变量在图 8.5 中做出了介绍，该图出自 MATLAB 自带的帮助。

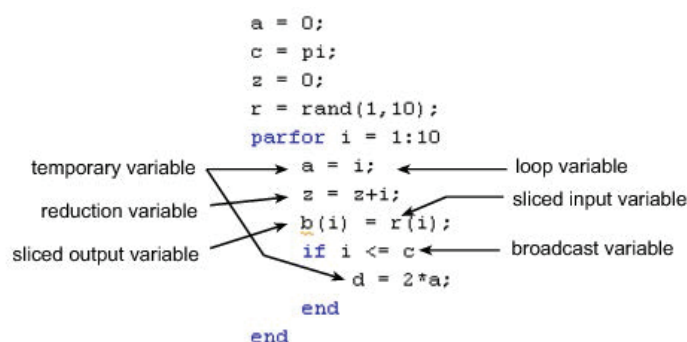


图 8.5: parfor 中的变量类型

循环变量 (loop variable)，特指 parfor 的循环变量，在循环内部禁止做任何的改变。

可切变量 (sliced variable) 是指可以被切割成片，然后分别用于各个 MATLAB worker 的变量，每次循环迭代作用于可切变量的不同位置片，使用这种变量可以减少主从通信。若  $i$  为循环变量，则：只有按第一级索引切的才可以，如  $A(i)$  不是可切变量； $A(i)+A(i+1)$  中  $A$  不是可切变量，因为每次循环的  $A$  是不独立的；可切变量的下标只能是一个数、':'、'end'，不能使用

一个局部范围，比如  $A(i, 1:2)$  会出错；可切变量的下标必须是透明的，不能使用函数，不能使用结构体的 field value；可切变量的必须保持固定尺寸，在 `parfor` 中禁止改变大小。该类变量即可以作为输入也可以作为输出。

广播变量 (broadcast variable) 是指循环中使用的且不随循环发生任何改变的变量，相比于可切变量会引起更大的主从间的通信，有时可以在 `parfor` 内部采用临时变量会比使用广播变量更高效。

缩减变量 (reduction variable, 是否该如此翻译) 用于迭代的结果与每次迭代都有关系，但是跟迭代次序没有关系的循环。正如之前讲依赖关系时介绍的那样，这些使用诸如  $X = X + \text{expr}$  形式表达式的  $X$  便是缩减变量。这些运算还包括：减 ( $-$ )、乘 ( $*$ )、点乘 ( $.*$ )、与 ( $\&$ )、或 ( $|$ )、最大/最小 ( $\min/\max$ )、矩阵合并 ( $[X; \text{expr}]$ )、元胞合并 ( $\{X; \text{expr}\}$ )、交集/并集 ( $\text{union/intersect}$ )。其中矩阵合并和元胞合并时虽然是迭代次序不同，但是当迭代变量  $i = N$  时，一定是将  $\text{expr}$  产生的结果放在第  $N$  行，而不是乱序排列。需要注意：①. 在 `parfor` 循环中的缩减变量只能使用其中的一种运算，换句话说，不能在有的条件下对  $X$  进行加操作而在另一种情况下对其进行减操作；②. 使用矩阵乘法 ( $*$ ) 和矩阵合并时只能让缩减变量作为第一个参数，像  $A = [\text{expr}, A]$  就是错误的。

临时变量 (temporary variable) 的定义是 “any variable that is the target of a direct, nonindexed assignment, but is not a reduction variable”。简单的讲就是 `parfor` 循环内非索引赋值的变量，也不是一个缩减变量的变量。有别于广播变量，临时变量在 `parfor` 循环中被改变。注意：临时变量可以和 `parfor` 外的变量同名，但是在循环内做修改不会改变循环，这种关系就类似于调用子函数时的局部变量；这里所谓的赋值其实特指初始化，初始化后可以通过索引对变量值进行改变，下一小节中的例子中会有介绍。

这 5 种变量定义都比较严格，各种变量具有互斥性，是其中的一种变量就不能是另外一种。`parfor` 中除了这 5 种变量之外的任何变量都不是合法的变量，都会提示变量不能被分辨进而导致程序不能运行。

**注意：**`parfor` 中非常不建议使用全局变量，虽然在其中使用只会给 warning，但 `parfor` 不仅可以在一台电脑上运行，还可以在不同的机器上同时运行，而每台计算机的工作空间可能不相同，会导致不可预知的意外或错误。所以 MATLAB 建议在使用 `parfor` 之前，先将全局变量保存为本地变量。

## 8.6 两个具体问题

使用 `parfor` 最容易出的问题还是在变量的使用上。本节就介绍两个实际中遇到的两个问题的解决。

使用索引对矩阵进行赋值用的还是比较多的，但是在 `parfor` 中却要求临时变量是 “nonindexed assignment”。接下来对临时变量使用索引赋值进行探

讨，下边的程序会提示 a 变量不能被分类。

```
1 parfor i=1:10
2     for j=1:10
3         a(j)=j;
4     end
5 end
```

最初的想法是 a 应该是一个临时变量，但是又是通过索引进行赋值导致了 MATLAB 不能将它归入到临时变量类别中。改为如下程序，在 parfor 循环内首先对 a 进行初始化，然后再使用索引进行赋值就不会报错了。

```
1 parfor i=1:10
2     a=zeros(10);
3     for j=1:10
4         a(j)=j;
5     end
6 end
```

其次，讨论一个可切变量的赋值问题。我想计算一个乘法表，使用如下的程序：

```
1 result=zeros(10,10);
2 parfor i=1:10
3     for j=1:10
4         result(i,j)=i*j;
5     end
6 end
```

很明显其中的 result 变量是一个可切变量 (sliced variable)，因为他的行索引使用的循环变量，列索引使用的是一个数。但是当我想生成下三角乘法表时，将 for 循环变量 j 的范围改成 1:i 之后，提示 result 是不可分类的变量。虽然 j 在每次循环中仍然是一个数，但是可能会导致 MATLAB 在切割 result 时出问题，所以 result 变量不再是一个可切变量。

要解决此问题，可以在中间用临时变量链接，修改后的程序中增加了 resultTemp 临时变量，先将对应每个循环变量 i 的行计算出来，存入循环变量，再使用循环变量将整行赋给可切变量。

```
1 result=zeros(10,10);
2 parfor i=1:10
3     resultTemp=zeros(1,10);
4     for j=1:i
5         resultTemp(j)=i*j;
6     end
7     result(i,:)=resultTemp;
8 end
```

MATLAB 并行计算还有好多内容，还有一个比较重要的 `spmd`(single program, multiple data)，可以将一块数据分割，然后分别在多个 labs 上跑，也是比较方便的。

如果你在使用 `parfor` 时遇到更多的问题可以去 MathWorks 工程师 Loren 的博文 [Using parfor Loops: Getting Up and Running](http://blogs.mathworks.com/loren/2009/10/02/)<sup>1</sup> 中找找答案，里边除了介绍 `parfor` 的使用外，还有 80 多条评论的回复，解答了一些常见的问题。

并行里边对语法要求比较严格，随着 MATLAB 版本的提高、功能的增加，部分严格的要求会被取消。不是逗你玩，上边介绍的博客中提到的几个问题，在现在的版本上已经不是问题了。

---

<sup>1</sup> <http://blogs.mathworks.com/loren/2009/10/02/>

## Chapter 9

# MATLAB Coder ——生成独立的 C/C++ 代码

由于项目需要，我们经常需要将自己设计的一些 MATLAB 方法在 ARM 平台上进行实现。但初次由 MATLAB 改 C 工作量过大，而其后每一次修改 MATLAB 都需要对应修改 C 又过于繁琐。所以迫切想找一种简单方便的方法来实现这枯燥乏味的工作。

于是，我们找到了 MATLAB Coder，它是 MATLAB 从 2011a 版本开始加入的新功能。能够直接生成清晰可读的、独立的、可移植的 C 和 C++ 代码。MATLAB 官方网站上曾经出过相关的在线研讨会，名字叫做“使用 MATLAB Coder 从 MATLAB 生成 C/C++ 代码”。<sup>1</sup>

我在这里写一写基本过程和实际遇到的问题，欢迎大家补充。

### 9.1 准备工作

MATLAB 版本至少在 R2011a 以上。当然 MATLAB 版本越高，Coder 的版本自然越高，支持转换的函数自然就越多。（是否需要安装 mbuild？）

### 9.2 生成 C 代码基本步骤

#### 9.2.1 编写 MATLAB 函数

首先需要根据自己的算法，写好 MATLAB 程序，虽然 MATLAB Coder 支持的 MATLAB 函数非常有限，但是还是先按照自己思路，写好程序才对。在后续处理步骤中，我会介绍怎么把一些不支持的函数变成“支持的函数”。

---

<sup>1</sup> <http://www.mathworks.cn/company/events/webinars/wbmr58105.html>

另外需要强调的是，MATLAB 的一个函数，就是 C 的一个函数。期望的 C 函数肯定是在工程中的一个中间函数，这个函数应该有输入变量、输出变量。只要是 C 函数里边有的，MATLAB 的方法也一定要写上；C 函数里没有的，MATLAB 里边也不应该出现；变量一一对应。

在 MATLAB 中写好函数后，要再外边写一个测试用的脚本。当脚本能运行并得到正确结果之后，再进行后续的工作。

这里需要进行转换的例子是一个简单的函数 `my_add`。

```
1 function z = my_add(x,y) %#codegen
2 z = x + y;
3 end
```

在函数声明之后或者在之前第一行加上 `%#codegen`，来表示该函数是要生成代码的。添加此指令可以提示 MATLAB 的代码分析器，帮助诊断和解决代码生成过程中将可能产生的错误和冲突。

## 9.2.2 定义输入变量

当 MATLAB 程序写好后，可以打开 Coder 了。在命令窗中敲入 `coder` 命令即可，出来 Code Generation Project 界面。

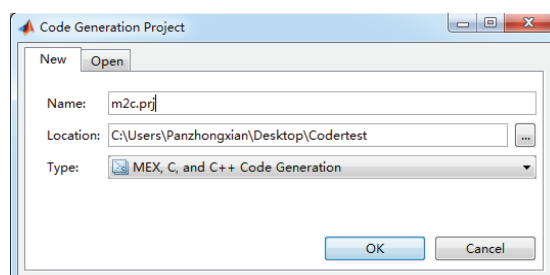


图 9.1: 新建或打开代码生成工程

此时，需要建立一个 Coder 工程，也可以打开已建立的工程。接下来在 Overview 中添加 MATLAB 的方法，如果还调用了其他的方法，那么只需要最外层的那个函数即可。添加完函数之后，会出现函数的输入变量，点击变量右侧一栏可以选择输入变量类型和长度，如果需要可以定义全局变量。这个输入变量的尺寸也是支持 MATLAB 中可变大小的输入变量的。

## 9.2.3 设置生成代码选项

打开 Build 选项卡，可以选择输出类型。我们想要把函数作为整个工程的一部分，可以选择生成 C/C++ 静态库，并勾选“只产生的代码”；如果是单独的一个程序作为 main 函数，可以选择生成可执行文件。



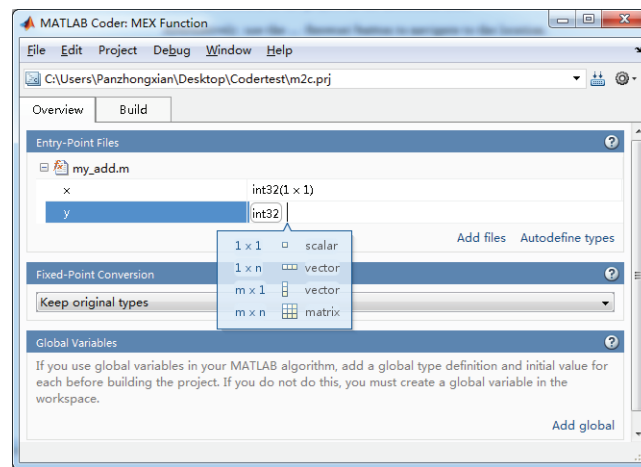
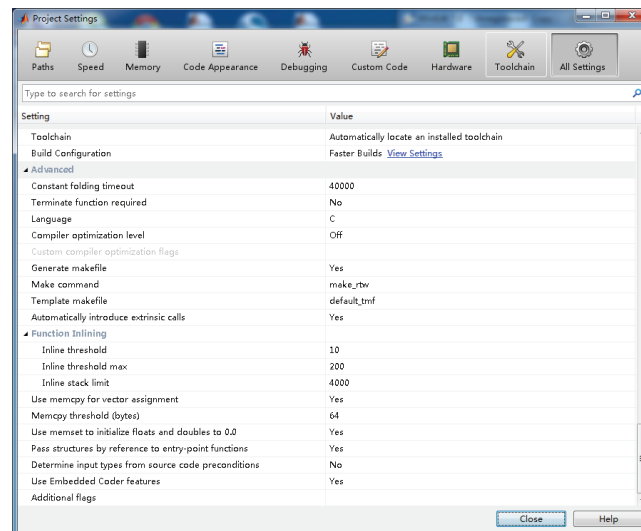


图 9.2: 设置输入变量类型和尺寸

点击设置键（齿轮状）或从菜单栏中选择 Project → Settings... 可以选择更多的选项。



其中，重要的几项有：

a. 选择语言类型：

All Settings → Advanced → Language → C/C++；

b. 合并生成的文件：

Code Appearance → Generate file partitioning method → Generate all function into a single file

c. 将源代码作为注释写入：

Code Appearance → Comments → MATLAB source code as comments

d. 取消支持无穷大数:

Speed → Support non-finite numbers (勾掉)

e. 取消终止函数:

All Settings → Advanced → Terminate function required → No

如此设置会将代码生成的更加简洁明了。剩下的选项大家可以根据自己需求进行探索。

## 9.2.4 生成 C 代码

在生成 C 代码之前，**必须检查**“项目代码生成准备状态”(Project Code Generation Readiness)，对这个功能后边会做介绍。

点击 Build 键即可生成 C 代码了。论文生成过程是否有错误的，都会产生一个非常有用的 Code Generation Report，如图 9.3。

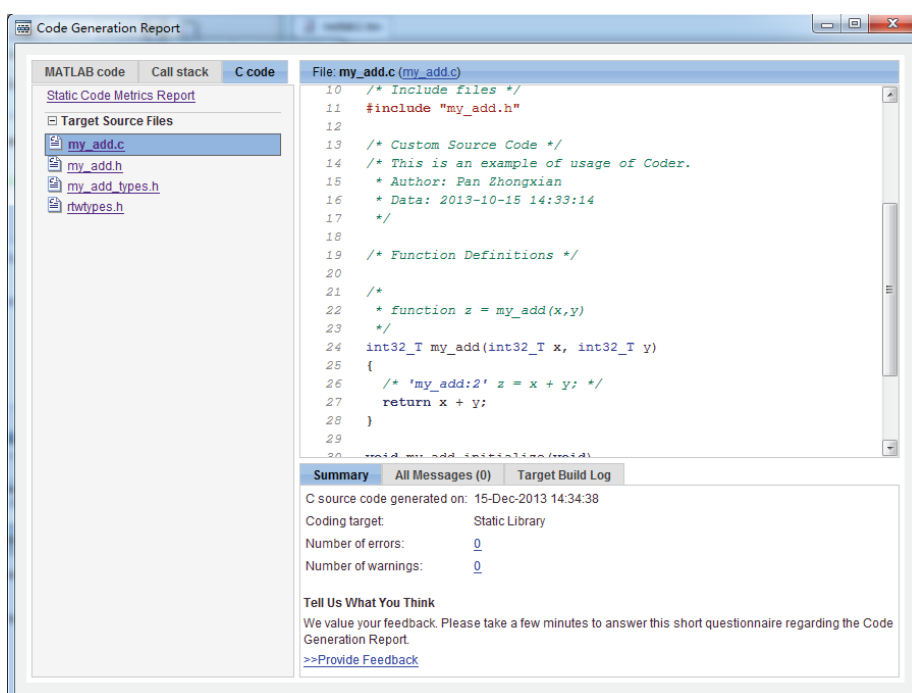


图 9.3: 生成 C 代码后的 Report

Report 中有三个选项卡：在 MATLAB code 选项卡中，我们可以将光标移动到变量和函数上去，会提示关于该变量或函数的相关信息，在右下方还会有一个 Summary 和 Variable 等信息的分类显示；C code 展示的是生成的 C 语言代码，在代码中可以加入固定的代码信息；还有一个 Call stack 选项，和 MATLAB code 相似。如果有错误，可以根据 Report 里的提示，修改掉错误和 warning。

如果没有错误将会产生了四个文件，my\_add.c, my\_add.h, my\_add\_types.h, rtwtypes.h。其中，前两个文件是我们转换成的函数的 C 和 H 文件；第三个存放的是函数中自己定义的结构体，如果没定义就是空；rtwtypes.h 的含义是 Real-Time Workshop Types（RTW 是生成 C 代码的一个工具），这个文件中包含了 MATLAB 中的常规数据。

这四个文件没必要再去合并了，因为别的文件可能也需要使用 rtwtypes.h 文件，保留一个可以共用。另外 my\_add\_types.h 文件中若无内容，删掉即可，别忘了删掉对应主要头文件 my\_add.h 函数中的 #include 行。另外，会产生一些空的初始化函数，如 my\_add\_initialize(void) 函数，没有有用但也没必要删。

### 9.2.5 验证 C 代码运行的正确性

新建一个 C 工程，建立一个主函数，将几个文件放到工程中并 include 到主文件中。定义输入变量和输出变量，运行程序并检查结果是否正确。尤其需要注意的是变量类型要与定义一致。

```
1  #include "my_add.h"
2  main()
3  {
4      int a,b,c;
5      a=1;
6      b=2;
7      c=my_add(a,b);
8  }
```

这里 C 的运行结果和 MATLAB 中的结果是一样的，证明转换的函数没有什么问题。

## 9.3 保证代码生成的正确性

为保证代码生成的正确性，我们必须要认真做好每一步的操作。其中，有两个步骤在生成 C 代码基本步骤一节中没有单独列出，但是确是保证正确性的关键步骤，本节做一下讨论。

### 9.3.1 Project Code Generation Readiness

之前也提到过，在需要转换的代码前加上 %codegen，是为了标示该函数是用来生成代码的，加上这个标识之后 MATLAB 的代码分析器可以帮助我们诊断和修复可能导致在代码生成过程中导致崩溃的错误。由于之前我们的程序简单，没有错误，所以没有机会使用这个功能。现在我们写一个有问题的 MATLAB 代码，学习使用这一功能。

```
1  %#codegen
2  function t = my_array
3  t = 1;
4  for index = 1:15
5      t(index)=index;
6  end
```

这一函数首先 `t` 定义为一个 `double` 型的数，然后在循环中将变量尺寸逐渐增加，在 MATLAB 中没有问题，但是转换成 C/C++ 代码的时候就会出现问題。

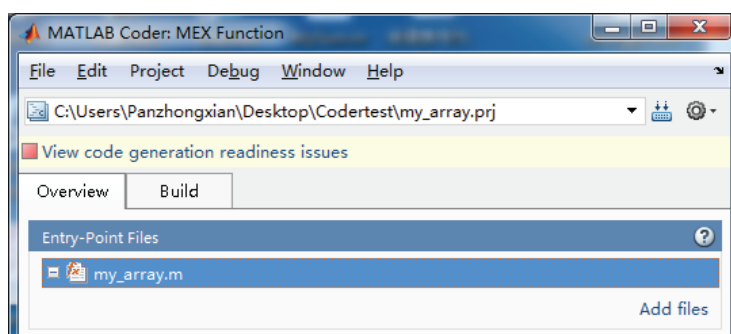


图 9.4: 出现 readiness issues

注意添加上文件之后，出现红色的方块，然后提示 ‘View code generation readiness issues’，点击就会进入 ‘Project Code Generation Readiness’。如果说

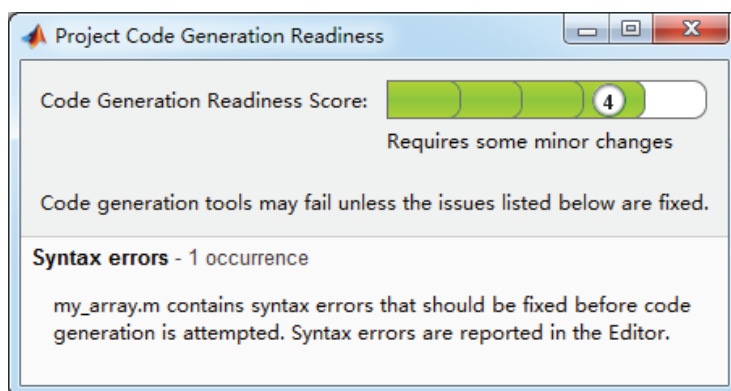


图 9.5: 出现 Readiness Issues

程序的得分是 4 分，提示有一处语法错误。这里的错误必须解决，否则即便通过，也会出错。回到 Editor 中查看右侧进度条中的 Warning Found，提示 ‘Code generation does not support variable ‘t’ size growth through in-

dexing.’ 这时候我们对 t 初始化的时候分配 15 长的数组即可。将 t=1 改成 t=zeros([1,15]) 即可。

如果不改，代码生成的过程是可以顺利走下来，但是结果是错误的，我们可以看看忽略这个错误进行转换后的 C 代码：

```

1  real_T my_array(void)
2  {
3      real_T t;
4      int32_T b_index;
5
6      /* 'my_array:3' t=1; */
7      t = 1.0;
8
9      /* 'my_array:4' for index=1:15 */
10     for (b_index = 0; b_index < 15; b_index++) {
11         /* 'my_array:5' t(index)=index; */
12         t = 1.0 + (real_T)b_index;
13     }
14
15     return t;
16 }
```

很明显，此处的 t 一直是一个 double 的变量，每次都是给 t 进行赋值，最终得到 t=15.0，而不是一个数组。

**提示：**如果显示 “Unable to determine code generation readiness”，则是因为需要转换的文件不再当前的工作路径中。

### 9.3.2 使用 Code Generation Report

如果在检查 readiness 中没有检查出问题来，并不代表这个程序就可以正确的进行代码转换了。即使一些明显的错误，readiness 的检查也不一定能够检查出来。例子是在之前求和代码基础上改的：

```

1  %#codegen
2  function z = my_add(x,y)
3      z = x + y;
4      z(2) = z;
5  end
```

很明显的是变量 z 首先被声明的是一个 1 标量，而之后又要变成向量，这种变量类型的转换在未开启“变化的变量尺寸”时，这种代码时不能够被转换的。但是 readiness 的检查没有检查出任何的错误，但是在 Build 时候会报错。这时候需要点击 “Open error report” 的超链接，打开 Code Generation Report 查看其中的错误。

Report 会提示错误的具体内容。错误提示如我们所分析的那样：“Index

expression out of bounds. Attempted to access element 2. The valid range is 1-1”。

通过上面的分析发现，这两种检查代码的方法是互补的，有的错误只会在某种情况下给出错误提示，所以我们不能只依赖于一种方法。而当两种情况下错误都排除了之后，我们基本上就可以放心的去做代码转换了。

## 9.4 代码转换中的变量

变量问题是在转换过程中遇到的最多问题。User's Guide 中也有很大一部分专门介绍变量，本节也做一个介绍。

### 9.4.1 变量类型和大小的变化

在 MATLAB 中，我们可以在任何地方给变量进行赋值，而且多次赋值的变量类型、大小以及是复数还是实数都是可以变化的。比如 `x = [x; 1]`；就是在每次操作都会改变 `x` 的尺寸。但是在 C/C++ 中，变量都是需要先声明后使用，而且变量通常是类型、大小都确定的。这样一来，类型、尺寸如果有变化，生成代码时就会产生错误。

首先，先举个上一节类似的例子，未声明变量就直接使用变量，往往会报错 “Undefined function or variable 't'. The first assignment to a local variable determines its class”。

```
1 for index=1:b
2     t(index)=index;
3 end
```

另外，大小、类型、复数实数的改变也会有类似的问题。如果一个数定义的时候是个实数，而在运行的过程中又赋值为复数，在生成代码时就会可能会出错。这里说的是“可能”而不是“肯定”，因为 MATLAB Coder 在转换代码时，会进行一定的优化。例如对左边的 MATLAB 代码进行转换，两种检查代码错误的方法都通过了，Build 得到右侧的 C 代码，可以发现，在最后一句之前对 `x` 的操作都被优化掉了。

```
1 %#codegen
2 function x = foo(c)
3     x = 3;
4     x = complex(0);
5     x = int32(3);
6     x = 1 + 2i;
7     end
```

```
1     creal_T foo(void)
2     {
3         creal_T x;
4         x.re = 1.0;
5         x.im = 2.0;
6         return x;
7     }
```

在使用变量之前先要明确的定义。通过赋值定义变量，会将变量的值、类型、尺寸和是否是复数都确定下来。如果需要定义结构体，需要将结构体的各个域内的值都明确的定义。应该坚持“one variable/one type”的原则。如果需要某个变量的值但是却要使用别的类型，那就需要使用类型转换运算符。

不能在 if 中利用赋值进行定义，在生成代码的时候，会判断这种情况可能变量会没有被定义。可以改成在 if 之前就定义，而在 if 中的是对值的改变。

如果是可以确定长度和类型的变量，在开始正确的声明就好了；如果不确定长度但有知道长度的范围，那么分配足够的空间即可。假设 b 一定不会超过 100，那么就可以预分配长为 100 的数组。

解决上述麻烦的一个简单方法就是勾选变化尺寸支持选项，但是。所以尽量通过其他方法改进一下。

通过在 Coder 的设置中勾选是否支持可变尺寸变量来选择，可以解决很大一部分变量问题。但是动态分配会降低一定的效率，也不利于我们对待程序严谨的态度，要有就是这个方法也不是万能的，仍有一些问题仍不能兼容，如确定变大小的 N 维矩阵的尺寸时。

如果当你勾选了“Enable variable-sizing”，仍然在变量上出问题的话，那就需要去用户指导手册中细心找找答案了。

### 9.4.2 MATLAB函数的变量类型限制

函数的输入类型往往是固定的，而且用到的 int32 很多，但 MATLAB 里的许多函数只能对 double 型数据进行处理，所以应该在进入函数之后进行类型转换。而根据“one variable/one type”的原则，我们应该使用其他变量名来保存转换成 double 的变量。

如果不在进行类型转换，会出现下边的错误：

```
>> coder -build m2c.prj

??? Function 'sin' is not defined for values of
class 'int32'.
```

图 9.6: 不能对 int32 型变量使用 sin 函数

另外，当两个类型不同的变量进行四则运算的时候，也会出现类型不同的错误提示，也需要将变量转成相同的类型。

## 9.5 不确定尺寸的输入

MATLAB 支持尺寸大小不同的输入变量，而转换成代码时，我们需要设置输入变量的类型和尺寸大小，这让我们觉着转换成的代码只能够接收固定大

小的变量。其实不然，我们其实在转换的代码中也能够使用不确定尺寸的输入变量。

### 9.5.1 确定尺寸上限的输入

我们在图 9.2 中，置输入变量尺寸的时候，我们可以选择变量的尺寸是 `int32(1 x 2)` 这个意思第二维尺寸是不确定的，但是最大不超过 2。另外，当我们不确定上限时，我们还可以选择使用 `:Inf`。三种选择都能转换函数，但是转换出来的是三种不同的结果。

为了方便比较，我们使用三种尺寸选择，对 `my_add` 函数分别进行转换，另外为了比较，将使用 `int32(1 x 2)` 尺寸的也加入比较。对生成的四个函数声明进行比较。

```

1  /* x, y both use size of int32 1 x 1 */
2  extern int32_T my_add(int32_T x, int32_T y);
3
4  /* x, y both use size of int32 1 x 2 */
5  extern void my_add(const int32_T x[2], const int32_T y[2],
6                    int32_T z[2]);
7
8  /* x, y both use size of int32 1 x :2 */
9  extern void my_add(
10     const int32_T x_data[2], const int32_T x_size[2],
11     const int32_T y_data[2], const int32_T y_size[2],
12     int32_T z_data[2], int32_T z_size[2]);
13
14 /* x, y both use size of int32 1 x :Inf */
15 extern void my_add(const mxArray_int32_T *x,
16                   const mxArray_int32_T *y, mxArray_int32_T *z);

```

首先是两个标量作为输入，输出也为标量。使用的就是普通的形参传递，结果返回。

第二是两个向量作为输入，输出也为向量。传入两个 `const` 的数组地址以及一个非 `const` 的数组地址，由于尺寸确定，所以不需要输入其他信息。

第三是使用可变大小但是上限确知的向量，输出的大小受限于输入。所以除了数组地址外，每个数组变量之后需要加一个 `size` 来告诉函数输入变量的尺寸。需要注意的时候，如果使用一个标量，需要取地址，而不是直接写上变量。

最后是使用可变大小输入同时不确定上限的向量，输出也取决于输入。这个格式和第二种固定输入向量的结构是类似的，区别在于该函数的输入是 `mxArray_int32_T` 的结构体的指针，输入输出的数据都放在这类结构体中。



## 9.6 emxArray 数据结构体

emxArray 结构体是一种可嵌入的 mxArray 结构体 (An embeddable version of the MATLAB mxArray), 对应不同数据类型有不同种类的 emxArray 结构体 命名规则为 emxArray\_<baseTypedef>, 其中 <baseTypedef> 指的就是 rtwtypes.h 函数中预定义的数据类型, 包括 real\_T, time\_T, boolean\_T, int\_T, uint\_T, ulong\_T, char\_T, byte\_T 等等许多类型。

以 real\_T 为例, 介绍 emxArray\_real\_T 结构体。

```
1 struct emxArray_real_T
2 {
3     double *data;
4     int *size;
5     int allocatedSize;
6     int numDimensions;
7     boolean_T canFreeData;
8 }
```

首先是 \*data 指向数据数组, 然后 \*size 指向尺寸数组, 然后 allocatedSize 表示元素个数, numDimensions 表示数组维数, canFreeData 表示是使用 MATLAB 自动决定何时释放内存还是通过调用函数来释放内存。

在生成代码的过程中会产生一些工具函数, 它们用于产生和销毁 emxArray 结构体。

```
1 //Creates a new 2-dimensional emxArray
2 emxArray_real_T *emxCreateWrapper_real_T(
3     real_T *data, int32_T rows, int32_T cols)
4 //Creates a new N-dimensional emxArray
5 emxArray_real_T *emxCreateWrapperND_real_T(
6     real_T *data, int32_T numDimensions, int32_T *size)
7 //Creates a new two-dimensional emxArray initialized to zero.
8 emxArray_real_T *emxCreate_real_T(
9     int32_T rows, int32_T cols)
10 //Creates a new N-dimensional emxArray initialized to zero.
11 emxArray_real_T *emxCreateND_real_T(
12     int32_T numDimensions, int32_T *size)
13 //Frees dynamic memory
14 emxDestroyArray_real_T(emxArray_real_T *emxArray)
```

## 9.7 其他几个问题

### 9.7.1 循环步进

为了保证循环的正常（不溢出），循环变量应该选择整数的起始和步进。另外，一个类似的情况是，使用诸如 `1:1:200` 表达式赋值时，其起始、终止以及步进都需要是确定的，而不应该是变量。例如，下边的代码在一些版本上就是错误的（在新版本上没有问题了，但是用户手册中也提醒使用整数的循环计数）：

```
1 a=3;
2 t=1:1/a:15/a;
```

我们完全可以改成：

```
1 a=3;
2 t=(1:15)/a;
```

### 9.7.2 varargin 和 varargout

`varargin` 和 `varargout` 供我们在写不确定输入输出的函数时使用。通过 `varargin{i}` 来读取第几个输入变量，用 `varargout{i}` 来给第几个输出变量赋值。注意取第几个参数的时候使用的是花括号而不是小括号。

但是代码转换中的最外层的函数不支持这样的用法，也就是需要转换的函数要有明确的输入和输出。内部的函数支持使用表达式 `varargin` 和 `varargout`。下边的代码可以进行正确的转换。

```
1 %#codegen
2 function [cmlen,cmwth,cmhgt] = foo(inlen,inwth,inhgt)
3 [cmlen,cmwth,cmhgt] = inch_2_cm(inlen,inwth,inhgt);
4
5 function varargout = inch_2_cm(varargin)
6 for i = 1:length(varargin)
7     varargout{i} = varargin{i} * 2.54;
8 end
```

### 9.7.3 将不支持的函数变为“支持的函数”

Coder 支持的 MATLAB 函数越来越多，但比起 MATLAB 所有的函数来说，还是很有限的。所以当我们想用一些“不支持的函数”时就会遇到问题。我们分析一下 MATLAB 不支持的函数的原因，可能是因为 MATLAB 函数考虑的太多的，对于不同的输入格式，MATLAB 总能找到一种对应的方法，那么函数本身就会非常复杂。而往往我们只用到程序的一点点功能，这就给从

“不支持”到“支持”转换创造了条件。以 smooth 函数为例，在 R2011a 里边，smooth.m 文件共有 679 行，其中有 71 行的 help，可见这个函数的复杂性。但我们只用到其中一种最基本的功能，而且输入的信号也是一个确定长度的一维向量，就是调用如下函数形式：

```
1 smooth_sig = smooth(sig_w,50);
```

在 MATLAB 的编辑窗口中，设置断点，进入到函数体内部，单步执行。会发现针对我们这种应用都调用了些什么命令，把这些命令单独的贴出来，构成一个新的文件，就会变的非常简单 MATLAB 函数。在这个例子中，我单步执行后，程序仅仅走过了 12 行就走完了整个函数，而且每行都是支持的函数。接下来就是改个函数名，就可以转换成 C 了。

#### 9.7.4 排除函数

排除函数（Extrinsic Functions）是用在一些 MATLAB Coder 不支持的函数。有些需要转换的文件，我们经常进行就修改和测试，但其中的一些函数不能被转换和也不需要被转换，就需要使用 coder.extrinsic 函数，进行标注，这样在 MATLAB 环境下也能够运行，在转换时也不会报错，另外如果转换成 MEX 文件的时候可能有用（未测试过）。使用方式如下：

```
1 coder.extrinsic('patch');  
2 a = 1; b = 1;  
3 x = [0;a;a];  
4 y = [0;0;b];  
5 patch(x, y, 'r');  
6 axis('equal');
```

#### 9.7.5 空拷贝函数

为了避免过多的赋值，可以使用 coder.nullcopy(A) 函数，他的作用是声明未初始化的变量，只是复制 A 的类型、尺寸、是否为复数，不复制每个元素的值。使用方式如 `x = coder.nullcopy(zeros(1,N));`。

#### 9.7.6 一个低级的错误

这是一个题外的错误，但是困扰了我好几天的错误。在我建工程进行测试时，函数在函数体中和返回的结果不一样。无论是在 Windows 下还是在 Linux 下都有这种情况：

```
result_inside=-7737.500000  
result_outside=0.000000  
result_inside=-7737.500000  
result_outside=27.000000
```

原因是没有将头文件 include 进去。编译和链接的时候并没有报错，很难发现问题，所以一定要细心。

## Chapter 10

# MATLAB Compiler —— 生成 dll 与 exe

MATLAB Compiler 的作用是将 MATLAB 程序转成应用或库文件发布给未安装 MATLAB 的用户。

针对不同版本的 MATLAB Compiler, MathWorks 公司都会有一本 User's Guide 用户, 可以到官方网站上下载。

既然 User's Guide 给出了那么详细的解释, 本章是不是就没有意义了? 非也, 本章介绍常见的概念和一些常用的功能, 介绍如何从一个 M 文件生成一个可执行文件或者库文件, 让大家对整个流程有个理解, 对初学者是有益的。而当你有更深层的问题或需要, 则应去查 User's Guide, 做深入的探究。

### 10.1 一些基本概念

mcc 正是 MATLAB Compiler 的缩写, 使用 mcc 命令来调用 MATLAB Compiler。或者在命令窗口中输入 deploytool 命令使用 MATLAB Compiler 的 GUI 界面。

MCR 是 MATLAB Component Runtime 的缩写。发布的应用或者库都需要包含 MCR 作为支持。提供 MATLAB 语言运行的完整支持。

MEX 文件是 MATLAB executable file 的缩写。它由 C 或 Fortran 语言编写的源代码, 经 MATLAB 编译器处理而生成的同名二进制文件, 扩展名为 mexw32、mexw64 等。可以像使用 MATLAB 函数一样来使用 MEX 文件。

一个有 MATLAB Compiler 生成的应用或者库有两部分: 特定平台的二进制文件 (binary file) 和包含 MATLAB 函数和数据的存档文件 (archive file)。

包装文件 (wrapper file) 给已编译的 M 代码提供接口, 根据可执行环境变得不同变化。主要功能: 初始和终止接口、定义数据数组、将接口函数调用转

向对 MCR 中 MATLAB 函数的调用、保存应用文件的 main 函数、保存库文件的 M 函数入口指针等。

CTF 是 Component Technology File 的缩写。和最终产生的应用或者库文件是独立的，但是和操作系统平台有关系。这种文件以 .ctf 为扩展名，包含定义应用和库的 MATLAB 函数和数据。

在使用 MATLAB Compiler 之前，需要使用 `mbuild -setup` 命令来配置和 MATLAB Compiler 一起工作的 C/C++ 编译器。不同版本的 MATLAB Compiler 支持的第三方 C/C++ 编译器不同，可以去 MathWorks 的官网上查询。

`mbuild` 命令用来选择默认的编译器和连接器、改变编译器或者编译器设置、创建自己的应用。使用 `mcc` 命令会自动调用 `mbuild` 命令。

使用一句 `mcc -m foo.m` 命令就能得到可执行文件。但是这一过程又包括以下的几个步骤：①.独立性分析，列出函数的调用关系；②.生成包装代码，将所有需要的源代码生成为目标组件；③.创建存档文件，生成 CTF 文件；④.生成指定目标的包装代码；⑤.调用第三方编译器，产生二进制软件组件。

MATLAB Compiler 用来独立产生动态链接库和可执行文件，同时是 MATLAB Builder EX/JA/NE 的内核来产生 java, web, com, .NET 等语言或组件。

可以通过 Deployment Tool 的设置来选择任何选项 `mcc` 的命令，同时 Deployment Tool 还有其他一些优点：直观方便，可保存项目的信息，可以打包发布等。

Windows 独立应用和控制台独立应用的区别：

`loadlibrary` 用来加载库文件。

## 10.2 创建独立应用的基本步骤

- a. 安装 MATLAB Compiler 和第三方编译器；
- b. 使用 `mbuild -setup` 命令，连接 MATLAB Compiler 和第三方编译器，注意中间有个空格；
- c. 创建可部署的 MATLAB 函数，脚本不能被部署；
- d. 在命令窗中输入 `deploytool`，回车进入 Deployment Project 界面；
- e. 输入新建项目的名称和位置，选择部署类型 Windows Standalone Application 或 Console Application，点击 OK；
- f. 在 Build 标签页下的主函数（Main File），在 Resources 中添加其他资源文件和帮助文件。
- g. 点击设置键（齿轮状）或从菜单栏中 Project → Settings... 中设置各种属性；这里要特别说一下其中 General 标签页中的 Intermediate 和 Output 两个文件路径，根据名字可以看出这两个路径分别是存放中间文件和输出文件的。

h. 点击 Build 键或者从菜单栏中 Project → Build, 对所选的文件进行转换。

i. 点击 Package 选项卡, 可执行文件和 readme.txt 会自动加入到打包的选项中, 另外需要选择的是是否将 MCR 一并打包进去, 还可以选择从网上下载 MCR 安装包, 但是需要输入网址。如有需要, 可以双击 readme.txt 进行修改。

j. 设置完需要打包的东西之后, 点击打包键 (纸盒状) 或者从菜单栏中 Project → Package, 然后会提示打包后可执行文件要存放的位置, 对所选的文件进行打包, 得到可执行文件。

k. 在没有安装 MATLAB 的计算机上, 新建一个安装目录, 将打包生成的可执行文件转移该目录下, 双击就会解压出独立的应用程序。

l. 安装 MCR: 如果已经将 MCR 打包进 package, 则在解压完成后就会自动安装; 如果没有将 MCR 一并打包, 则需要将相应的 MCR 安装文件。

m. 至此, 已完成所有转换和安装的步骤, 用户可以使用独立的应用程序。

### 10.3 创建 C/C++ 共享库

创建 C/C++ 共享库的过程与创建独立的应用程序类似, 首先都需要安装 MATLAB Compiler 和第三方编译器, 以及使用 `mbuild -setup` 连接 MATLAB Compiler 和第三方编译器, 其他步骤如下:

a. 编写的 MATLAB function 文件, 这些文件正式创建的 C/C++ 共享库中所包含的函数;

b. 在命令窗中输入 `deploytool`, 回车进入 Deployment Project 界面;

c. 输入新建项目的名称和位置, 选择部署类型 C Shared Library 或 C++ Shared Library, 点击 OK;

e. 在 Build 标签页下的导出函数 (Export Functions) 中添加需要到处的 MATLAB 函数, 在 Resources 中添加其他资源文件和帮助文件; 注意有一些 MATLAB 函数也需要放在资源文件中, 而不是放导出文件中。

f. 点击设置键 (齿轮状) 或从菜单栏中 Project → Settings... 中设置各种属性;

接下来的创建 (Build) 和打包 (Package) 步骤与创建独立应用类似。同样会在输出文件夹得到相应的文件。其实打包就是一个压缩的过程, 将文件压缩后变成一个 exe 文件, 及时电脑上没有压缩软件, 也可以解压缩。

得到的四个输出文件包括: `readme.txt`、`.h file`、`.dll file`、`.lib file`。

.h 文件是库文件头文件, 调用动态链接库的程序需要 include 该文件, 这里面有导出函数的声明。

.dll 文件是二进制动态链接库文件, 这里的 dll 只是 Windows 系统中的扩展名, 在其他操作系统平台上可能是其他的名字。

.lib 文件是导入库 (Import Library)。目前以 .lib 后缀的库有两种, 一种为静态链接库, 另一种为动态链接库的导入库, 这里生成的 .lib 文件是后者。方便程序静态载入动态链接库, 否则你可能就需要自己 LoadLibrary 调入 DLL 文件, 然后再手工 GetProcAddress 获得对应函数了。有了导入库, 你只需在链接导入库后, 按照头文件函数接口的声明调用函数就可以了。

## 10.4 调用动态链接库

本小节介绍调用的生成的动态链接库, 不按部就班, 而是探索性的进行, 遇到问题解决问题。测试环境如下:

操作系统	Win7
MATLAB	R2013a
M Compiler	Version 4.18.1
第三方编译器	VS 10.0

我们写一个函数 `dlltest`, 作为转换成动态链接库的目标。函数的目的是为了求一个矩阵的最大值, 作用与 `max` 作用类似, 后边加了一个显示的选项, 如果显示选项为字符 'y', 则打印出最大值。当然, 我们可以把许多函数都放到一个 `dll` 中, 这里只是为了简洁。

```
1 function y=MyMax(x,dim,disp_opt)
2 y=max(x,[],dim);
3 if disp_opt=='y'
4     disp(['The maximum number is ' num2str(y)]);
5 end
```

使用 Deploy tool, 为了区分函数和动态链接库, 将项目名称改成 `dlltest`, 得到四个文件: `readme.txt`, `dlltest.dll`, `dlltest.lib`, `dlltest.h`。

在 VS 中新建一个 Visual C++ 空项目, 将上述四个文件复制到工程目录下。添加头文件 `dlltest.h`。双击打开我们会看到几个函数的定义, 我们重点关注三个函数: `dlltestInitialize(void)`, `dlltestTerminate(void)`, `MyMax(int nargout, mxArray& y, const mxArray& x, const mxArray& dim, const mxArray& disp_opt)`, 这三个功能分别是初始化动态链接库、结束该动态链接库、使用 `MyMax` 函数。

分析一下 `MyMax` 函数中的变量, 第一个 `nargout` 表示输出变量的个数, 第二个 `y` 用来存放输出, 而之后的三个参数为函数的输入, 由于不应该改变输入变量的内容, 所以输入变量是 `const` 的。关于 `mxArray` 的定义稍后在做介绍。

这里先写一个测试的 `main.cpp`, 文件如下:

```
1 #include <iostream>
```



```

2  #include "dlltest.h"
3  using namespace std;
4
5  int main()
6  {
7      //Initialize the Dynamic Link Library(dll)
8      dlltestInitialize();
9      //Defined the input and output argument of MyMax in mxArray
10     mxArray p_dim(1, 1, mxDOUBLE_CLASS);
11     mxArray p_x(1, 5, mxDOUBLE_CLASS);
12     mxArray p_disp_opt('y');
13     mxArray p_y(1, 1, mxDOUBLE_CLASS);
14     //Assign value to the Input;
15     double x[] = {1,2,3,4,5};
16     double dim=2;
17     p_x.SetData(x,5);
18     p_dim.SetData(&dim,1);
19     //Execute the function
20     MyMax(1, p_y, p_x, p_dim, p_disp_opt);
21     //Get the value from the result
22     double y;
23     p_y.GetData(&y,1);
24     //End the Dynamic Link Library
25     dlltestTerminate();
26
27     return 0;
28 }

```

按 F7 键，提示错误找不到 dlltest.h 文件中的一个头文件：“fatal error C1083: 无法打开包括文件: “mclmcrtr.h” : No such file or directory”。这个错误是由于我们没有设置头文件的路径，应为调用动态链接库时不仅仅是生成的三个文件，还需要 MCR 中的其他文件。我们电脑上已经安装了 MATLAB 就没有必要再去装 MCR 了，头文件在 [matlabroot '/extern/include'] 目录中，库文件在 [matlabroot '/extern/lib/win32/microsoft'] 目录中。在 VS 的项目属性页 → 配置属性 → VC++ 目录中，在“包含目录”和“库目录”中增加相应的两个目录。

增加目录后，再按 F7 重建，依然会有报错：“fatal error LNK1120: 12 个无法解析的外部命令”。该错误是由于编译时缺少相应的附加依赖项引起的。在项目属性页 → 配置属性 → 链接器 → 输入 → 附加依赖项中增加我们产生的'dlltest.lib'。

增加 dlltest.lib 后，再按 F7，依然报错：“fatal error LNK1120: 7 个无法解析的外部命令”。但是与上一个错误的不同之处在于，这次无法解析的外部命令少了 5 个，说明我们增加附加依赖项的做法是对的。

查了一下 User's Guide 中的 About mbuild and Linking 一小节中，有一

段话: Compiled applications all must link against MCLMCRRT. This shared library explicitly dynamically loads other shared libraries. You cannot change this behavior on any platform. (所有编译的应用程序都必须链接 MCLMCRRT, 这个共享库显式动态加载其他共享库, 任何平台上都要这么做。)

因此我们在附加依赖项中增加 mclmcr rt.lib, 这个函数对应的头文件也就是最初提醒目录错误时中的那个头文件。增加完后按 F7 重建, 成功。

F5 运行, 得到的结果和在 MATLAB 中执行 `y = MyMax(1:5, 2, 'y');` 命令的结果是一样, 如图 10.1。

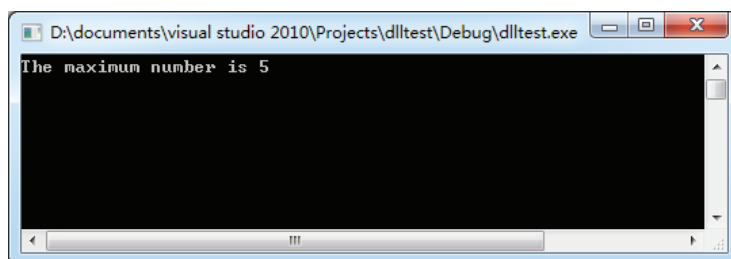


图 10.1: 在 VS 中运行测试程序的结果

用于测试的 C 程序中注释的已经很清楚了。需要注意的问题是需在定义 `mwArray` 变量之前就对动态链接库进行初始化, 因为 `mwArray` 的定义也依赖于 MATLAB 的一些动态链接库。

## 10.5 `mwArray` 类的介绍

上一小节 C++ 程序中出现了一个 `mwArray` 类, 该类用来给由 MATLAB Compiler 产生的 C++ 接口函数传递输入和输出参数。在 MATLAB R2013b 对应的 Compiler 用户手册中有 40 页来介绍这个类。本小节仅作简单介绍, 如有问题可以去用户手册查找。

所有 MATLAB 中的数据都可以表示成矩阵 (换句话说, 即便是一个简单的数据结构体也可以作为一个 `1x1` 的矩阵被声明)。而 `mwArray` 类提供必要的构造函数、方法、操作符来产生阵列、初始化阵列以及简单的索引。

使用 `mwArray` 类需要两个头文件: `mclcppclass.h` 和 `mclmcr rt.h`。在 MATLAB Compiler 生成的头文件中已经包含了这两个头文件。

首先对其构造函数做一个介绍。

```
1 mwArray()  
2 //构造 mxDOUBLE_CLASS 型空阵列  
3  
4 mwArray(mxClassID mxID)  
5 //构造指定类型的空阵列  
6
```

```

7  mxArray(mwSize num_rows, mwSize num_cols, mxClassID mxID,\
8  mxComplexity cmplx = mxREAL)
9  //构造一个二维复数阵列
10
11 mxArray(mwSize num_dims, const mwSize* dims, mxClassID mxID,\
12  mxComplexity cmplx = mxREAL)
13  //构造 n 维附属阵列
14
15 mxArray(const char* str)
16  //构造字符串
17
18 mxArray(mwSize num_strings, const char** str)
19  //构造字符矩阵
20
21 mxArray(mwSize num_rows, mwSize num_cols, int num_fields,\
22  const char** fieldnames)
23  //构造结构体矩阵
24
25 mxArray(mwSize num_dims, const mwSize* dims, int num_fields,\
26  const char** fieldnames)
27  //构造 n 结构体阵列
28
29 mxArray(const mxArray& arr)
30  //深度复制一个已经存在的 mxArray 变量
31
32 mxArray(<type> re)
33  //构造一个实数标量
34
35 mxArray(<type> re, <type> im)
36  //构造一个复数标量

```

然后，我们再说这个类对应的方法，按照功能对其进行粗略的分类：

```

1  /* 矩阵信息获得 */
2  mxClassID ClassID() const           //数据类型
3  int ElementSize() const             //元素字节数
4  mwSize NumberOfElements() const     //元素个数
5  mwSize NumberOfNonZeros() const     //非零元素个数
6  mwSize MaximumNonZeros() const      //
7  mwSize NumberOfDimensions() const   //数据维数
8  int NumberOfFields() const          //结构体数组的域个数
9  mwString GetFieldName(int index)    //结构体的指定域名
10 mxArray GetDimensions() const       //各维的尺寸
11 mxArray RowIndex() const            //返回行索引
12 mxArray ColumnIndex() const         //返回列索引

```

```

1  /* 矩阵判定的函数 */
2  bool IsEmpty() const

```

```

3 bool IsSparse() const
4 bool IsNumeric() const
5 bool IsComplex() const
6 bool Equals(const mxArray& arr) const
7 int CompareTo(const mxArray& arr) const

```

```

1 /* 其他函数 */
2 int GetHashCode() const           //根据数组内容构造唯一散列值
3 mwString ToString() const         //转换成 mwString 类型
4 void MakeComplex()                //将数值型由实数转换成复数
5 static mxArray NewSparse(...)     //产生稀疏矩阵
6 mxArray Clone() const             //深度复制一个mxArray
7 mxArray SharedCopy() const        //复制后指向共享同一块区域的数据
8 mxArray Serialize() const         //序列化成字节数组 (1xn 的
9                                   //mxUINT8_CLASS 矩阵)
10 static mxArray Deserialize(const mxArray& arr)
11                                   //Serialize 函数的逆函数

```

```

1 /* 读写指定的元素 */
2 mxArray Get(mwSize num_indices, ...)
3 mxArray Get(const char* name, mwSize num_indices, ...)
4 mxArray Get(mwSize num_indices, const mwIndex* index)
5 mxArray Get(const char* name, mwSize num_indices, \
6             const mwIndex* index)
7 void Set(const mxArray& arr)

```

```

1 /* 取实部和虚部 */
2 mxArray Real()
3 mxArray Imag()

```

```

1 /* 读写整个数据 */
2 void GetData(<numeric-type>* buffer, mwSize len) const
3 void SetData(<numeric-type>* buffer, mwSize len) const
4 void GetLogicalData(mxLogical* buffer, mwSize len) const
5 void SetLogicalData(mxLogical* buffer, mwSize len) const
6 void GetCharData(mxChar* buffer, mwSize len) const
7 void SetCharData(mxChar* buffer, mwSize len) const

```

```

1 /* 产生和判定特殊值 */
2 static double GetNaN()
3 static double GetEps() //浮点相对误差限
4 static double GetInf()
5 static bool IsFinite(double x)
6 static bool IsInf(double x)
7 static bool IsNaN(double x)

```

```
1  /* 操作符 */
2  mxArray operator() (mwIndex i1, mwIndex i2, mwIndex, i3, ..., )
3  mxArray operator() (const char* name, mwIndex i1, mwIndex i2, ...,)
4  mxArray& operator=(const <type>& x)
5  operator <type>() const
```

## 10.6 注意的问题

MCR 的版本问题：MCR 不是向后（向下）兼容的，所以生成独立应用程序和 C/C++ 共享链接库的 MATLAB Compiler 和 MCR 需要一致；如果使用新版本 MCR 则需要用对应版本的 Compiler 进行重新编译。如果不重新编译，使用动态链接库或独立应用程序的时候，都会报错。

MATLAB 版本的位数应该和第三方编译器的位数一致，若不一致，虽然使用 `mbuild -setup` 能够连接两个编译器，但是在转换时会出错（或许需要别的配置）。

**不要创建和使用非常量静态状态变量：**避免在 MATLAB 代码中使用全局变量（global variables）；避免在 MEX 文件中使用静态变量；避免在 Java 代码中使用静态变量（是指使用 JA Builder 的时候）。

## 10.7 mex, mcc, mbuild

MEX 作为 MATLAB 中的命令时，作用是将  
mcc 是 MATLAB Compiler

## Chapter 11

# 人机交互

本章介绍人机交互编程的一些小技巧，不是介绍 GUI。

### 11.1 一些 UI 开头的

UI是用户界面（User Interface）的缩写，以UI开头的一些命令都是使用用户界面让用户完成一些操作。

比如让用户选择打开文件或获取文件路径的若干命令：uigetdir，选择路径；uigetfile，获得文件路径和文件名；uiopen，使用默认程序打开文件；uiputfile，覆盖保存文件；uisave，保存数据空间到指定位置；uiload，加载数据文件；uiimport，导入数据文件。

这几个命令的使用方式是类似的，以 uigetfile 为例介绍一下。

```
1 [filename, pathname, filterindex]=uigetfile({'*.m','MATLAB ...  
    files';...  
2     '*.mat','MAT-files';...  
3     '.*','All files' }, '选择文件', ...  
4     'C:/Users/Panzhongxian/Desktop');
```

命令的参数包括选择打开的文件格式、对话框的名字和默认路径。文件类型用两列的 cell 来表示，第一列为通配符表达式，用于过滤文件，第二列为对应通配符的说明。默认路径只是打开对话框时的位置，可以自己再选择，设置默认只是为了方便，也可以填到默认的文件名字。当我们执行这条命令后，产生如下对话框。

该命令的有三个返回值，分别是文件名（filename）、路径名（pathname）和类型标签（filterindex）。通过 fullfile 命令可以将 filename 和 pathname 组合成绝对路径，以便后续进行处理。

这里的 uiputfile 函数也是用来获得已有文件的文件名和路径的，和 uigetfile 命令不同的是它会提示将要选定的文件进行覆盖操作，并询问是

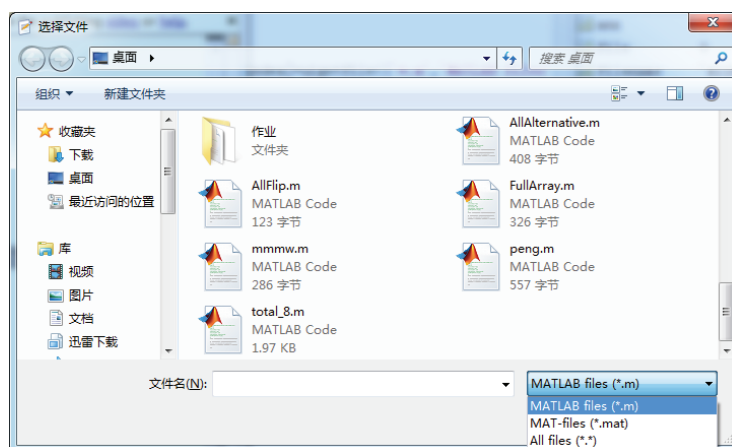


图 11.1: uigetfile 函数打开的标准窗口

否确定。但选择了确定也并不会覆盖文件，因为覆盖是通过后续的文件操作进行的。

除了 ui 开头的文件操作之外，还有其他很多 ui 开头的命令，如让用户选择字体的 `uifont` 命令、选择颜色的 `uicolor` 和 `uicolormap` 命令等。还有一些 ui 开头的命令专门用在 GUI 编程的，这里就不介绍了。

## 11.2 一些以dlg结尾的

dlg 是对话框 (dialog box) 的缩写。dialog 本身是个函数用来根据属性产生空对话框。下边将要提到的若干对话框属性都可以通过 `get(h)` 的方式获得，并进行修改。

对话框的作用多种多样，首先介绍提示信息的对话框：`errordlg`，错误信息提示；`warndlg`，警告信息提示；`helpdlg`，帮助信息提示。这些对话框可以看成信息对话框的特例，都可以通过 `msgbox` 实现。

```
1 errordlg({'Error dialog box test', 'Generated by ...'}, 'Error1');
2 msgbox({'Error dialog box test', 'Generated by ...'}, 'Error2', 'error');
```

还有一种对话框是给出提示信息，让用户进行选择的。比如 `questdlg` 函数，其返回值对应用户的选择。

```
1 choice = questdlg('选择何种识别方式?', '提示信息', ...
2   '模拟信号识别', '数字信号识别', '数字信号识别');
```

各个字符串分别是提示内容、对话框名、按键名和最后的默认按键位置，我们点击其中的一个按键后，choice 得到的就是按键名称。另外可以给一个

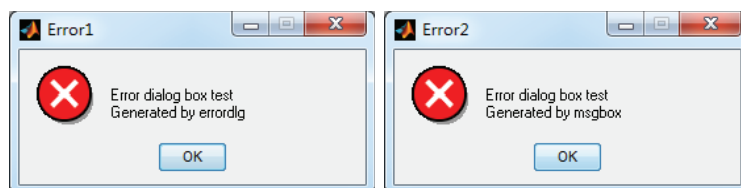


图 11.2: 使用 errordlg 和 msgbox 产生提示信息

option 的结构体赋值，如下边的一段代码出自 MATLAB 的 help，结构体中的域名就是属性名，值就是对应的设置。

```
1 options.Interpreter = 'tex';
2 options.Default = 'Don't know';
3 qstring = 'Is \Sigma(\alpha - \beta) < 0?';
4 choice = questdlg(qstring, 'Boundary Condition', ...
5     'Yes', 'No', 'Don't know', options)
```

另外，函数 menu 的作用和 questdlg 函数的作用类似。区别在于 questdlg 适用于选项不太多的问题，返回的是按键的名称（字符串）；menu 则适用于选项较多的问题，返回的是选项对应的序号（数值）。

```
1 choice = menu('选择何种识别方式?', '模拟信号识别', '数字信号识别');
```

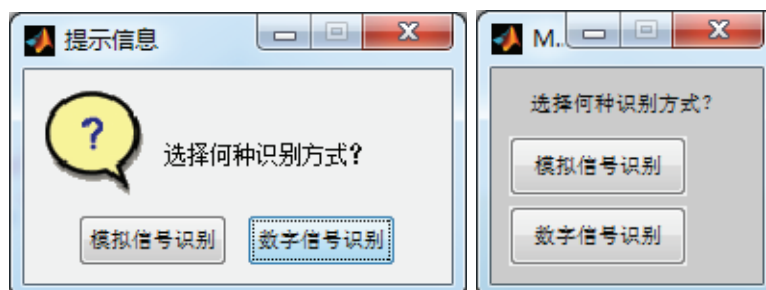


图 11.3: inputdlg 和 questdlg 创建的对话框

还有一种对话框是提示用户输入信息的，这些信息往往不是可以使用按钮列举完的。这时，我们就用到了 inputdlg 函数。

```
1 prompt = {'国家: ', '省份: '};
2 dlg_title = '请输入所在地区';
3 def = {'中华人民共和国', '陕西省'};
4 num_lines = 1;
5 answer = inputdlg(prompt, dlg_title, num_lines, def);
```

inputdlg 函数产生的对话框如图 11.4 所示。



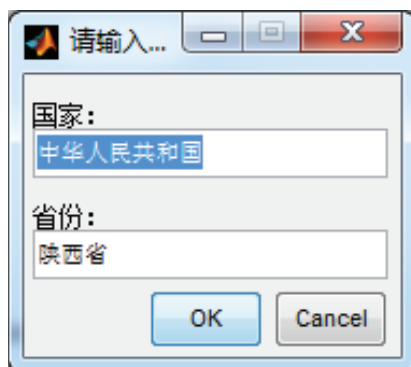


图 11.4: inputdlg 创建的输入对话框

### 11.3 waitbar

进度条 (waitbar) 用来告诉用户程序运行到什么程度, 当我们进行大量的蒙特卡洛实验的时候可以使用进度条。进度条的“进度”是我们在程序中给出的百分比, 而不是 MATLAB 自动判断。比如我们需要进行 1000 次相似的实验, 每次实验所需要的时间都相近, 这是我们就可以采用进度条。(或者说这种情况下进度条是匀速前进的, 如果进度条不能匀速的前进, 是不是就失去了进度条的意义了呢?) 每进行一次实验, 就完成 1/1000 的进度, 每次循环都更新进度然后赋给进度条即可。

```
1 h = waitbar(0, '', 'Name', '正在进行仿真实验...');  
2 steps = 1000;  
3 for step = 1:steps  
4     waitbar(step/steps, h, [sprintf('%3.1f', step/10), '%']);  
5     pause(0.01); % computations take place here  
6     waitbar(step / steps)  
7 end  
8 close(h)
```

以上的程序, 不仅以进度条表示程序执行情况, 还计算了进度的百分比, 以文字形式呈现, 如图 11.5。

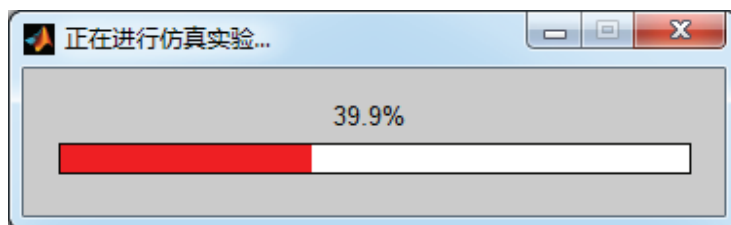


图 11.5: waitbar 创建的进度条窗口

什么时候需要而且适合使用进度条？长时间仿真，如果程序的运行时间甚至还没有创建和关闭进度条的时间长，那就肯定没必要了；时间可以估计的，如果上边我们的程序每次循环中运行时间不相同，那么很可能造成每个 1% 的“进度”时间不同，那么这就不能反应进度了。

## 11.4 figure 和命令窗中的交互

介绍两个特别简单的命令 `gtext` 和 `ginput` 函数，其中的 `g` 表示 Graphical。`gtext` 是让用户选择位置，插上 `text`。而 `ginput` 则是通过用户选择输入图中的若干个点的。

```
1 figure;  
2 plot(1:5);  
3 gtext('指定位置加注释');
```

```
1 plot(rand(1,5));  
2 [x,y]=ginput(5)  
3 plot(x,y);
```

选择方式就会在坐标系中出现一个十字坐标，如图 11.6。

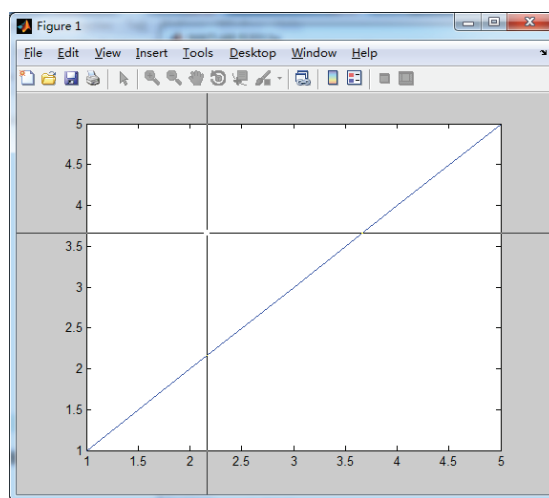


图 11.6: 使用 `gtext` 和 `ginput` 选取坐标点

另外，大部分弹出来的窗口，大都可以在命令窗进行实现。弹出的对话框和提示信息形式往往是给对 MATLAB 不熟悉的用户进行交互的；而对于常用者来说，许多交互都可以在命令窗口中完成。

比如提示输入的语句可以用 `A=input('提示信息', 格式)` 的方式。进度也可以在命令窗中 `disp` 百分比来表示。提示信息可以用之前介绍的 `error`, `warning` 等函数来显示。

## Chapter 12

# 简单的网络应用

这一章主要介绍简单的网络应用 URL、FTP 和 Email，这章纯属是娱乐性质的。MATLAB 中还有专门的数据库工具箱（Database toolbox），讲的太深，有兴趣可以自己研究一下。按照应用顺序，介绍几个函数就行了。

### 12.1 urlread 函数

本小节主要介绍 urlread 函数，最后顺带提一下 urlwrite 和 web 函数。

研究生成绩不让公布，但由于一些原因，必须查一下别人的成绩。被逼着找到学校教务处的网站的一个漏洞：

- 通过修改网址可以查到自己不同科目的成绩；
- 通过修改网址可以查到别人的成绩；
- 如果忘记密码可以通过两门课程的成绩找回自己的密码；
- 每个人都会必修两门体育课。

之所以说是一个漏洞，是因为单独看其中某一条可能都不算个漏洞，最多泄露一点点信息，可是当这四条组合到一起的时候，就很严重了。你可以根据查你成绩的网址找到你这一届几乎每一个同学的密码。知道密码之后，最少能得到的信息有：学号、密码、专业、导师、所有科目成绩、身份证号、邮箱、电话、籍贯等信息。一些认真的同学在入学的时候，还填入了更多的信息。这些信息泄露算不算很严重呢？有的人的密码可能就是他的银行卡密码，有的人喜欢用出生日期做密码，不敢再继续往下想了。

批量获得密码的思路是：通过遍历对每个学生进行操作；已知两门体育课的课程编号，查询两门体育课成绩，同时获得学生姓名和学号；将学号、课程编号、体育课成绩填入表格中；获取密码。

准备工作只需要知道某届学生查询成绩的网址即可。 `http://**/queryDegreeScoreAction.do?studentid=xdleess20120514sn0001&degreecourseno=0022001`

工欲善其事，必先利其器，网页编程的基础知识还是需要了解一点的。查询成绩网站中以 “`action?variable1=value1&variable2=value2...`” 表示的方法是用来从服务器上获得数据的 Get 方法。实际网址中，动作是查询成绩动作，变量 1 和变量 2 分别是学生编号和课程的编号。

而在我们取回密码过程中，填表并点击提交按钮的动作，是将表单中的数据放在表单的数据体中，按照变量和值相对应的方式，传递到 action 所指向 URL 的方法为 Post 方法。

因为不太了解网页编程，直接读 HTML 代码困难且没必要，所以在 Firefox 浏览器中安装插件 Web Developer extension，针对一个学生将上述过程走一遍，记录整个过程中的动作。第一步使用 Get 查询带有两个变量的，如图 12.1。

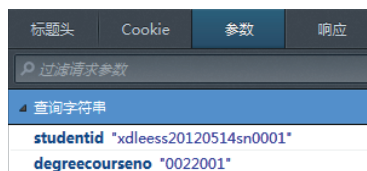


图 12.1: 使用 Firefox 查看 Get 的变量

第二步将信息填入表单使用 Post 方法，如图 12.2。

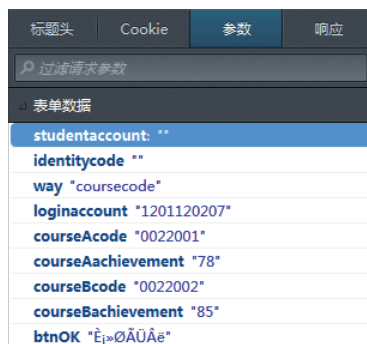


图 12.2: 使用 Firefox 查看 Post 的表单

MATLAB 中有一个 `urlread` 函数可以将 URL 的内容读到本地的字符串中，这个函数可以 Get 也可以 Post，需要的是我们把要查询或者推送的变量都定义好。

```
1 param1={'studentid',[ 'xdleess20120514sn',SN], 'degreecourseno',CN1};
2 %SN Series Number; CN1 Course Number 1
3 str1 = urlread([URL,'/',actDegree], 'Get', param1);
```

```
4 %actDegreeQueryDegreeScoreAction
```

str1 中就包含了体育课的成绩，我们再通过字符串的比对，确定姓名、学号、成绩的位置，然后将几个变量提取出来即可。当所有的变量都提取完毕后，就需要 Post 表单，获取密码了。

```
1 param3={'studentaccount','','identitycode','','way','coursecode',...
2         'loginaccount',STU_ID,'courseAcode','0022001',...
3         'courseAAchievement',PE1,'courseBcode','0022002',...
4         'courseBAchievement',PE2,'btnOK','取回密码'};
5 % STU_ID — Student ID Number; PE1 & PE2 — Scores of 2 Courses;
6 str3=urlread([URL, '/'], 'Post', param3);
7 % actPswd
```

str3 中包含了用户的密码。在最外层加一个 for 循环就可以得到所有学生的密码。

确实有学生的并不是两门成绩都有，这是返回 PE 值就可能是错误的，将错误的值填入到表中再提交必然会报错，从而中断程序。对于这个问题，我们只需要使用 try..catch.. 函数即可。找回 3000 学生密码的时间总共需要 8 分钟。结果如图 12.3，为使防止信息泄露，此处只保留姓和密码的前三位。

Command Window		
1201120001	刘×	密码: min*****
1201120002	任×	密码: 120*****
1201120003	江×	密码: 120*****
1201120004	高×	密码: 112****
1201120005	马×	密码: 689***
1201120006	李×	密码: 120*****
1201120007	王×	密码: 120*****
1201120008	张×	密码: 120*****
1201120009	宋×	密码: 120*****
1201120010	陈×	密码: 120*****
1201120011	刘×	密码: 899***
1201120012	谢×	密码: 1vd*****
1201120013	周×	密码: 888****
1201120014	赵×	密码: 615***
1201120015	杨×	密码: 107****

图 12.3: 批量取回的账号和密码

除了 Get 和 Post 外，还经常设置编码字集（'Charset'）和超时时间（'Timeout'）。

与这一任务里边的关键的 urlread 函数对应的另外一个函数是 urlwrite 函数。urlwrite 函数的使用方式和 urlread 函数非常相似，却别在于前者将读取的 URL 内容保存于一个文件中并返回文件的绝对地址，而后者将读取的 URL 内容作为返回值保存在字符串中。

web 函数是使用 MATLAB 自带的 Web 浏览器打开 URL 链接，也可以通过选项控制使用系统的浏览器。当然也可以打开本地文件，也可以将 HTML 的字符串显示出来。

```
1 url = 'http://www.baidu.com';
2 web(url, '-browser')           % 使用系统浏览器打开 URL
3 web(['file://',htmlFile])      % htmlFile 需要绝对路径
4 web(['text://', '<html><h1>Hello World</h1></html>']);
```

## 12.2 sendmail 函数

初次使用 Outlook 或者 Foxmail 时，我们需要进行一系列的配置。同样，如果想使用 MATLAB 发送邮件，也需要设置好这些参数。

```
1 %% 账号和密码
2 myaddress = 'panzhongxian@126.com';
3 mypassword = 'abcdefg';
4
5 %% 设置邮件服务器，用户名和密码，具体不详
6 setpref('Internet', 'E_mail', myaddress);
7 setpref('Internet', 'SMTP_Server', 'smtp.126.com');
8 setpref('Internet', 'SMTP_Username', myaddress);
9 setpref('Internet', 'SMTP_Password', mypassword);
10
11 %% 建 java 对象设置系统的网络和端口属性
12 props = java.lang.System.getProperties();
13 props.setProperty('mail.smtp.auth', 'true');
14 props.setProperty('mail.smtp.socketFactory.class', ...
15     'javax.net.ssl.SSLSocketFactory');
16 props.setProperty('mail.smtp.socketFactory.port', '465');
17
18 %% 关键部分，发送 Email 给自己，当然也可以发给别人
19 sendmail(myaddress, 'Mail Test', 'This is a test message.', ...
20     {'attach1.m', 'attach2.m'});
```

如果你觉着每次贴这么一段代码都非常麻烦，那就自己贴到一个函数中吧，留好接口就行啦。如果你怕这个函数被别人打开偷走你的密码，也很简单，用 pcode 加密一下就可以啦。据说还有人各种设置，通过 sendmail 发出飞信，我有空一定也要试试。

## 12.3 ftp 函数

ftp 是不是不太多了？现在大家都在学校 BT 网站上下载各种资源，好像 FTP 不是很火了，不过看到很多人在用，比如学校开源社区的 FTP 服务器上

还是有很多资源的。

ftp 函数是一类函数，用法非常简单。对 FTP 服务器上的文件进行操作的过程：首先使用 ftp 连接到 FTP 服务器并建立一个对象，使用 cd、dir、mget、mput、delete、mkdir、rmdir、rename 等函数对文件进行操作，最后使用 close 关闭连接。

如果在 FTP 服务器上有用户名和密码，可以写进去；如果没有，可以直接输入 FTP 的地址。文件操作跟使用的普通命令的区别就是多了一个 FTP 对象作为输入变量。

```
1 mw=ftp('ftp.xdlinux.info');
2 % connect ftp and create an object
3 cd(mw, 'books');
4 % change directory
5 dir(mw);
6 % display the contents of current folder
7 mget(mw, 'Archlinux.pdf', '..');
8 % save the pdf to the directory above
9 close(mw);
10 % close the connect
```

## 12.4 教务处网站的修复

让人很欣慰的是，14 年 03 月 06 日把这个漏洞 Email 到教务处之后，三天之内就把网站改了，看看修改前后的变化吧。

The image shows two screenshots of a web application's password recovery page, illustrating a security fix. The top screenshot shows the original interface with two radio button options. The first option, '按学生帐号, 身份证号' (By student account, ID card number), has fields for '学生帐号:' (Student account) and '身份证号:' (ID card number). The second option, '按登陆帐号, 两门课程的编号及成绩' (By login account, two course numbers and scores), has fields for '登录帐号:' (Login account), '课程A编号:' (Course A number), '课程A成绩:' (Course A score), '课程B编号:' (Course B number), and '课程B成绩:' (Course B score). Below these fields are buttons for '取回密码' (Recover password) and '返回' (Return). The bottom screenshot shows the updated interface where the second option is disabled (greyed out), leaving only the first option '按学生帐号, 身份证号' available for selection. The layout and other elements remain the same.

图 12.4: 教务处网站修改前后对比

只可惜没有在漏洞修复之前就记录下所有人的信息，哈哈，开玩笑的了。  
希望我们以后这样的漏洞越来越少，我们的信息才能安全。



## Chapter 13

# MATLAB 文件的发布

写的 MATLAB 程序，无非分为几类：自己写自己用的，给别人用也公开代码的，给别人用却不想公开代码的。自己用的文件自己管理好就 OK 啦，给别人用的就需要进行文件的发布。当然不是开发布会公告一下，而是对文件做一些处理。公开的代码，可以 `publish` 成不同格式的文档；非公开的代码，可以使用 `pcode` 对代码进行加密和保护。

### 13.1 `publish` 需要的文件格式

有良好的编程习惯的人，会在合适的地方加上必要且恰当的注释，注释良好的程序经过 `publish` 后就轻松地编程了文档。换句话说，`publish` 的重点不在使用这个函数，而在于良好的注释。借着本节再说说注释。

#### 13.1.1 `%%`, `%` 和 `%%%`

看花了眼了？在说“写自己的函数”一章介绍过注释的一些基本写法。注释中，`%%` 用于分割两个代码单元（cell），`%` 表示普通的注释，`%%%` 在普通注释中好像没有什么特别的含义，属于以 `%` 开头的内容。

而在发布 m 文件的时候，这些符号就有了一些区别。

`%%` 的用法有两种。第一种是之后接有文字，则文字会在代码发布的过程中被转换成了题目和节标题，表示开始新的一个 section。第二种是没有接文字，则作为分割代码和普通注释的符号。

`%` 的用法也有两种，这两种都体现在跟 `%%` 的关系上。第一种是紧跟在 `%%` 或 `%%%` 之后，作为描述的正文。第二种是跟在命令之后，作为代码的一部分显示在代码框中。

`%%%` 的作用同 `%%` 类似，用来开辟一个新的 section，但是在 Editor 中却不会产生一个新的代码单元。

对下边的代码进行 publish，几种符号的用途就会一目了然了。

```
1 %% Title
2 % This is a start of a document.
3 %% First Section Title
4 % This is a start of a section.
5 x=1:5;
6 % This will be published as a comment after code.
7 %%
8 % Separated with code.
9 %%% Second Section Title
10 % This is a start of a new section.
```

### 13.1.2 格式和列表

对某些文字限定格式，需要使用成对的符号来限定。成对的下划线表示斜体，在代码外解释变量通常都在变量两侧加上下划线，如 `_k_` 表示变量 `k`；成对的星号表示黑体；成对的竖线表示等间隔字体；还有两个商标符号，需要在文字之后加上 (TM) 和 (R) 来表示 TRADEMARKS 和 REGISTER。

```
1 %%
2 % _ITALIC_ *BOLD* |MONOSPACED| TRADEMARKS(TM) REGISTER(R)
```

列表分为有、无序号两种。需要注意的是，列表必须使用 %% 和其他内容分开，两种列表不能混合使用；列表符号和之后的文字之间需要加空格。

```
1 %% Bulleted List
2 % * BULLETED ITEM 1
3 % * BULLETED ITEM 2
4 %% Numbered List
5 % # NUMBERED ITEM 1
6 % # NUMBERED ITEM 2
```

### 13.1.3 插入其他代码

我们往往需要插入其他类型的代码。比如，可以插入整段的 L<sup>A</sup>T<sub>E</sub>X 语句，不过这个只有在转换成 TEX 文件的时候才有用。

```
1 %%
2 % <latex>
3 %\begin{array}{cc}
4 % 2 & 3 \\
5 % 1 & 4
6 %\end{array}
7 % </latex>
```

比如，可以插入 L<sup>A</sup>T<sub>E</sub>X 表达式，和在 L<sup>A</sup>T<sub>E</sub>X 下直接写是一样的，分为嵌入的和独立成块的，分别使用 `$` 和 `$$` 符号括起来。

```
1 %%
2 % $L(x) := \sum_{j=0}^k y_j \ell_j(x)$
3 % $$L(x) = \ell_1(x) \sum_{j=0}^k \frac{w_j}{x-x_j} y_j$$
```

比如，可以插入 HTML 代码。

```
1 %%
2 % <html>
3 % <table border=2><tr>
4 % <td>one</td>
5 % <td>two</td></tr></table>
6 % </html>
```

比如，可以插入超级链接。

```
1 %%
2 % <http://www.xidian.edu.cn Xidian>
```

比如，可以插入图片，使用双尖括号。不过需要注意的是要生成的 html 文件，则要放到同目录下的 html 文件夹下，如果要生成 pdf 文件，则要放到同目录下的 pdf 文件夹中。

```
1 %% External Graphics
2 % <<test.png>>
```

## 13.2 万事俱备，只欠 publish

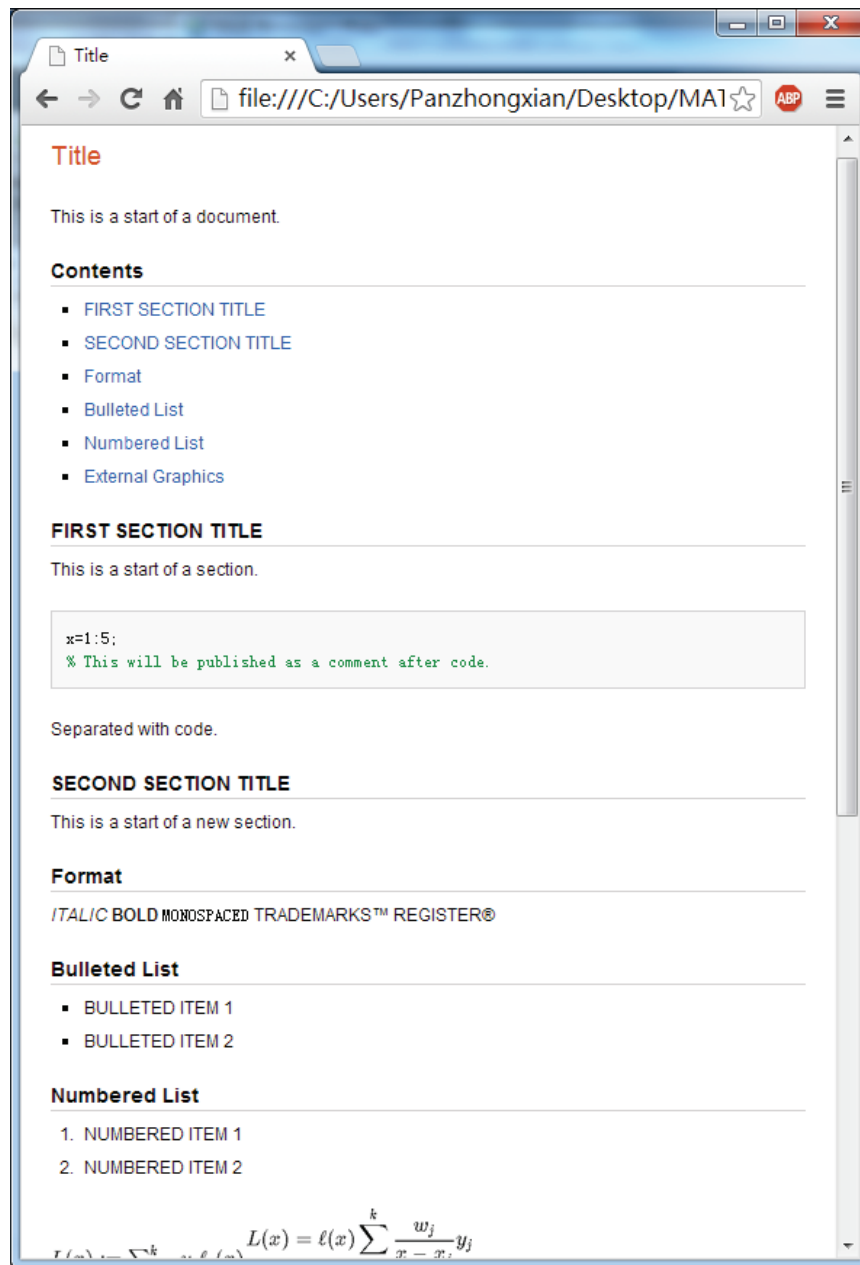
当按照上边的一些格式写好代码之后，我们就可以 publish 了。

输出的格式有 html、doc、latex、ppt、xml、pdf 等几种格式，默认的是 html 格式。各种格式基本上都试过，转出来最好看的还是 html 格式的文件。当然各类文件都有优缺点：html 里的 L<sup>A</sup>T<sub>E</sub>X 公式都是图片格式；latex 格式文件转出来后直接编译甚至会出错误；像 ppt 转出来许多图片的位置都需要人工去调整；pdf 转出之后全是黑白颜色、没有区分度，不过公式漂亮。

还有其他几类选项——输出选项、图片选项、代码选项——可以设置，各类又有许多细节的参数可以设置。

敲参数太麻烦？当然有 GUI 界面给你用啦！其实上边介绍的格式、列表、插入其他代码等内容都被做到了 GUI 界面中了。打开 Editor 里边的 Publish 选项卡，各种内容都浮现了出来。Publish 的选项隐藏的还是很深，在 Publish 下拉菜单中打开 Edit Publishing Options... 即可设置所有的参数啦。

发布出来的文件如图 13.1。

图 13.1: 使用 `publish` 命令生成的 html 文件

### 13.3 pcode 的使用

pcode 函数使用方法简单，跟 publish 相比实在是太简单了。功能也很简单，只是对文件进行加密。

pcode 函数的语法和 delete 等函数语法类似，都是直接跟文件名就可以了。可以使用括号，也可以直接并列在后边。可以使用星号 (\*) 进行匹配，不能使用问号 (?)。

```
1 pcode('*.m')
2 % 加密所有函数
3 pcode('test*.m')
4 % 加密所有以 test 开头的文件
5 pcode('test1.m', 'test2.m', 'test3.m')
6 % 加密指定的文件
```

将脚本文件 pcode 之后，得到的就是脚本文件；将函数文件 pcode 之后，得到的就是函数文件。

有时候想可不可以跟使用动态链接库一样，把所有的函数放到同一个文件中。好像行不通，因为产生 .p 文件的目的是为了替换对应的 .m 文件而已，而 .m 文件就是一个文件对应着一个函数。

pcode 函数，默认产生的 .p 文件会放在当前文件夹中。可以带参数 'inplace'，表示产生的 .p 文件将和对应的 .m 文件放在同一个文件夹中。

.p 文件的加密算法改良过，不同版本 MATLAB 产生的 .p 文件可能会发生兼容性问题，R2007b 之后版本的 MATLAB 可以运行之前产生的 .p 文件，但是之前版本 MATLAB 不能运行之后产生的 .p 文件。

产生的 .p 文件不能通过 help 或者 doc 产看 .m 文件中的帮助信息。如果想告诉别人用法，需要另外写文档。

如果 .m 文件和 .p 文件在同一个目录下，调用脚本或者函数的时候，会执行 .p 文件，即便是修改了 .m 文件也只是执行 .p，或者会给个警告，提示 .p 文件没有对应的 .m 文件新。

## 未完待续的部分

限于时间关系，还有很一部分没有整理，还有数学部分（基础计算、概率统计、最优化理论、拟合分析数值计算、符号计算等）、信号分析和处理部分、通信仿真部分（信道建模、滤波器设计、同步）等几方面问题。零零散散的整理了一部分，放在了 `\end{document}` 之后就没有显示出来。希望找完工作之后，能把这些内容都整理出来。