

# 目录

## 第一部分 创作中

### 第一章 创作中

---

### 第二章 修改审阅中

---

Matlab 简介 [\[4\]](#) Matlab 的变量与矩阵 [\[8\]](#) Matlab 的判断与循环 [\[18\]](#) Matlab 的函数 [\[23\]](#) Matlab 画图 [\[27\]](#) Matlab 的程序调试及其他功能 [\[30\]](#) 二分法 [\[33\]](#) 多区间二分法 [\[35\]](#) 冒泡法 [\[37\]](#) 四阶龙格库塔法 [\[40\]](#) 中点法  
解常微分方程（组） [\[44\]](#) 本书编写规范 [\[48\]](#)

# 第一部分

创作中

# 第一章

创作中

## 第二章

修改审阅中

## Matlab 简介

**Matlab (Matrix Laboratory)** 的中文名叫矩阵实验室，是一款著名的科学计算软件，也指这个软件中使用的编程语言。这里仅介绍最基本的 Matlab 功能和语法，且仅介绍本书使用到的功能。

### 界面介绍

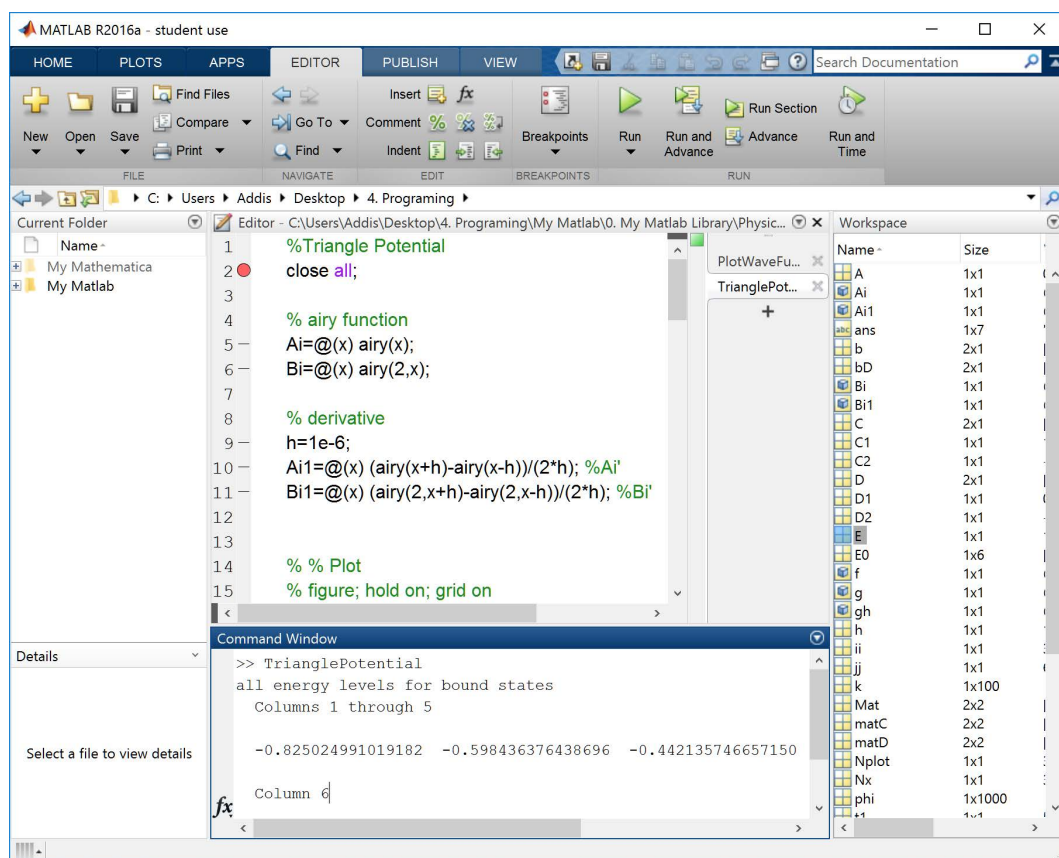


图 1: Matlab 的 IDE 界面

Matlab 的编程界面（图 1）属于集成开发环境（IDE/Integrated Development Environment），简而言之就是一切与 Matlab 编程有关的工作都可以在该界面完成<sup>1</sup>。以下介绍界面中常用的窗口。要选择显示的窗口，可在 Home 菜单中点

<sup>1</sup>界面语言默认与操作系统语言相同，本书使用英文界面。

击 Layout 按钮，并在 Show 下面勾选需要的窗口。

**Editor** 用于编辑代码，同时具有自动检测语法错误，代码调试等功能。Matlab 的代码文件分为脚本文件和函数文件两种形式，后缀名都为“.m”，用图 1 中 Editor 菜单栏的 Save 按钮可保存代码文件。Matlab 作为一种解释语言（Interpreted Language）可以直接在 Editor 中运行源代码，无需传统的编译过程。为了让 Matlab 能运行代码文件，需要把文件所在的目录<sup>2</sup>添加到 Matlab 的搜索路径下。

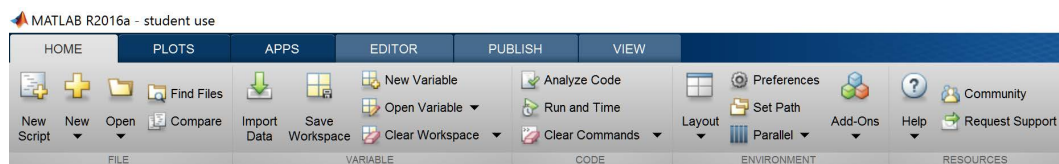


图 2: Home 菜单

如图 2，Home 菜单中的 Set Path 按钮可以设置 Matlab 的搜索路径。点开后用 Add Folder 按钮可以添加单个文件夹（不包含子文件夹），用 Add with Subfolders 添加文件夹（包含子文件夹）。用 Remove 删除已添加的路径，用 Default 还原初始设置，用 Save 保存修改，用 Close 关闭窗口。若要运行程序，回到 Editor 菜单点击 Run 按钮即可。

**Command Window** 主要用于输入临时指令或者调试程序，可输入除了函数定义外的任意指令。Command Window 只能按输入顺序执行，不方便修改和编辑，如果指令较长或有多个指令，应该使用 Editor。在 Command Window 中按回车执行输入的指令，按上箭头可重复已输入的指令。

**Workspace** 用于查看 Matlab 当前的所有变量的列表。Matlab 的所有变量都可以理解为矩阵，单个值可理解为  $1 \times 1$  的矩阵。列表中 Name 是变量名，Size 是矩阵维度，Value 是变量值，右上角的下拉菜单中的 Choose Columns 中还可设置显示更多属性，例如 Bytes 是占用字节数，Class 是变量类型，Min 是最小值，Max 是最大值，Mean 是平均值，Median 是中位数，Std 是标准差等。双击 Workspace 中的变量可显示变量值。

<sup>2</sup>在英文界面下 Matlab 不能识别中文目录，建议用英文命名文件夹。

## 计算器

下面我们仅用 Command Window 来熟悉 Matlab 的基本语法。我们先看如何把 Matlab 当做普通的科学计算器使用。

```
>> 1.2/3.4 + (5.6+7.8)*9 -1
```

```
ans = 119.9529
```

```
>> ans + 1
```

```
ans = 120.9529
```

```
>> 1/exp(1)
```

```
ans = 0.3679
```

```
>> exp(-1i*pi)+1
```

```
ans = 0
```

常用的运算符号有

+, -, \*, /, ^ (指数)

常用的数学函数有

sqrt (开方), exp, sin, cos, tan, cot, asin, acos, atan, acot, real (实部), imag (虚部), conj (共轭)

等 (三角函数前面加 a 代表其反函数)。运算的优先顺序与数学上的习惯一样。注意这些函数的自变量都可以是复数。为了区分虚数单位 i 和变量 i，好的习惯是在 i 前面加数字 (上面的第三条命令)。pi 是圆周率，注意是小写。

mod(N,n) 是求余运算，计算 N 被 n 整除后的余数。注意这个函数有两个变量，用逗号隔开。要注意在 Matlab 中，这种有输入和输出的命令都是广义的函数 (function)，不仅是数学函数。

sign(num) 函数用于求实数 num 的符号。如果 num > 0，则返回 1，若 num < 0 则返回 0，若 num = 0 则返回 0。

用大写 E 或小写 e 表示科学计数法 (不允许有空格)，如  $2.997 \times 10^8$  表示为 2.997e8 或 2.997e+8。用小写 pi 表示圆周率，用 exp(1) 表示自然对数底，用 1+2i 或 1+2j 等表示复数，注意 i 和 j 前面不能有空格。

如果需要多位小数，可使用 format long 命令使结果显示为双精度 (约 16 位有效数字)，用 format short 命令恢复默认格式。

```
>> format long; pi  
ans = 3.141592653589793  
>> exp(1)  
ans = 2.718281828459046
```



## Matlab 的变量与矩阵

预备知识 Matlab 简介<sup>[4]</sup>

### 变量与矩阵

**变量 (variable)** 可用于储存数据并通过变量名获取变量值。变量名可以由多个字母，数字和下划线组成。注意变量名区分大小写，且首字符只能是字母。用等号可以对变量赋值，被赋值的变量放在等号左边，等号右边表达式的运算结果会储存在被赋值的变量中，直到再次被赋值。

```
>> a = 1.2/3.4 + (5.6+7.8)*9 -1
a = 119.9529
>> b = atan(a + 1)
b = 1.5625
```

如果新的变量第一次被赋值，它会自动出现在 **Workspace** 窗口中。注意 **Workspace** 中的一个特殊的变量 **ans**，如果命令的输出结果没有赋值给变量，就会自动赋值给 **ans**。注意一般不要对 **ans** 赋值。另外两个特殊的变量是 **pi**（圆周率）和 **i**（虚数单位），一般也不要对他们赋值。自然对数底没有对应的变量，若要使用自然对数底，用 **exp(1)** 即可。

另外，如果在命令后面加分号（**semicolon**）“**;**”，则命令执行后不输出结果。也可以用分号把多个命令写到一行。

```
>> 1 + 1; a = ans^2
a = 4
```

用 **Editor** 编写程序时，每个命令后面都需要加分号，需要在 **Command Window** 输出时，用 **disp** 函数。

```
>> disp('something'); disp(10);
something
10
```

`clear` 命令可以清空 Workspace 中的所有变量，用 `clear <var1>,<var2> ...` 清除指定的变量（`<var1>,<var2>` 是变量名）。用 `clc` 命令可以清空 Command Window（按上箭头仍然可以查看历史命令）。

本书只涉及到 3 种变量类型（**class**）：双精度（**double**），字符（**char**）和逻辑（**logical**）。

## 双精度变量

双精度变量用于储存数值，有效数字约为 16 位（如果是复数，实部和虚部各 16 位），取值范围约为  $10^{-308}$  到  $10^{308}$ 。如无变量类型声明，所有命令中出现的常数及储存数值的变量都为 `double`。

Matlab 中的所有变量都可以理解为矩阵，单值变量（标量，`scalar`）可以理解为  $1 \times 1$  的矩阵，只有一行或一列的矩阵叫做行矢量（**row vector**）和列矢量（**column vector**）。一些简单的矩阵操作如下

```
>> a = [1,2,3]
```

```
a = 1  2  3
```

用方括号创建矩阵，用逗号分隔每行的矩阵元，行矢量中逗号可省略

```
>> a = [1 2 3]
```

```
a = 1  2  3
```

用分号分隔行

```
>> b = [1;2;3]
```

```
b =
```

```
1
```

```
2
```

```
3
```

```
>> c = [1 2 3; 2 3 4; 3 4 5]
```

```
c =
```

```
1  2  3
```

```
2  3  4
```

```
3  4  5
```

方括号还可以用来合并矩阵（注意矩阵尺寸必须合适）

```
>> d = [a;a]
```

```
d =
```

```

1  2  3
1  2  3
>> e = [a a]
e =
1  2  3  1  2  3

```

用 `size` 函数获取矩阵尺寸，用 `numel` 函数获取矩阵元个数

```

>> size(d)
ans = 2  3
>> size(d,1)
ans = 2
>> numel(d)
ans = 6

```

用 `zeros` 函数生成全零矩阵

```

>> zeros(2,3)
ans =
0  0  0
0  0  0

```

用 `zeros([2,3])` 和 `zeros(size(d))` 结果也相同。用 `ones` 可以生成全 1 矩阵，也可以乘以任意常数

```

>> ones(2,3)*5
ans =
5  5  5
5  5  5

```

用 `eye(N)` 生成  $N \times N$  的单位矩阵，用 `rand(M,N)` 生成随机矩阵，矩阵元从 0 到 1 均匀分布。用 `M:step:N` 生成等差数列（行矢量），例如

```

>> 1:2:10
ans = 1  3  5  7  9
>> 0:pi/3:pi*2
ans = 0  1.0472  2.0944  3.1416  4.1888  5.2360  6.2832
>> 10:-2:1
ans = 10  8  6  4  2

```

如果只用一个冒号，那么间隔默认为 1

```
>> 1:3  
ans = 1 2 3
```

用 `linspace(x1,x2,Nx)` 生成指定首项尾项和项数的等差数列（行矢量）

```
>> linspace(0,pi,4)  
ans = 0 1.0472 1.2566 2.0944 3.1416
```

下面介绍矩阵运算。同规格的尺寸可以进行 + 和 - 运算，矩阵和标量也可以，结果是把每个矩阵元加（减）标量；矩阵乘法 \* 既可以把常数与矩阵相乘，也可以进行数学上的矩阵乘法；矩阵的幂 “^” 相当于矩阵与自己多次相乘；“/” 可以把矩阵除以一个常数。

```
>> a = [1 2; 3 4]; b = [1 -1; 2 -2];  
>> a + b  
ans =  
2 1  
5 2  
>> a * b  
ans =  
5 -5  
11 -11
```

若要两个尺寸相同，可进行**逐个元素运算**，如

```
>> a .* b  
ans =  
1 -2  
6 -8  
>> a.^2  
ans =  
1 4  
9 16  
>> a ./ b  
ans =  
1.0000 -2.0000  
1.5000 -2.0000
```

单引号 “'” 可以使实数矩阵转置，或使复矩阵取厄米共轭。若需要对复矩阵转置，用 “.’” 即可。

```
>> c = a + 1i*b
c =
    1.0000 + 1.0000i    2.0000 - 1.0000i
    3.0000 + 2.0000i    4.0000 - 2.0000i
>> c'
ans =
    1.0000 - 1.0000i    3.0000 - 2.0000i
    2.0000 + 1.0000i    4.0000 + 2.0000i
>> c.'
ans =
    1.0000 + 1.0000i    3.0000 + 2.0000i
    2.0000 - 1.0000i    4.0000 - 2.0000i
```

使用 `fliplr` 和 `flipud` 函数可以分别把矩阵左右翻转和上下翻转，例如

```
>> fliplr(a)
ans =
     2     1
     4     3
>> flipud(a)
ans =
     3     4
     1     2
```

Matlab 自带的数学函数一般支持矩阵自变量，结果是该函数对每个矩阵元分别运算

```
>> cos(0:pi/4:pi)
ans =
    1.0000    0.7071    0.0000   -0.7071   -1.0000
```

用矢量运算可以使代码简短易懂，且提高计算效率。

用 `sum` 函数可以分别求矩阵每列或每行的和。当矩阵为行矢量或列矢量时，`sum` 对所有矩阵元求和。其他情况下，`sum` 默认对每列求和，若想对每行求和，可以在第二个输入变量中输入 2（行是矩阵的第二个维度）。例如

```
>> a = [1,2; 3,4];  
>> sum(a)  
ans = 4 6  
>> sum(a,2)  
ans =  
    3  
    7
```

用 `mean` 函数可以求平均值，使用格式与 `sum` 相同。

## 矩阵索引

矩阵索引用于表示矩阵部分矩阵元，例如

```
>> a = [1 2 3; 4 5 6; 7 8 9];  
ans =  
    1    2    3  
    4    5    6  
    7    8    9  
>> a(1,2)  
ans = 2  
>> a(2:3,1)  
ans =  
    4  
    7  
>> a(2:3,1:2)  
ans =  
    4    5  
    7    8  
>> a([2,3],[1,2])  
ans =  
    4    5  
    7    8  
>> a(:,2)  
ans =  
    2  
    5
```

```
8
>> a(1:end-1,2:3)
ans =
    2    3
    5    6
```

其中 `end` 表示某维度的最大索引。以上的格式可以归结为“在小括号中用逗号把行矢量隔开”。注意索引不仅可以用来取值，还可以放在等号左边赋值。

```
>> b = a; b(1:3) = a(2:4)
b =
    4    2    3
    7    5    6
    2    8    9
```

要求左边的矩阵元个数等于右边。唯一的例外是当右边为标量

```
b(1:3) = 0
b =
    0    2    3
    0    5    6
    0    8    9
```

我们还可以用单个索引

```
>> a(2:5)
ans = 4    7    2    5
>> a(7:end)
ans = 3    6    9
```

注意单个索引的顺序是先增加第一个维度（行标），再增加第二个维度（列标）。虽然上面都是以双精度矩阵为例，但这些索引方法适用于任何数据类型的矩阵。

## 字符变量

字符型变量一般用于控制行输出结果或对生成的图片进行标注。把  $N$  个字符放在一对单引号内，可生成  $1 \times N$  的字符类型数组。

```
>> str1 = '这是一个字符串'; str2 = 'this is a string';
```

```
>> [str1, ',', str2]
ans = '这是一个字符串, this is a string'
>> numel(str)
ans = 24
```

把双精度变类型变为字符串可以用 `num2str` 函数（注意 2 的英文读音与 “to” 相同，`num` 代表 “number”，`str` 代表字符串 “string”），通常用于与其他字符矩阵合并，如

```
>> number = 3; str = ['The number is', num2str(number), '.']
str = 'The number is 3.'
```

若要在字符串中加入英文单引号，可用两个英文单引号表示。

## 逻辑变量

逻辑变量只能具有 0 或 1 两个值，分别代表假（**false**）和真（**true**）。最常用的地方是判断语句 `if` 的后面以及获取矩阵元。逻辑算符有

`>`，`>=`（大于等于），`<`，`<=`，`==`（等于）

可用于比较双精度数组，返回逻辑型数组

```
>> L = 1 + 1 > 3
L = logical 0
```

在控制行中，输出的 logical 与 0 各占一行，这里为了节约空间将其写成一行。逻辑 “与”，“或”，“非” 算符分别为（仅用于逻辑标量）`&&`，`||`，`~`。当算符两边都为真时，与运算才能为真，若至少有一边为真，或运算就为真。非运算用于把真假互换。例如

```
>> 1 > 0 && 2 > 1
ans = logical 1
>> 1 > 0 || 2 < 1
ans = logical 1
>> ~(1 > 0)
ans = logical 0
```

注意由于非运算的优先级比 `>` 运算要高，所以最后一条命令必须要加括号。



在需要的时候，双精度变量可以自动转换为逻辑变量，规则是只有双精度的 0 转换为逻辑 0，其他双精度值一律转换为逻辑 1。如

```
>> 1.3 && -0.8
ans = logical 1
```

除了上文中介绍的索引方法外，我们还可以用相同大小逻辑数组索引任意矩阵。注意逻辑索引的输出结果是一个列矢量，以下为了节约空间我们将输出结果改为行矢量。

```
>> a = [1 2 3; 4 5 6; 7 8 9];
>> mark = logical([1 0 0; 0 0 1; 1 0 0]); a(mark)
ans = 1 7 6
```

逻辑索引常见的例子如

```
>> a(a-3 <= 1)
ans = 1 4 2 3
```

注意逻辑索引中不能使用双精度类型代替逻辑类型。

`find` 函数可以用于寻找逻辑矩阵中所有值为 1 的矩阵元的位置。如果只用一个输出变量（或不提供输出变量），函数返回单索引，如果提供两个输出变量，函数返回行标和列标索引。注意输出变量仍然是列矢量，为了节约空间以下记为行矢量。例如

```
>> a = [1, 0; 0, 1]
a =
    1    0
    0    1
>> find(a)
ans = 1 4
>> [c, r] = find(a)
c = 1 2
r = 1 2
```

注意以上的双精度矩阵 `a` 被自动转换为逻辑矩阵。

`any` 函数用于判断逻辑矩阵中的每一列是否存在任何值为“真”的矩阵元，

若有，则返回真，否则返回假。all 函数用于判断逻辑矩阵中的每一列是否所有矩阵元都为“真”，若是，返回真，否则返回假。这两个函数的使用格式与 sum 和 mean 类似，当第二个输入变量为 2 时，对每行进行操作。

```
>> a = [1 6 7; 2 7 1; 3 8 9];
>> all(a > 5)
ans = 1×3 logical array
    0    1    0
>> any(a > 7, 2)
ans = 3×1 logical array
    0
    0
    1
>> all(a(:) < 10)
ans = logical 1
```

## Matlab 的判断与循环

预备知识 Matlab 的变量与矩阵<sup>[8]</sup>

### 脚本文件

在讲解更复杂的程序结构前，我们先来看脚本文件。脚本（**script**）文件是包含若干个指令的文件，文件后缀名为“.m”。脚本文件可以单独执行，也在其他文件或 Command Window 中被调用（注意需要将所在文件夹添加到搜索路径）。后者当于把被调用脚本的代码直接插入到调用指令处，调用指令就是脚本文件的文件名。脚本中的每条命令后面应该加分号以隐藏输出结果，若需要输出，用 `disp` 函数。

```
>> disp('good'); a = 3; disp(['a = ', num2str(a)])
good
a = 3
```

在脚本文件中，可以在行首或命令后用百分号%进行注释（**comment**）<sup>3</sup>。注释是程序的说明，使程序更易读，但在执行程序时会被忽略（图 1）。

### 判断结构

现在来看一段代码（脚本文件）。要生成新的脚本文件，可以在 Editor 菜单（图 1<sup>[4]</sup>）的左边单击 New，然后选 Script，或者用快捷键 Ctrl+N。在生成的 Editor 中输入以下代码

```
1 a = rand(1,1); b = 0.5;
2 if a > b % a 较大
3     disp('a is larger');
4 else % b 较大
5     disp('b is larger');
6 end
```

<sup>3</sup>截止到 Matlab 2017b，在英文版 Matlab IDE 中，任何中文注释都会在 Matlab 重启后变为乱码。若要使用中文注释，建议使用中文操作系统和中文 Matlab。

这段程序用 `rand` 函数随机生成一个从 0 到 1 的数，如果随机数大于 0.5 则输出第一段文字，否则输出第二段文字。不难猜测出这里的 `if` 用于判断，如果条件满足，则只执行 `if` 和 `else` 之间的指令。如果条件不满足，则只执行 `else` 到 `end` 的指令。注意 `else` 语句可以不在判断结构中出现，若不出现，当判断条件不满足时程序将直接执行 `end` 后面的代码。

要执行该代码，在 Editor 菜单中单击 Run 图标（绿色三角形），如果代码没有保存，Matlab 会先弹出保存对话框。再次强调文件必须保存在 Matlab 的搜索路径下。

`elseif` 语句可用于在判断结构中产生多个分支，如

```
1 a = rand(1,1);
2 if a > 0.9
3     disp('a in (0.9, 1]');
4 elseif a > 0.6
5     disp('a in (0.6, 0.9]');
6 elseif a > 0.3
7     disp('a in (0.3, 0.6]');
8 else
9     disp('a in [0, 0.3]');
10 end
```

这个程序用于判断随机数 `a` 的区间。若 `if` 的条件判断成功，判断结构就只执行 `if` 到第一个 `elseif` 之间的命令。若 `if` 判断失败，程序就继续判断第一个 `elseif` 中的条件，若判断成功，就只执行第一个 `elseif` 到第二个 `elseif` 之间的命令，以此类推。如果 `if` 和所有的 `elseif` 条件都判断失败，则执行 `else` 后面的命令。

## 循环结构

我们先来看 `for` 循环

```
1 for ii = 1:3
2     disp(['ii^2 = ' num2str(ii^2)]);
3 end
```

运行结果为

```
ii^2 = 1
ii^2 = 4
ii^2 = 9
```

容易看出这段代码被执行了 3 次，循环变量 `ii` 按顺序取 1:3 中的一个矩阵元。注意选取 `ii` 作为变量名是为了与虚数单位区分，当然也可以选择其他变量名。再来看一个稍复杂的循环

```
1 Nx = 5;
2 x = zeros(1,Nx); % 预赋值
3 x(1) = 2;
4 for ii = 2:numel(x)
5     x(ii) = x(ii-1)^2;
6 end
7 disp(['x = ' num2str(x)])
```

在循环开始前 `x(1)` 被赋值为 2，在循环中，第 `ii` 个矩阵元依次被赋值为第 `ii-1` 个矩阵元的平方。运行结果为

```
x = 2   4   16   256  65536
```

注意在循环前用 `zero` 对矩阵进行了预赋值（**preallocation**）。预赋值不是必须的，但如果不进行预赋值，每次循环矩阵的尺寸都要改变，会导致程序运行变慢。另外注意循环中不允许给循环变量赋值。

再来看另一种循环叫做 `while` 循环。下面来看一个例程，输出 100 以内的斐波那契数列（ $a_1 = 1, a_2 = 1, a_{n+1} = a_n + a_{n-1}$ ）。

```
1 a1 = 1; disp(a1);
2 a2 = 1; disp(a2);
3 a3 = a1 + a2;
4 while a3 <= 100
5     disp(a3);
6     a1 = a2;
7     a2 = a3;
8     a3 = a1 + a2;
```

9 `end`

`while` 结构在每个循环开始会判断 `while` 后面的条件，如果条件成立，则进行一次循环，否则退出循环。以上的程序中由于我们事先并不知道我们要进行几次循环，所以选用 `while`，当最后一项大于 100 时，循环终止。运行结果为（每个数占一行）：

```
1  1  2  3  5  8 13 21 34 55 89
```

在 `for` 循环或 `while` 循环的内部，使用 `continue` 命令可以直接进入下一个循环（`while` 的仍然要先判断条件），使用 `break` 命令可以跳出循环。以下例程计算 100 以内的斐波那契数列的所有奇数项

```
1  a1 = 1; a2 = 1;
2  disp(a1); disp(a2);
3  a3 = a1 + a2;
4  while 1
5      a1 = a2;
6      a2 = a3;
7      a3 = a1 + a2;
8      if a3 > 100
9          break;
10     elseif mod(a3, 2) == 0
11         continue;
12     end
13     disp(a3);
14 end
```

先来看第 4 行，`double` 类型的非零数在这里会自动转换为 `logical` 类型的 1（`true`），只有 `double` 类型的 0 才会转换为 `logical` 类型的 0（`false`）。乍看之下，`while` 循环将永远执行下去（称为死循环），然而第 9 行的 `break` 在 `a3 > 100` 时就会使程序跳出循环。如果 `a3 <= 100` 且为偶数，则第 10 行的 `elseif` 判断为真，`continue` 命令被执行，程序将直接跳过之后的 `disp` 函数直接进入下一个循环，所以数列的偶数项都不会被输出。程序的运行结果为（每个数占

一行)

1 1 3 5 13 21 55 89

### return 命令

在一个脚本文件的任何地方，如果 `return` 命令被执行，则程序将结束该脚本文件的执行。如果该脚本文件是被单独执行的，程序将终止。如果该脚本文件是被其他脚本文件或函数文件调用的，程序将继续执行调用命令的下一个命令。

## Matlab 的函数

预备知识 Matlab 的判断与循环<sup>[18]</sup>

### 函数文件

我们已经学了一些函数，现在来看如何自定义函数。Matlab 中定义了函数的文件叫做**函数文件**。函数文件同样以“.m”作为后缀名，文件中的第一个命令必须是 `function`，用于定义主函数。文件名必须与主函数同名。文件中其他函数都是子函数。主函数可以调用子函数，子函数可以调用同文件中的其他子函数，但不能调用主函数，主函数和子函数都可以调用 Matlab 的内部函数或搜索路径下其他函数文件中的主函数。若函数文件在搜索路径下，其他 m 文件或 Command Window 中可以直接调用它的主函数。注意函数文件中的子函数不能从文件外被调用。

函数的 `workspace` 是独立的，即函数在执行的过程中，只能读写输入变量，函数内部定义的变量，以及全局变量（暂不介绍，不建议使用），而不能读取调用该函数的代码中的变量。相比之下，调用脚本相当于把脚本的代码直接插入到调用命令处，所以脚本中可以获取调用脚本的代码中的变量。注意函数只能通过函数文件定义，不能在脚本文件或控制行中定义。

### 函数句柄

**函数句柄（function handle）** 是一种特殊的变量类型，可用于定义一个临时的函数，也可传递到其他函数中。首先，对于已经存在的函数（包括函数文件定义的），可直接在函数名前面加 `@` 生成函数句柄

```
>> f = @sin
>> f(pi/2)
ans = 1;
```

若要将一个含有变量的表达式变为函数句柄，要在 `@` 后面用小括号指定函数句柄的变量

```
>> A = 3; w = 5; phi = pi/2;
>> f1 = @(x) A*sin(w*x+phi) + (2*x/pi).^2;
```



```
>> f1([0,pi/2])
ans =
    3.0000    1.0000;
>> f2 = @(x,phi) A*sin(w*x+phi) + (2*x/pi).^2;
>> f2([0,pi/2],pi/2)
ans =
    3.0000    1.0000;
```

以上的函数句柄在定义时用了逐个元素的幂运算“`.^`”，使函数句柄支持矩阵输入。其中 `f1` 的变量仅为 `x`，`f2` 的变量为 `x` 和 `phi`。注意如果在函数句柄定义后改变定义表达式中的变量，函数句柄不变。

```
>> A = 5; f1(0)
ans = 3.0000
```

有时候一些函数并不支持矢量运算，这时可以通过 `arrayfun` 函数来进行矢量运算，例如

```
>> f = @(x) x^2 + 1/x;
>> arrayfun(f, [1,2;3,4])
ans =
    2.0000    4.5000
    9.3333   16.2500
```

这样做相当于以下代码

```
1 f = @(x) x^2 + 1/x;
2 x = [1,2; 3,4];
3 ans = zeros(size(x));
4 for ii = 1:numel(x)
5     ans(ii) = f(x(ii));
6 end
```

### 自定义函数 (function)

自定义函数的格式为

```
[< 输出 1>,< 输出 2>,...] = function < 函数名 >(< 变量 1>,< 变量
```

```
2>...)
```

```
< 函数体 >
```

```
end
```

其中 < 函数名 > 是字母, 数字和下划线的组合, 例如 MyFun\_123, 第一个字符不能是数字或下划线. 若函数无变量, 则小括号可省略. 若函数无输出, 则等号及方括号可省略, 若只有一个输出, 方括号也可省略. 在一些情况下, 如果 函数体中没有使用某些输入变量, 就可以把这些变量用 “~” 符号代替.

函数的调用格式为

```
[< 输出变量 1>,< 输出变量 2>,...] = < 函数名 >(< 输入变量 1>,< 输入变量 2>...)
```

调用函数时, 如果输出变量个数少于函数定义中的输出变量个数, 则函数仅输出前几个变量. 若调用函数时不需要前面的某几个变量, 也可用 “~” 符号代替.

调用函数时, 输入变量的个数也可以少于函数定义中的输入变量, 但是函数体内部必须要做出相应的措施以防止函数体使用未生成的变量. 我们来看下面一个函数

```
1 function y = myfun(x, A, phi, y0)
2 if nargin < 4
3     y0 = 0;
4     if nargin < 3
5         phi = 0;
6         if nargin < 2
7             A = 1;
8         end
9     end
10 end
11 y = A*cos(x + phi) + y0;
12 end
```

注意函数体中使用了一个特殊的变量 `nargin`, 每当函数被调用时, 这个变量的值将会等于输入变量的个数 (同理, `nargout` 将等于输出变量的个数). 以上定义的函数允许 1-4 个输入变量, 函数体中的 2-10 行根据 `nargin` 的值对没

有输入的几个变量依次赋值。例如在控制行中调用该函数

```
>> myfun([0, pi/2])  
ans = 1.0000 0.0000  
>> myfun(0, 1, pi/2)  
ans = 0.0000
```

调用函数时的一种特殊格式是，如果被调用的函数的输入变量是若干个没有空格的字符串，则可以省略括号，逗号和双引号。例如我们之前见过的 `format` 函数

```
>> format long;
```

和将在“Matlab 画图<sup>[27]</sup>”中见到的

```
>> hold on; axis equal; xlabel x;
```

与脚本文件相同，若函数文件中执行了 `return` 命令，该函数的执行将立即终止，程序将执行调用该函数的命令的下一个命令。若该函数是单独被调用的（例如按开始按钮或在控制行被直接调用），则程序结束。

## Matlab 画图

### 预备知识 Matlab 的文件，程序结构和函数<sup>[??]</sup>

Matlab 具有强大的画图功能，这里仅介绍一些基础知识。最常用的画图函数是 `plot`，例如

```
>> x = linspace(0, 2*pi, 100); y = sin(x);  
>> plot(x, y);
```

结果如图 1（左）所示。如果要在该坐标系继续画图，要用 `hold on` 命令（`on` 是 `hold` 的输入变量），否则每用一次 `plot`，之前画过的图都会被清除。用 `hold off` 可以重新恢复自动清除。

```
>> y1 = cos(x);  
>> hold on; plot(x, y1);
```

结果如图 1（右）所示，注意新增曲线的颜色变化。

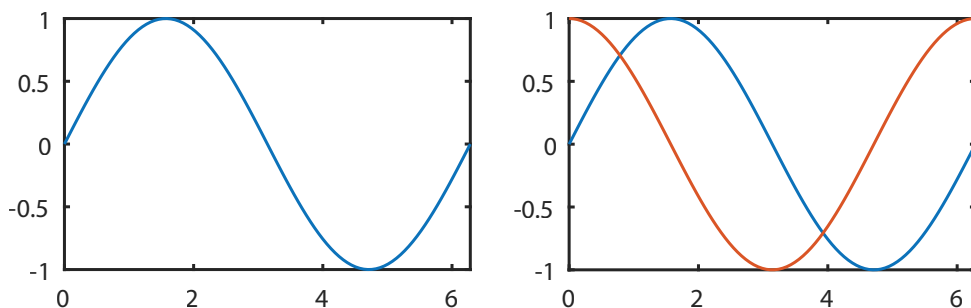


图 1: `plot` 函数

如果我们要新建一个画图窗口，用 `figure` 函数。若要指定画图的颜色，可以添加 `figure` 的第三个变量，用一个字符表示颜色（red: 'r', green: 'g', blue: 'b', yellow: 'y', magenta: 'm', cyan: 'c', black: 'k', white: 'w'）。例如

```
>> x2 = cos(x); y2 = sin(x);  
>> figure; plot(x2, y2, 'r');
```

在新增的窗口中, 结果如图 2 (左) 所示. 注意根据窗口尺寸的不同,  $x$  轴和  $y$  轴的单位长度一般不同, 若要使其相同, 可以在 `plot` 后面用 `'axis equal'` 命令 (其中 `equal` 是 `axis` 函数的输入变量), 得到图 2 (右). 若要调整坐标

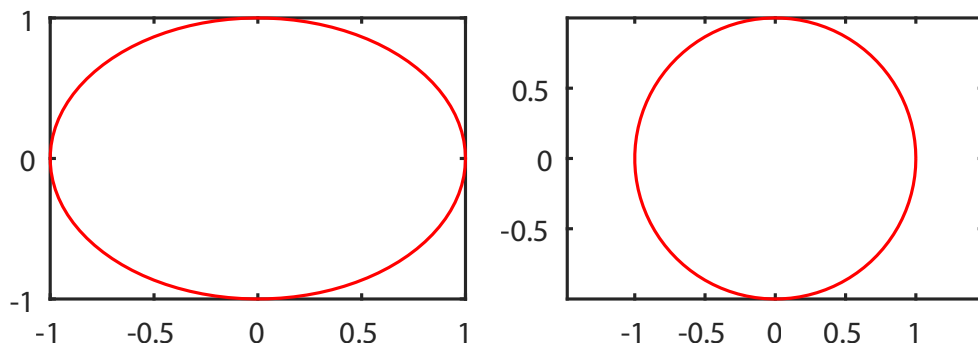


图 2: 红色的单位圆

轴的范围, 也可用 `axis` 函数. 另外可以在 `plot` 的第三个变量的字符串中设定曲线的形状, 用 `xlabel` 和 `ylabel` 函数分别设置  $x$  轴和  $y$  轴的文字, 用 `title` 函数设置图片标题

```
>> plot(x2, y2, '.-r');
>> axis([-1.2, 1.2, -1.2, 1.2]);
>> xlabel('x'); ylabel('y'); title('unit circle');
```

其中 `'.-'` 表示带点的连线, 点的坐标由 `x2` 和 `y2` 决定 (另外 `'+-'` 表示带加号的连线, `'o-'` 表示带圆圈的连线). `axis` 中行矢量中的四个数分别是  $x$  轴的最小最大值和  $y$  轴的最小最大值. 结果如图 3 (左) 所示.

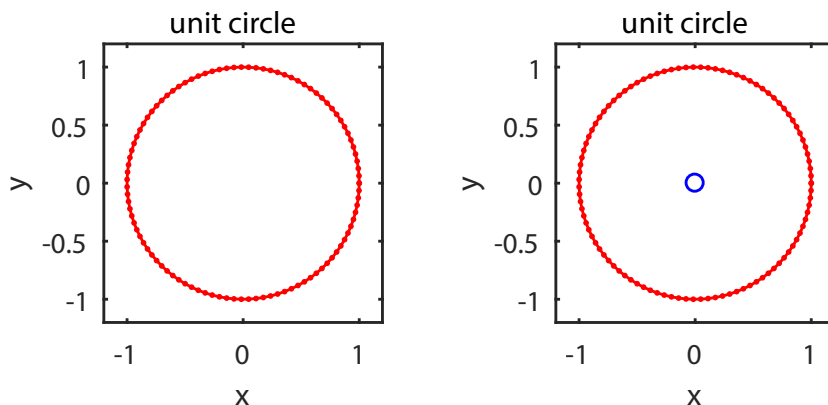


图 3: 红色的单位圆

除了 `plot` 以外，常用的还有 `scatter` 函数，用于画散点图。格式与 `plot` 相似。默认的散点形状是圆圈，但也可以在第三个变量中设置颜色和 `'+'`，`'x'`，`'.'` 等形状。例如

```
>> hold on; scatter(0, 0, 'b');
```

结果如图 3（右）所示。

最后，如果要关闭当前画图窗口，用 `close` 函数（无输入变量），如果要关闭所有窗口，用 `close all` 即可。

## Matlab 的程序调试及其他功能

预备知识 Matlab 的文件，程序结构和函数[??]

### 程序调试

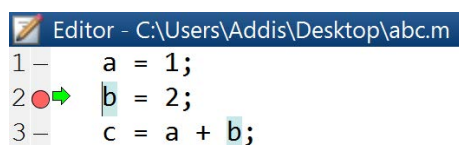


图 1: 在行首设置 Breakpoint

若要调试程序，可选择一行代码并单击该行前面的横线，这时会出现红色圆点 Breakpoint（图 1），程序运行到 Breakpoint 会暂停。

此时要查看变量情况，可通过 Workspace 查看各个变量的情况，也可用光标悬停在某个变量上。还可以用 Command Window 改变某些变量的值，或画图 etc. 在这种调试状态下，也可以通过 Edit 菜单中的一些按钮控制接下来程序如何运行（图 2）。其中“Continue”（快捷键 F5）是继续运行直到下一个

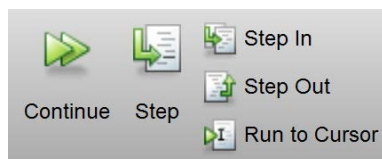


图 2: Step 菜单

Breakpoint 或结束。“Step”（F10）是运行到下一行，“Step In”（F11）是进入子程序并暂停，“Step Out”是运行完当前子程序并回到子程序被调用的地方。“Run to Cursor”是运行到光标所在处。

### warning 和 error 函数

有时候当我们的程序中出现了某个错误时，程序会终止并在控制行中返回一个 error 信息，例如我们给 sin 函数输入两个变量时，控制行的 error 信息将提示变量个数太多。

```
>> sin(1,2)
Error using sin
Too many input arguments.
```

另一些情况下当错误不是那么严重时，我们会得到一个 warning 提示，程序在输出提示后将继续运行。

我们可以在我们自己的脚本或函数中用 error 或 warning 函数达到同样的效果

```
1 function myfun(char)
2 if char == 'w'
3     warning('this is a warning');
4 elseif char == 'e'
5     error('this is an error');
6 else
7     disp('Hello World!');
8 end
9 end
```

当 error 或 warning 函数被执行时，控制行不但会输出对应的字符串，还会输出它们所在的文件和行号，如

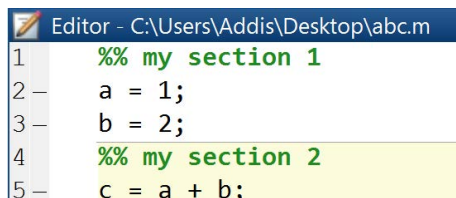
```
>> myfun w
Warning: this is a warning
> In myfun (line 3)
```

单击带下划线的单词，可在 Editor 中显示对应位置。若 myfun 函数被其他函数调用，那么调用的位置也会被逐级列出。使用 warning 的效果类似，但程序不会被终止。



## 分节

在行首用两个百分号“%%”可以对代码进行分节（图 3）。这样做一是可以使代码结构更清晰，二是可以单独选择某一节运行（Edit 菜单中的“Run Section”按钮）。



```
1 %% my section 1
2 a = 1;
3 b = 2;
4 %% my section 2
5 c = a + b;
```

图 3: 代码分节

## 工具箱（Toolbox）

在购买和安装 Matlab 软件时，可以选择各种各样的工具箱，常用的工具箱有曲线拟合（Curve Fitting，从离散的数据点得到一条曲线），图像处理（Image Processing，图像变换，增强，降噪，二值化等），图像获取（Image Acquisition，从相机获取图像），Matlab 编译器（MATLAB Compiler，编译代码，提高运行速度）。注意使用了工具箱功能的代码在没有工具箱的 Matlab 软件上将无法运行。

## Matlab Online

具有 Matlab 的基本功能，和类似于软件的界面，需要购买了正版 Matlab 的 Matlab 账号登录（学生账号也可以）。若账号购买了工具箱（Toolbox），也可以使用对应的工具箱。本书官网提供免费的 Matlab 账号供读者试用和体验 Matlab Online，网址见本书封面或前言。

## 二分法

二分法是数值计算中一种求连续一元函数零点的简单方法. 先来看一个显然的结论: 如果我们知道一个连续函数在某个开区间左端的值和右端的值 (假设都不为 0) 乘积小于 0 (即正负号不同), 那么这个函数在该区间内必有至少一个根. 为了进一步缩小这个根所在的区间, 我们在区间中点求函数值, 如果中点处的函数值与区间左端的函数值同号 (相乘大于 0), 则函数的根必然在区间中点和区间右端之间, 于是我们可以把区间左端移动到区间中点处, 再来求新区间的终点. 如果区间中点的函数值与区间右端的函数值同号, 同理我们也可以把区间右端移动到区间中点处得到新的区间. 如果区间中点的函数值恰好为 0, 我们便找到了一个根, 另一方面, 如果区间的长度小于我们对根的精度的要求, 那么我们就找到了一个根的近似值.

Matlab 中自带的 `fzero` 函数如果按照以下格式使用, 大致可以认为是二分法

```
>> f = @(x)x-1;
>>fzero(f, [0,2])
ans = 1
```

`fzero` 的默认精度是  $2.2\text{e-}16$ . 注意要把一个函数作为其他函数的输入变量, 必须使用函数句柄. 下面我们给出一个二分法的 Matlab 程序.

### bisection.m

```
1 % 二分法求函数的根
2 function x = bisection(f, int, err)
3 fl = f(int(1)); fr = f(int(2));
4 % 两端点是否为 0
5 if fl == 0
6     x = int(1); return;
7 elseif fr == 0
8     x = int(2); return;
```

```
9  end
10 % 两端点是否同号
11 if fl * fr > 0
12     error('两端点同号');
13 end
14 % 主循环
15 while(int(2) - int(1) > err)
16     mid = 0.5*(int(1) + int(2));
17     fm = f(mid);
18     if fm * fl > 0
19         int(1) = mid;
20     elseif fm * fr > 0
21         int(2) = mid;
22     else % fm = 0
23         break;
24     end
25 end
26 x = mid;
27 end
```

我们先来看函数的自变量，与 `fzero` 类似，该函数的前两个输入变量分别是函数句柄和求根区间，第三个变量是误差值，当区间的长度小于 `err` 时，函数将输出区间中点作为输出。函数的第 3-13 行做了一些必要的检查，确保区间两端的函数值为异号。第 15 行开始主循环，每循环一次，函数的区间长度减半，知道区间中点处的函数值为 0 或区间长度小于 `err` 时跳出循环，最后返回区间中点的函数值。

## 多区间二分法

### 预备知识 二分法<sup>[?]</sup>

这里介绍一种多区间二分法，可以求出连续函数在某区间内几乎全部的根。方法就是把这个区间等分为若干个相等的小区间，然后分别判断这些小区间两端函数值的符号，对所有两端异号的区间使用二分法即可。显然，小区间的个数越多，越有可能找到所有的根。例程如下。

#### bisectionN.m

```
1 function roots = fzeroN(f, int, N)
2 x = linspace(int(1), int(2), N); % 划分区间
3 y = arrayfun(f, x); % 求所有 f(x(ii))
4 figure; plot(x, y, '.-') % 画图
5 title('f(x)')
6 Sign = sign(y);
7 ind = find(Sign(1:end-1) .* Sign(2:end) <= 0); % 找符合
   条件的区间序号
8 Nroot = numel(ind);
9 roots = zeros(1, Nroot); % 预赋值
10 for ii = 1:Nroot
11     roots(ii) = fzero(fun, [x(ind(ii)), x(ind(ii)+1)]);
12 end
13 end
```

函数的前两个输入变量分别是需要求根的函数句柄和求根区间（二元行矢量或列矢量），第三个变量  $N$  是子区间端点的个数（即子区间的个数加一）。函数中先求出所有的端点  $x$ ，以及对应的函数值  $y$ ，然后画图。第 6-7 行寻找所有两端异号或有一端为 0 的区间的序号，然后在第 10 行的循环中对这些区间逐个使用二分法。为了提高运算效率，这里并没有使用“二分法<sup>[?]</sup>”中的例程，而是使用了 Matlab 自带的 `fzero` 函数。

`bisectionN` 的画图功能是为了让用户判断是否有可能出现漏根，以下举两个例子说明。

```
>>f = @(x)exp(-0.2*x)*sin(x);  
>>roots = bisectionN(f, [0, 15], 50)  
roots = 0 3.1416 6.2832 9.4248 12.5664
```

运行结果如图 1，由于画出的曲线较为光滑，可判断漏根的可能性很小。再看

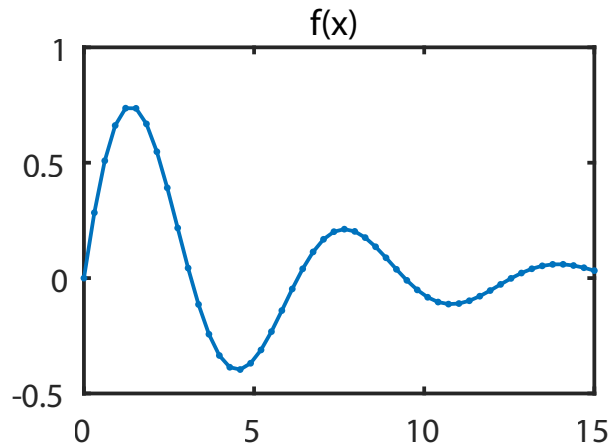


图 1: 运行结果

另一个例子

```
>>f = @(x)sin(1/x);  
>>roots = bisectionN(f, [0, 0.3], 50)  
roots = 0.0245 0.0398 0.0455 0.0531 0.0637 0.0796 0.1061 0.1592
```

我们已经知道函数  $\sin(1/x)$  在该区间上有无数个根，且越接近  $x = 0$ ，相邻根之间的距离越小。运行结果如图 2，可见在区间  $[0, 0.1]$  内，子区间端点的函数值非常不平滑，极有可能出现漏根。为了求得更多的根，我们可以增加子区间的个数。

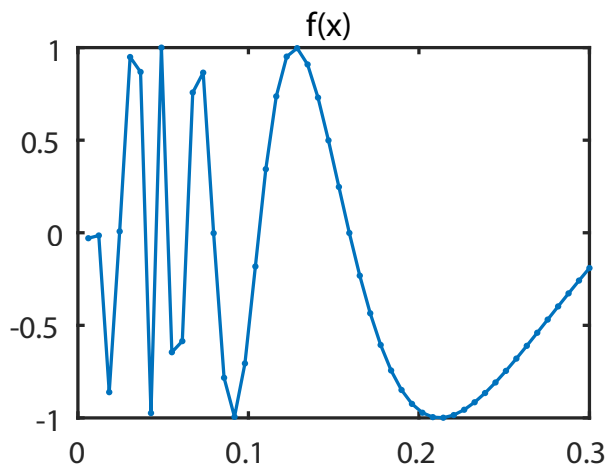


图 2: 运行结果

## 冒泡法

### 预备知识 Matlab 的程序结构<sup>[??]</sup>

我们先来看 Matlab 自带的排序函数 `sort`. 假设数列 `age` 是几个小朋友的年龄, `name` 是这几个小朋友对应的名字, 现在按年龄从小到大排序如下

```
>> age = [1, 6, 2, 5, 3];
>> name = ['a', 'b', 'c', 'd', 'e'];
>> [age1, order] = sort(x);
age1 = 1 2 3 5 6
order = 1 3 5 4 2
>> name1 = name(order);
name1 = 'acedb'
```

其中输出变量 `order` 是排序后每个数字在原来数列中的位置索引, 即 `name1` 等于 `name(order)`. 现在我们用冒泡法实现 `sort` 的功能. `sort` 函数默认把数列升序排列, 即第二个输入变量默认为 `'ascend'`. 若要降序排列, 可以用 `'descend'` 作为第二个输入变量.

冒泡法是一种简单的排序算法, 效率没有 `sort` 中的算法快, 所以在实际使用时还是建议用 `sort`. 冒泡法的算法为: 以升序排列为例, 给出一个数列,

先把第一个数与第二个进行比较，若第一个数较大，就置换二者的位置（具体操作是，把第一个数的值赋给一个临时变量，再把第二个数的值赋给第一个，最后把临时变量的值赋给第二个），再把第二个数与第三个进行比较，若第二个较大，就置换二者的位置，这样一直进行到最后两个数，完成第一轮。然后再进行第二轮，第三轮，直到某一轮没有出现置换操作，即可确定排序完成。至于输出变量 `order`，我们可以先令 `order = 1:N`，每置换数列的两个数，就把 `order` 中对应的两个数也置换即可。这样，数列与其原来的索引将始终以一一对应。

### **bubble.m**

```
1 % 冒泡法排序
2 function [x, order] = bubble(x, option)
3 N = numel(x); % 数列个数
4 order = 1:N; % 索引
5 changed = 1; % 是否有置换
6 while(changed == 1)
7     changed = 0;
8     for ii = 1:N-1
9         if x(ii) > x(ii + 1)
10             % 置换
11             changed = 1;
12             temp = x(ii);
13             x(ii) = x(ii + 1);
14             x(ii + 1) = temp;
15             temp = order(ii);
16             order(ii) = order(ii + 1);
17             order(ii + 1) = temp;
18         end
19     end
20 end
21 % 是否是降序排列
```

```
22 if nargin > 1 && option(1) == 'd'
23     x(:) = flipud(x(:));
24     order = fliplr(order);
25 end
26 end
```

第 6 行的循环每循环一次，数列将从头到尾被扫描一遍。每个循环开始时 `changed` 的值被设为 0，若有任何置换，`changed` 则变为 1（第 11 行），使 `while` 的判断条件成立，循环继续。为了使第一个循环发生，在循环前必须把 `changed` 设为 1。再来看第 9-18 行的判断结构。如果前一个数大于后一个数，则置换发生。注意要置换数列中的两个数，必须要设一个临时变量（`temp`）。函数的最后，判断输入变量的个数，如果只有一个变量，则默认按照前面的代码升序排列，若第二个变量为 `'descend'`，则把 `x` 和 `order` 翻转一下即可。



## 四阶龙格库塔法

预备知识 中点法解常微分方程（组）<sup>[44]</sup>

龙格库塔法是一类数值解微分方程的算法，其中较常见的是四阶龙格库塔法。这里不进行推导，仅仅给出公式如下（ $y_n, t_n, h$  的定义类比<sup>??[??]</sup>）

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (1)$$

其中

$$\begin{aligned} k_1 &= f(y_n, t_n) & k_2 &= f\left(y_n + h\frac{k_1}{2}, t_n + \frac{h}{2}\right) \\ k_3 &= f\left(y_n + h\frac{k_2}{2}, t_n + \frac{h}{2}\right) & k_4 &= f(y_n + hk_3, t_n + h) \end{aligned} \quad (2)$$

由以上两式，不难把该算法拓展到方程组的情况。对于  $N$  元微分方程组

$$\begin{cases} y_1'(t) = f_1(y_1, \dots, y_N, t) \\ y_2'(t) = f_2(y_1, \dots, y_N, t) \\ \vdots \\ y_N'(t) = f_N(y_1, \dots, y_N, t) \end{cases} \quad (3)$$

我们可以把该式记为矢量函数的形式

$$\mathbf{y}'(t) = \mathbf{f}(\mathbf{y}, t) \quad (4)$$

现在我们仅需要把式 1 和 式 2 中的所有  $y_i$  和  $k_i$  都变为  $N$  维列矢量  $\mathbf{y}_i$  和  $\mathbf{k}_i$  即可将微分方程拓展为微分方程组。

### 例题

到目前为止，我们每求一个微分方程的数值解都要重新写一次程序，对于一些较为复杂的算法这样做效率较低。我们这里不妨把四阶龙格库塔法写到一个单独的函数文件 `odeRK4.m` 中，当我们要解某个特定的方程时，只需把式 2 中的  $f(y, t)$  作为自变量输入即可解出  $y(t)$ 。

**odeRK4.m**

```

1 function [Y, t] = odeRK4(f, tspan, Y0, Nt)
2 Nvar = numel(Y0); % 因变量的个数
3 dt = (tspan(2) - tspan(1)) / (Nt-1); % 计算步长
4 Y = zeros(Nvar, Nt); % 预赋值
5 Y(:, 1) = Y0(:); % 初值
6 t = linspace(tspan(1), tspan(2), Nt);
7
8 for ii=1:Nt-1
9     K1 = f(Y(:,ii), t(ii));
10    K2 = f(Y(:,ii)+K1*dt/2, t(ii)+dt/2);
11    K3 = f(Y(:,ii)+K2*dt/2, t(ii)+dt/2);
12    K4 = f(Y(:,ii)+K3*dt, t(ii)+dt);
13    Y(:,ii+1) = Y(:,ii) + dt/6 * (K1+2*K2+2*K3+K4);
14 end
15 end

```

我们先来看第 1 行的函数声明，输入变量中， $f$  是式 4 中  $f(y, t)$  的函数句柄<sup>[??]</sup>， $tspan$  是一个  $2 \times 1$  的列矢量， $tspan(1)$  是初始时间， $tspan(2)$  是终止时间， $Y0$  是一个列矢量， $Y0(ii)$  是第  $ii$  个因变量的初始值， $Nt$  是  $t_n$  的个数， $tspan$  定义的时间区间被等分为  $Nt - 1$  个小区间。因变量中， $Y$  的行数是因变量的个数，列数是  $Nt$ ， $t$  是一个行矢量，由第 6 行定义， $Y(ii, jj)$  就是第  $ii$  个变量在  $t(jj)$  时刻的值。第 5 行把初值  $Y0$  赋给  $Y$  的第 1 列，第 8-14 行的循环根据式 1 和式 2 的矢量形式由  $Y$  的第  $ii$  列 ( $y_i$ ) 求第  $ii+1$  列 ( $y_{i+1}$ )。

我们先来用这个函数来计算“天体运动的简单数值计算<sup>[??]</sup>”中的问题。我们令因变量  $y$  的四个分量依次为一阶方程组 (??<sup>[??]</sup>)

$$\begin{cases} x' = v_x \\ y' = v_y \\ v_x' = -GMx/(x^2 + y^2)^{3/2} \\ v_y' = -GM y/(x^2 + y^2)^{3/2} \end{cases} \quad (5)$$

中的  $x, y, v_x, v_y$ 。程序代码如下

**KeplerRK4.m**

```
1 function KeplerRK4
2 % 参数设定
3 GM = 1; % 万有引力常数乘以中心天体质量
4 x0 = 1; y0 = 0; % 初始位置
5 vx0 = 0; vy0 = 0.7; % 初始速度
6 tspan = [0; 4]; % 总时间和步数
7 Nt = 100; % 步数
8
9 Y0 = [x0; y0; vx0; vy0]; % 因变量初值
10 f = @(Y, t)fun(Y, t, GM);
11 [Y,~] = odeRK4(f, tspan, Y0, Nt);
12
13 % 画图
14 figure; hold on;
15 plot(Y(1,:), Y(2,:));
16 scatter(0, 0);
17 axis equal;
18 end
19
20 function Y1 = fun(Y, ~, GM)
21 % 因变量
22 x = Y(1); y = Y(2);
23 vx = Y(3); vy = Y(4);
24 Y1 = zeros(4,1); % 预赋值
25 Y1(1) = vx;
26 Y1(2) = vy;
27 temp = -GM / (x^2 + y^2)^(3/2);
28 Y1(3) = temp * x;
29 Y1(4) = temp * y;
30 end
```

运行结果如图 1 所示.

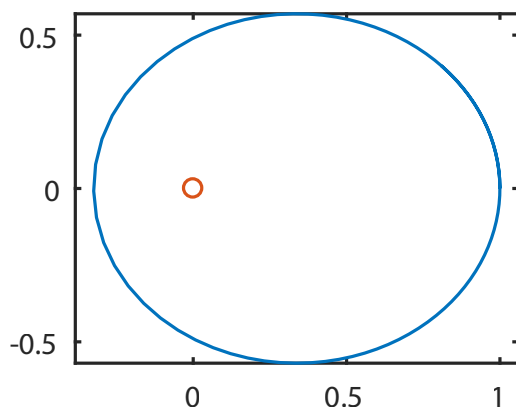


图 1: 运行结果

我们先来看函数 `fun` (20 行), 这个函数就相当于式 5. 第一个输入变量  $Y$  是一个列矢量, 是  $y_n$  的值, 第二个输入变量是  $t$ , 但由于式 5 中没有出现  $t$ , 我们用波浪线代替. 第三个输入变量是参数  $GM$ , 即万有引力常数和中心天体质量之积. 输出变量  $Y1$  是一个列矢量, 是  $y'_n$  的值.

再来看主函数 `KeplerRK4` (第 1 行), 参数设定中除了步数从 4000 变为了 100, 其他都和“天体运动的简单数值计算[?]”中的程序一样, 然而这里运行结果却精确得多 (曲线几乎闭合), 可见这种算法的优越性.

主函数第 10 行中将 `fun(Y, t, GM)` 变为函数句柄 `f(Y, t)`, 这样  $GM$  就可以在“参数设定”中设置, 而不用在 `fun` 函数内部设置. 第 11 行调用了上文中的 `odeRK4` 函数解方程组, 由于我们在画图时不需要用  $t$ , 所以把第二个输出变量改为波浪线.

## 中点法解常微分方程（组）

预备知识 常微分方程（组）的数值解<sup>[?]</sup>

我们先来尝试用欧拉法解一阶微分方程

$$y'(t) = y \quad (1)$$

令初始条件为  $y(0) = 1$ . 令步长为  $h = 0.1$ , 步数为 5, 结果如图 1 所示（代码见词条最后）.

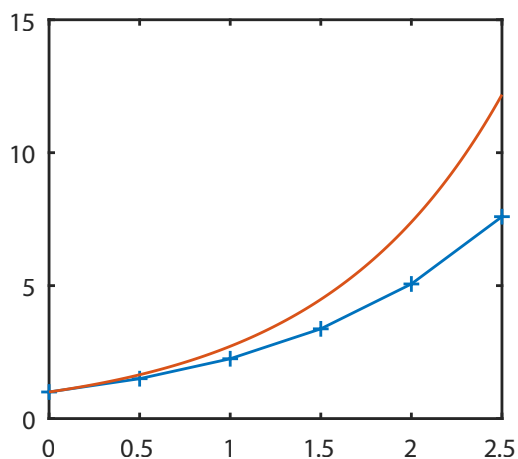


图 1: 欧拉法数值解（蓝）和解析解（红）

由“”我们知道该方程的解析解为  $y = e^t$ . 对比数值解和解析解, 不难分析出误差产生的原因: 我们仅用每段步长区间左端的导数预测整个区间的曲线增量. 如果我们能利用每个区间中点的导数计算整个区间的增量, 这个预测将会比欧拉法更精确.

考虑微分方程  $y'(t) = f(y, t)$  在区间  $[t_n, t_{n+1}]$  的曲线, 若我们已知区间左端的函数值为  $y_n$ , 我们可以先用微分近似估计曲线中点的函数值为

$$y\left(t_n + \frac{h}{2}\right) = y_n + \frac{h}{2}f(y_n, t_n) \quad (2)$$

然后再求出这个近似中点的导数为

$$y'\left(t_n + \frac{h}{2}\right) = f\left[y_n + \frac{h}{2}f(t_n, y_n), t_n + \frac{h}{2}\right] \quad (3)$$

最后我们利用这个导数估算该区间的曲线增量

$$y_{n+1} = y_n + hy' \left( t_n + \frac{h}{2} \right) = y_n + hf \left[ y_n + \frac{h}{2} f(t_n, y_n), t_n + \frac{h}{2} \right] \quad (4)$$

这就是解常微分方程的**中点法**。

我们再来用中点法取同样的步长计算[式 1](#)，结果如[图 2](#)所示。

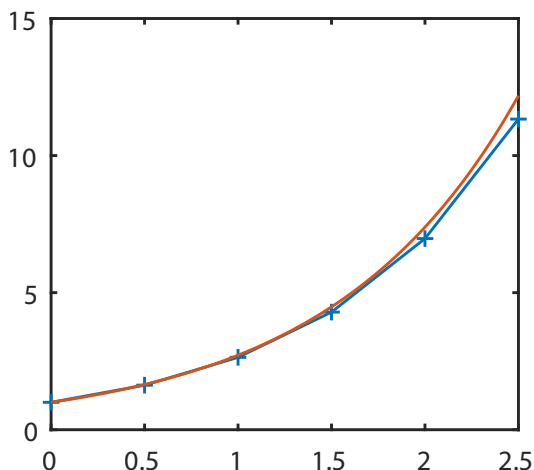


图 2: 欧拉法数值解（蓝）和解析解（红）

可见虽然中点法每一步的计算过程比欧拉法稍微复杂一些，但精度却大大地提高了。

中点法同样适用于微分方程组，例如对于常微分方程组

$$\begin{cases} x'(t) = f(x, y, t) \\ y'(t) = g(x, y, t) \end{cases} \quad (5)$$

首先计算近似中点为

$$\begin{cases} x_{n+1/2} = x_n + hf(x_n, y_n, t_n) \\ y_{n+1/2} = y_n + hg(x_n, y_n, t_n) \end{cases} \quad (6)$$

然后有

$$\begin{cases} x_{n+1} = x_n + hf \left( x_{n+1/2}, y_{n+1/2}, \frac{h}{2} \right) \\ y_{n+1} = y_n + hg \left( x_{n+1/2}, y_{n+1/2}, \frac{h}{2} \right) \end{cases} \quad (7)$$

## 例程

```
1  % 设置参数
2  N = 6;
3  h = 0.5;
4  t = linspace(0, (N-1)*h, N); % 自变量
5  t0 = linspace(0, (N-1)*h, 100); % 用于画图
6
7  % 欧拉法
8  y = zeros(1,N); % 预赋值
9  y(1) = 1; % 初值
10 for ii = 1:N-1
11     y(ii+1) = y(ii) + h*y(ii);
12 end
13
14 % 画图
15 figure;
16 plot(x,y, '+-');
17 hold on;
18 plot(t0, exp(t0));
19
20 % 中点法
21 y = zeros(1,N);
22 y(1) = 1;
23 for ii = 1:N-1
24     y(ii+1) = y(ii) + h*(y(ii) + 0.5*h*y(ii));
25 end
26
27 % 画图
28 figure;
29 plot(x,y, '+-');
```

```
30 hold on;  
31 plot(x0, exp(x0));
```



## 本书编写规范

预备知识 本书编写规范<sup>[48]</sup>

### 软件使用规范

本书使用 TeXLive2016 软件中的 XeLaTeX 进行编译. 如果 Windows 中编译卡在 eu1lmr.fd 上的时间较长, 说明 font config 有问题, 在 Windows 的控制行运行 “fc-cache -fv”, 重启 TeXLive, 多试几次即可. TeXWork 编辑器中 Ctrl+T 编译, Ctrl+ 单击跳转到对应的 pdf 或代码, 在 pdf 中 Alt+ 左箭头返回上一个位置. 代码中 \beq+Tab 生成公式环境, \sub+Tab 生成 subsection. Ctrl+F 进行查找, Ctrl+G 查找下一个. 菜单中的 Edit>Preference 设置默认字体为 Microsoft YaHei UI (11pt), 默认编译器为 XeLaTeX, 编码选择 UTF-8.

搜索文件夹内所有文档的内容用 FileSeek 软件, 搜索空格用 “\空格”, 搜索 “\$” 用 “\\$”, 以此类推. 对比两个文档或文件夹用 WinMerge 软件.

画图用 Adobe Illustrator 和 Autodesk Graphics, 用 MathType 在图中添加公式, 希腊字母粗体正体矢量用从 Symbol 字体中插入, 更简单的方法是, 先输入希腊字母, 选中, 然后在 Style 里面选 Vector-Matrix.

### 文件版本管理

使用 GitHub Desktop, 用 PhysWiki repository 管理所有文件, 每次 commit 需要做的事情如下

- 用 FileSeek 替换所有文档中的空心句号.
- 确保所有文档可以顺利编译.
- 解决编译产生的 warning.
- 清除编译产生的中间文件, 包括 content 文件夹中的非 tex 文件.
- 把 ManicTime 记录的写作时间记录到 “计时.txt”.

- 检查变化的内容.

每次 commit 的标题必须是下列之一

- 常规更新: 包括完善词条, 新词条等.
- 词条统计: 统计文件夹, 对照表, 和书中的词条, 查看不一致或缺失.
- 模板更新: 模板有更新.

词条统计的方法: 首先把 `contents` 文件夹中的所有文件名按顺序排列, 复制到表格中, 然后把词条对照表中的所有标签在表格中找到对应项, 做标记, 并把对照表中的词条名粘贴到表格中. 最后到 `PhysWiki.tex` 中逐个把标签在表格中找到对应项, 做标记, 对照词条名, 并对照词条文件中第一行的词条名.

## 词条编写规范

词条标签必须限制在 6 个字符内, 必须在 `PhysWiki.tex`, 词条标签对照表和词条文件名中一致. 词条的中文名必须在 `PhysWiki.tex`, 词条标签对照表和词条文件的第一行注释中一致. 如果不是超纲词条, 在主文件中用 `\entry` 命令, 否则用 `\Entry` 命令. 非超纲词条放在 `contents` 文件夹, 超纲词条放在 `contents1` 文件夹. 引用词条用 `\upref` 命令, “预备知识”用 `\pentry` 命令, “应用实例”用 `\eentry` 命令, 拓展阅读用 `\rentry` 命令. 总文件 `PhysWiki.tex` 编译较慢, 可以先使用 `debug.tex` 编译, 然后再把 `\entry` 或 `\Entry` 指令复制到 `PhysWiki.tex` 中. 注意 `PhysWiki.tex` 中 `\entry` 命令的后面可以用 `\newpage` 命令强制换页, 但为了排版紧凑不推荐这么做.

## 黑色的小标题

正文必须使用中文的括号, 逗号, 引号, 冒号, 分号, 问号, 感叹号, 以及全角实心的句号. 所有的标点符号前面不能有空格, 后面要有空格. 行内公式用单个美元符号, 且两边要有空格, 例如  $a^2 + b^2 = c^2$ , 后面有标点符号的除外. 方便的办法是先全部使用中文标点, 最后再把所有空心句号替换成全角实心句号.

公式的 label 必须要按照“词条标签\_eq 编号”的格式, 只有需要引用的公式才加标签, 编号尽量与显示的编号一致, 但原则上不重复即可. 图表的标

签分别把 eq 改成 fig 和 tab 即可，例题用 ex. 但凡是有 \caption 命令的，\label 需要紧接其后.

$$(a+b)^n = \sum_{i=0}^n C_n^i a^i b^{n-i} \quad (n \text{ 为整数}) \quad (1)$$

引用公式和图表都统一使用 \autoref 命令，注意前面不加空格后面要加空格，例如式 1. 如果要引用其他词条中的公式，可以引用“其他词条<sup>[48]</sup>”的式 1 也可以用“式 1<sup>[48]</sup>”，为了方便在纸质书上使用，词条页码是不能忽略的.

### 错别字替换

正文中常见的错别字如“他们”（它们），“一下”（以下），可以时常搜索替换.

### 公式规范

公式中的空格从小到大如  $a b c d e$ ，微分符号如  $dx$ ，自然对数底如  $e$ ，双重极限如

$$\lim_{\substack{\Delta x_i \rightarrow 0 \\ \Delta y_i \rightarrow 0}} \sum_{i,j} f(x_i, y_i) \Delta x_i \Delta y_j \quad (2)$$

导数和偏导尽量用 Physics 宏包里面的

$$\frac{d}{dx} \quad \frac{df}{dx} \quad \frac{d^2 f}{dx^2} \quad d^2 f / dx^2 \quad \frac{\partial}{\partial x} \quad \frac{\partial f}{\partial x} \quad \frac{\partial^2 f}{\partial x^2} \quad \partial^2 f / \partial x^2 \quad (3)$$

复数如  $u+iv$ ，复共轭如  $a^*$ ，行内公式如  $a/b$ ，不允许行内用立体分式. 公式中的绝对值如  $|a|$ ，矢量如  $\mathbf{a}$ ，手写矢量如  $\vec{a}$ ，单位矢量如  $\hat{\mathbf{a}}$ ，矢量点乘如  $\mathbf{A} \cdot \mathbf{B}$ （不可省略），矢量叉乘如  $\mathbf{A} \times \mathbf{B}$ . 量子力学算符如  $\hat{a}$ ，狄拉克符号如  $\langle a|, |b\rangle, \langle a|b\rangle$ . 梯度散度旋度拉普拉斯如  $\nabla V, \nabla \cdot \mathbf{A}, \nabla \times \mathbf{A}, \nabla^2 V$ ，但最好用  $\boldsymbol{\nabla} V, \boldsymbol{\nabla} \cdot \mathbf{A}, \boldsymbol{\nabla} \times \mathbf{A}, \nabla^2 V$ . 单独一个粗体的  $\nabla$  用  $\boldsymbol{\nabla}$ . 行列式，矩阵  $\mathbf{A}$ ，转置  $\mathbf{A}^T$ ，厄米共轭  $\mathbf{A}^\dagger$  如

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^T \quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}^\dagger \quad (4)$$

行内的列矢量用行矢量的转置表示，如  $(1, 2, 3)^T$ .

行间公式换行及对齐用 `aligned` 环境，或用自定义的 `\ali` 命令

$$\begin{aligned}(a-b)^2 &= a^2 + b^2 - 2ab \\ &= a^2 + b^2 + 2ab - 4ab \\ &= (a+b)^2 - 4ab\end{aligned}\tag{5}$$

$$\begin{aligned}k_1 &= f(y_n, t_n) & k_2 &= f\left(y_n + h\frac{k_1}{2}, t_n + \frac{h}{2}\right) \\ k_3 &= f\left(y_n + h\frac{k_2}{2}, t_n + \frac{h}{2}\right) & k_4 &= f(y_n + hk_3, t_n + h)\end{aligned}\tag{6}$$

左大括号用自定义的 `\leftgroup` 命令，里面相当于 `aligned` 环境

$$\left\{\begin{array}{l}d+e+f=g \\ a+b=c\end{array}\right.\tag{7}$$

可变化尺寸的斜分数线如下

$$\frac{d^2X}{dx^2} \Big/ X + \frac{d^2Y}{dy^2} \Big/ Y + \frac{d^2Z}{dz^2} \Big/ Z = \frac{1}{c^2} \frac{d^2T}{dt^2} \Big/ T\tag{8}$$

希腊字母如下

$$\begin{aligned}\alpha(a), \beta(b), \chi(c), \delta(d), \epsilon/\varepsilon(e), \phi(f), \gamma(g), \eta(h), \iota(i), \varphi(j), \kappa(k), \lambda(l), \mu(m), \\ \nu(n), o(o), \pi(p), \theta(q), \rho(r), \sigma(s), \tau(t), \upsilon(u), \varpi(v), \omega(w), \xi(x), \psi(y), \zeta(z)\end{aligned}\tag{9}$$

以下是 `script` 字母，只有大写有效。所谓大写  $\epsilon$  其实是花体的  $E$ 。

$$A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\tag{10}$$

另外，电介质常数一律用  $\epsilon$  而不是  $\varepsilon$ 。

写单位，用 `\si`，如  $a = 100 \text{ m/s}^2$ 。

## 图表

现在来引用一张图片，图片必须以 `eps` 以及 `pdf` 两种格式放在 `figures` 文件夹中，代码中使用 `pdf` 图片。图片宽度一律用 `cm` 为单位。在图 1 中，`label` 只能放在 `caption` 的后面，否则编号会出错。由于图片是浮动的，避免使用“上图”，“下图”等词。

再来看一个表格，如表 1。注意标签要放在 `caption` 后面，使用 `tb` 命名。

下面我们举一个例子并引用

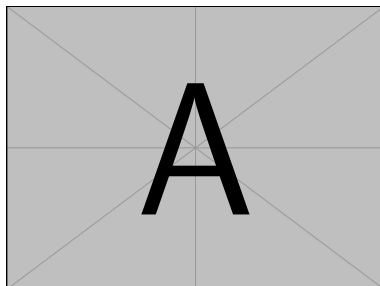


图 1: 例图

表 1: 极限 e 数值验证

$x$	$10^{-1}$	$10^{-2}$	$10^{-3}$	$10^{-4}$	$10^{-5}$	$10^{-6}$
$(1+x)^{1/x}$	2.59374	2.70481	2.71692	2.71815	2.71827	2.71828

**例 1 名称**

在例子中，我们的字体可以自定义，包括公式的字号会保持与内容一致。

$$(a+b)^n = \sum_{i=0}^n C_n^i a^i b^{n-i} (n \text{ 为整数}) \quad (11)$$

引用例子同样使用 \autoref，如[例 1](#)。

以下给出一段 Matlab 代码，代码必须有“.m”和“.tex”两个版本，放在 codes 文件夹中。

**Matlab 代码****显示 Command Window 中的代码**

注意方括号如果要出现在行首，必须用大括号括起来，否则会出错。

```
>> 1.2/3.4 + (5.6+7.8)*9 -1
ans = 119.9529
>> 1/exp(1)
ans = 0.3679
>> exp(-1i*pi)+1
ans = 0
```

## 显示 m 文件中的代码

代码必须以.tex 文件格式放在 code 文件夹中的中, 并用 \input 命令导入正文. .tex 代码文件的命名与图片命名相同. 代码前面必须写文件名, 便于读者从网站下载文件

### sample.m

```
1  % 验证二项式定理(非整数幂)
2  u = -3.5;
3  x = 0.6; % |x|<1 使级数收敛
4  N = 100; % 求和项数
5  Coeff = 1; % x^ii 项前面的系数
6  result = 1; % 求和结果
7  for ii = 1:N
8      Coeff = Coeff*(u-ii+1) / ii;
9      result = result + Coeff * x^(ii);
10 end
11 disp('直接计算结果为')
12 format long % 显示全部小数位
13 disp((1+x)^u)
14 disp('求和结果为')
15 disp(result)
16 format short % 恢复默认显示
17 % test keyword color
18 tan; cot; asin; acos; atan; arg; real; imag; sum; mean;
    diff; floor; ceil; mod; sinh; cosh; round; tanh;
    zeros; ones; rand; randn; eye; magic;
```

应用举例 本书格式规范<sup>[48]</sup>

拓展阅读 本书格式规范<sup>[48]</sup>