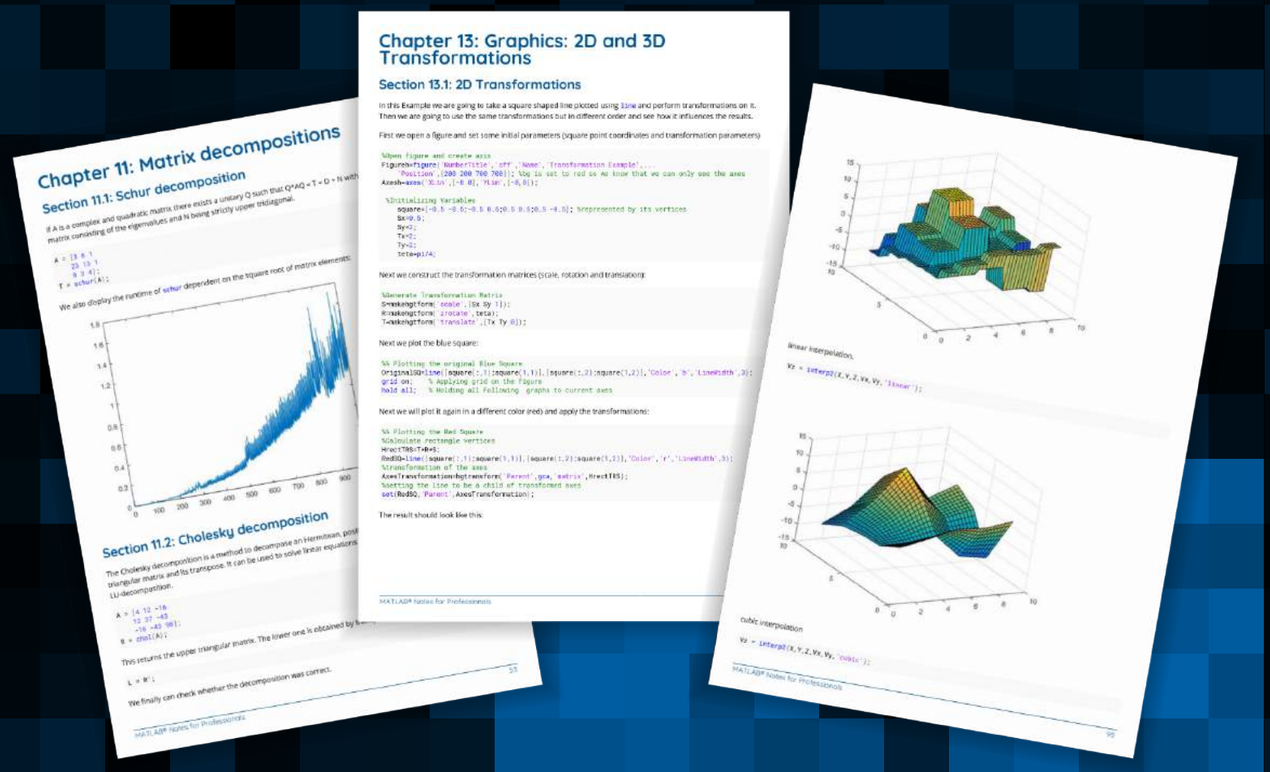
MATLAB®

专业人员笔记

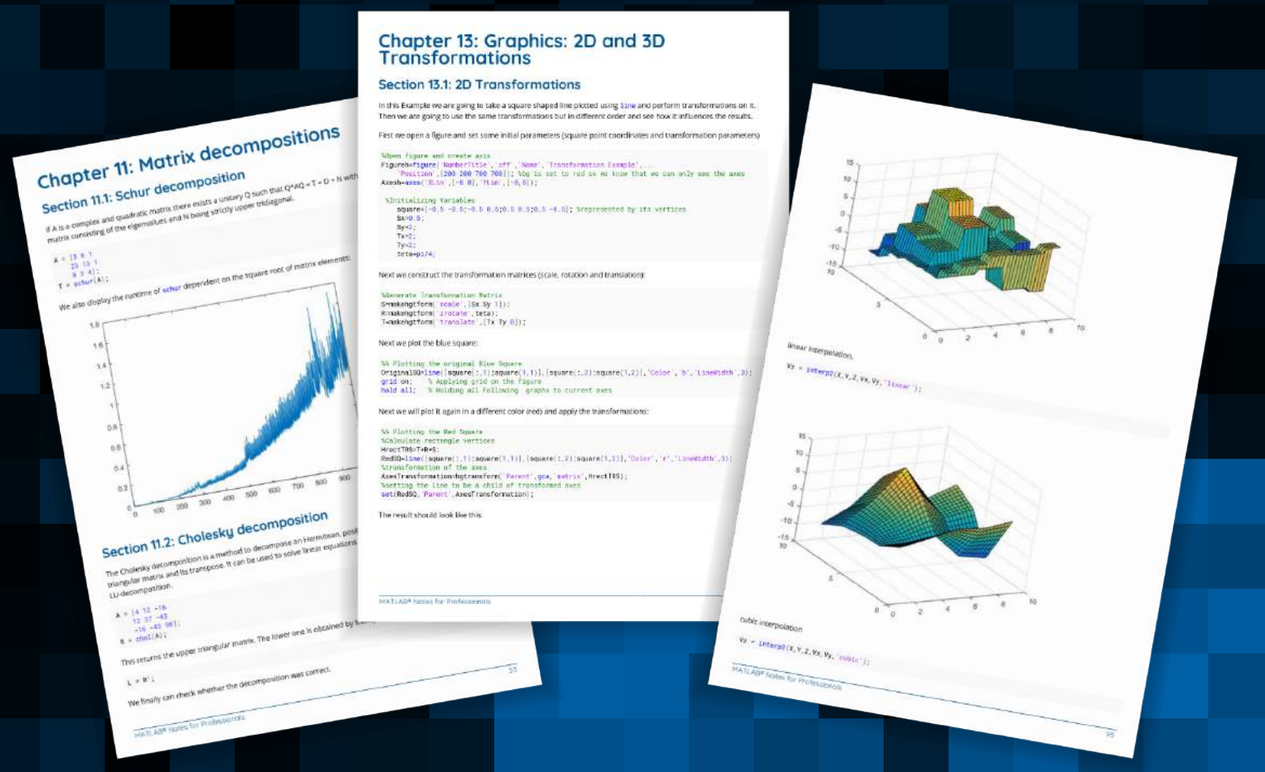


100多页

专业提示和技巧

MATLAB®

Notes for Professionals



100+ pages

of professional hints and tricks

目录

Contents

关于

欢迎免费与任何人分享此PDF，
本书的最新版本可从以下网址下载：
https://goalkicker.com/MATLABBook

这本*MATLAB® 专业人士笔记*是从Stack Overflow Documentation汇编而成，内容由Stack Overflow的优秀人士撰写。
文本内容采用知识共享署名-相同方式共享许可协议发布，详见本书末尾对各章节贡献者的致谢。除非另有说明，图片版权归各自所有者所有。

这是一本非官方的免费书籍，旨在教育用途，与官方MATLAB®组织或公司以及Stack Overflow无关。所有商标和注册商标均为其各自公司所有者的财产

本书中提供的信息不保证正确或准确，使用风险自负

请将反馈和更正发送至web@petercv.com

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
https://goalkicker.com/MATLABBook

This *MATLAB® Notes for Professionals* book is compiled from Stack Overflow Documentation, the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official MATLAB® group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

第1章：MATLAB语言入门

版本	发布	发布日期
1.0		1984-01-01
2		1986-01-01
3		1987-01-01
3.5		1990-01-01
4		1992-01-01
4.2c		1994-01-01
5.0	第8卷	1996-12-01
5.1	第9卷	1997-05-01
5.1.1	R9.1	1997-05-02
5.2	R10	1998-03-01
5.2.1	R10.1	1998-03-02
5.3	R11	1999-01-01
5.3.1	R11.1	1999-11-01
6.0	R12	2000-11-01
6.1	R12.1	2001-06-01
6.5	R13	2002-06-01
6.5.1	R13SP2	2003-01-01
6.5.2	R13SP2	2003-01-02
7	R14	2006-06-01
7.0.4	R14SP1	2004-10-01
7.1	R14SP3	2005-08-01
7.2	R2006a	2006-03-01
7.3	R2006b	2006-09-01
7.4	R2007a	2007-03-01
7.5	R2007b	2007-09-01
7.6	R2008a	2008-03-01
7.7	R2008b	2008-09-01
7.8	R2009a	2009-03-01
7.9	R2009b	2009-09-01
7.10	R2010a	2010-03-01
7.11	R2010b	2010-09-01
7.12	R2011a	2011-03-01
7.13	R2011b	2011-09-01
7.14	R2012a	2012-03-01
8.0	R2012b	2012-09-01
8.1	R2013a	2013-03-01
8.2	R2013b	2013-09-01
8.3	R2014a	2014-03-01
8.4	R2014b	2014-09-01
8.5	R2015a	2015-03-01
8.6	R2015b	2015-09-01

Chapter 1: Getting started with MATLAB Language

Version	Release	Release Date
1.0		1984-01-01
2		1986-01-01
3		1987-01-01
3.5		1990-01-01
4		1992-01-01
4.2c		1994-01-01
5.0	Volume 8	1996-12-01
5.1	Volume 9	1997-05-01
5.1.1	R9.1	1997-05-02
5.2	R10	1998-03-01
5.2.1	R10.1	1998-03-02
5.3	R11	1999-01-01
5.3.1	R11.1	1999-11-01
6.0	R12	2000-11-01
6.1	R12.1	2001-06-01
6.5	R13	2002-06-01
6.5.1	R13SP2	2003-01-01
6.5.2	R13SP2	2003-01-02
7	R14	2006-06-01
7.0.4	R14SP1	2004-10-01
7.1	R14SP3	2005-08-01
7.2	R2006a	2006-03-01
7.3	R2006b	2006-09-01
7.4	R2007a	2007-03-01
7.5	R2007b	2007-09-01
7.6	R2008a	2008-03-01
7.7	R2008b	2008-09-01
7.8	R2009a	2009-03-01
7.9	R2009b	2009-09-01
7.10	R2010a	2010-03-01
7.11	R2010b	2010-09-01
7.12	R2011a	2011-03-01
7.13	R2011b	2011-09-01
7.14	R2012a	2012-03-01
8.0	R2012b	2012-09-01
8.1	R2013a	2013-03-01
8.2	R2013b	2013-09-01
8.3	R2014a	2014-03-01
8.4	R2014b	2014-09-01
8.5	R2015a	2015-03-01
8.6	R2015b	2015-09-01

9.0	R2016a	2016-03-01
9.1	R2016b	2016-09-14
9.2	R2017a	2017-03-08

另见：维基百科上的MATLAB版本历史。

第1.1节：矩阵和数组的索引

MATLAB 允许多种方法来索引（访问）矩阵和数组的元素：

- 下标索引 - 你可以分别指定矩阵每个维度中所需元素的位置。
- 线性索引 - 无论矩阵的维度如何，矩阵都被视为一个向量。这意味着你用一个数字来指定矩阵中的每个位置。
- 逻辑索引 - 你使用一个逻辑矩阵（由true和false值组成的矩阵），该矩阵与要索引的矩阵维度相同，作为掩码来指定返回哪些值。

下面通过以下 3×3 矩阵M作为示例，详细说明这三种方法：

```
>> M = magic(3)

ans =

      8     1     6
      3     5     7
      4     9     2
```

下标索引

访问元素最直接的方法是指定其行列索引。例如，访问第二行第三列的元素：

```
>> M(2, 3)

ans =

     7
```

提供的下标数量与M的维度数量完全匹配（本例中为两个）。

请注意，下标的顺序与数学惯例相同：行索引为第一个。此外，
MATLAB的索引从1开始，而不是像大多数编程语言那样从0开始。

你可以通过为每个坐标传递一个向量而不是单个数字来一次索引多个元素。例如，要获取整个第二行，我们可以指定想要第一、第二和第三列：

```
>> M(2, [1,2,3])

ans =

      3     5     7
```

在MATLAB中，向量[1,2,3]更容易使用冒号运算符创建，即1:3。你也可以在索引中使用它。要选择整行（或整列），MATLAB提供了一个快捷方式，只需指定:。例如，以下代码也将返回整个第二行

9.0	R2016a	2016-03-01
9.1	R2016b	2016-09-14
9.2	R2017a	2017-03-08

See also: MATLAB release history on Wikipedia.

Section 1.1: Indexing matrices and arrays

MATLAB allows for several methods to index (access) elements of matrices and arrays:

- **Subscript indexing** - where you specify the position of the elements you want in each dimension of the matrix separately.
- **Linear indexing** - where the matrix is treated as a vector, no matter its dimensions. That means, you specify each position in the matrix with a single number.
- **Logical indexing** - where you use a logical matrix (and matrix of `true` and `false` values) with the identical dimensions of the matrix you are trying to index as a mask to specify which value to return.

These three methods are now explained in more detail using the following 3-by-3 matrix `M` as an example:

```
>> M = magic(3)

ans =

      8     1     6
      3     5     7
      4     9     2
```

Subscript indexing

The most straight-forward method for accessing an element, is to specify its row-column index. For example, accessing the element on the second row and third column:

```
>> M(2, 3)

ans =

     7
```

The number of subscripts provided exactly matches the number of dimensions `M` has (two in this example).

Note that the order of subscripts is the same as the mathematical convention: row index is the first. Moreover, MATLAB indices **starts with `1`** and **not `0`** like most programming languages.

You can index multiple elements at once by passing a vector for each coordinate instead of a single number. For example to get the entire second row, we can specify that we want the first, second and third columns:

```
>> M(2, [1,2,3])

ans =

      3     5     7
```

In MATLAB, the vector `[1,2,3]` is more easily created using the colon operator, i.e. `1:3`. You can use this in indexing as well. To select an entire row (or column), MATLAB provides a shortcut by allowing you just specify `:`. For example, the following code will also return the entire second row

```
>> M(2, :)

ans =

      3     5     7
```

MATLAB还提供了一个快捷方式，用end关键字指定维度的最后一个元素。end关键字的作用就像该维度最后一个元素的编号一样。因此，如果你想要从第2列到最后一列的所有列，可以写成如下：

```
>> M(2, 2:end)

ans =

      5     7
```

下标索引可能有限制，因为它不允许从不同的列和行中提取单个值；它将提取所有行和列的组合。

```
>> M([2,3], [1,3])
ans =

      3     7
      4     2
```

例如，下标索引不能仅提取元素M(2,1)或M(3,3)。要做到这一点，我们必须考虑线性索引。

线性索引

MATLAB允许你在仅使用一个维度索引时，将n维数组视为一维数组。你可以直接访问第一个元素：

```
>> M(1)

ans =

      8
```

注意，MATLAB中的数组是以列优先顺序存储的，这意味着你访问元素时是先沿着列方向访问。因此，M(2)是第一列的第二个元素，即3，而M(4)将是第二列的第一个元素，即

```
>> M(4)

ans =

      1
```

MATLAB中内置了将下标索引转换为线性索引，反之亦然的函数：`sub2ind` 和 `ind2sub`。你也可以手动将下标（r，c）转换为线性索引，方法是

```
idx = r + (c-1)*size(M,1)
```

要理解这一点，如果我们在第一列，那么线性索引将仅仅是行索引。上述公式对这种情况成立，因为当 `c == 1` 时，`(c-1) == 0`。在接下来的列中，线性索引是行号


```
>> M(2, :)

ans =

      3     5     7
```

MATLAB also provides a shortcut for specifying the last element of a dimension in the form of the `end` keyword. The `end` keyword will work exactly as if it was the number of the last element in that dimension. So if you want all the columns from column 2 to the last column, you can use write the following:

```
>> M(2, 2:end)

ans =

      5     7
```

Subscript indexing can be restrictive as it will not allow to extract single values from different columns and rows; it will extract the combination of all rows and columns.

```
>> M([2,3], [1,3])
ans =

      3     7
      4     2
```

For example subscript indexing cannot extract only the elements `M(2,1)` or `M(3,3)`. To do this we must consider linear indexing.

Linear indexing

MATLAB allows you to treat n-dimensional arrays as one-dimensional arrays when you index using only one dimension. You can directly access the first element:

```
>> M(1)

ans =

      8
```

Note that arrays are stored in column-major order in MATLAB which means that you access the elements by first going down the columns. So `M(2)` is the second element of the first column which is `3` and `M(4)` will be the first element of the second column i.e.

```
>> M(4)

ans =

      1
```

There exist built-in functions in MATLAB to convert subscript indices to linear indices, and vice versa: `sub2ind` and `ind2sub` respectively. You can manually convert the subscripts (r,c) to a linear index by

```
idx = r + (c-1)*size(M,1)
```

To understand this, if we are in the first column then the linear index will simply be the row index. The formula above holds true for this because for `c == 1`, `(c-1) == 0`. In the next columns, the linear index is the row number

加上前面所有列的行数。

注意，`end` 关键字仍然适用，现在指的是数组的最后一个元素，即 `M(end) == M(end, end) == 2`。

你也可以使用线性索引索引多个元素。注意，如果这样做，返回的矩阵将具有与索引向量矩阵相同的形状。

M(2:4) 返回一个行向量，因为 2:4 表示行向量 [2,3,4]：

```
>> M(2:4)

ans =

     3     4     1
```

另一个例子，M([1,2;3,4]) 返回一个 2×2 矩阵，因为 [1,2;3,4] 也是一个 2×2 矩阵。请看下面的代码以验证：

```
>> M([1,2;3,4])

ans =

     8     3
     4     1
```

请注意，仅使用:进行索引总是返回一个列向量：

```
>> M(:)

ans =

     8
     3
     4
     1
     5
     9
     6
     7
     2
```

此示例还说明了 MATLAB 使用线性索引时返回元素的顺序。

逻辑索引

第三种索引方法是使用逻辑矩阵，即仅包含true或false值的矩阵，作为掩码来过滤掉不需要的元素。例如，如果我们想找到所有大于M的元素
5我们可以使用逻辑矩阵

```
>> M > 5

ans =

  1  0  1
  0  0  1
  0  1  0
```


plus all the rows of the previous columns.

Note that the `end` keyword still applies and now refers to the very last element of the array i.e. `M(end) == M(end, end) == 2`.

You can also index multiple elements using linear indexing. Note that if you do that, the returned matrix will have the same shape as the matrix of index vectors.

`M(2:4)` returns a row vector because `2:4` represents the row vector `[2,3,4]`:

```
>> M(2:4)

ans =

     3     4     1
```

As another example, `M([1,2;3,4])` returns a 2-by-2 matrix because `[1,2;3,4]` is a 2-by-2 matrix as well. See the below code to convince yourself:

```
>> M([1,2;3,4])

ans =

     8     3
     4     1
```

Note that indexing with : alone will *always* return a column vector:

```
>> M(:)

ans =

     8
     3
     4
     1
     5
     9
     6
     7
     2
```

This example also illustrates the order in which MATLAB returns elements when using linear indexing.

Logical indexing

The third method of indexing is to use a logical matrix, i.e. a matrix containing only `true` or `false` values, as a mask to filter out the elements you don't want. For example, if we want to find all the elements of `M` that are greater than `5` we can use the logical matrix

```
>> M > 5

ans =

  1  0  1
  0  0  1
  0  1  0
```

索引M并仅返回大于5的值，如下所示：

```
>> M(M > 5)

ans =

        8
        9
        6
        7
```

如果你想让这些数字保持原位（即保持矩阵的形状），那么你可以赋值给逻辑补集

```
>> M(~(M > 5)) = NaN

ans =

    8      NaN    6
    NaN    NaN    7
    NaN    9      Nan
```

我们可以通过使用逻辑索引来简化包含if和for语句的复杂代码块。

采用非向量化（已通过线性索引缩减为单个循环）方式：

```
for elem = 1:numel(M)
  if M(elem) > 5
    M(elem) = M(elem) - 2;
  end
end
```

这可以用逻辑索引缩短为以下代码：

```
idx = M > 5;
M(idx) = M(idx) - 2;
```

甚至更短：

```
M(M > 5) = M(M > 5) - 2;
```

关于索引的更多内容

高维矩阵

上述所有方法都可以推广到n维。如果我们以三维矩阵M3 =rand(3,3,3)为例，那么你可以通过以下写法访问第三维第二个切片的所有行和列

```
>> M(:,:,2)
```

您可以使用线性索引访问第二个切片的第一个元素。线性索引会在遍历完第一个切片的所有行和所有列后，才进入第二个切片。因此，该元素的线性索引是

```
>> M(大小(M,1)*大小(M,2)+1)
```


to index `M` and return only the values that are greater than `5` as follows:

```
>> M(M > 5)

ans =

        8
        9
        6
        7
```

If you wanted these number to stay in place (i.e. keep the shape of the matrix), then you could assign to the logic compliment

```
>> M(~(M > 5)) = NaN

ans =

    8      NaN    6
    NaN    NaN    7
    NaN    9      Nan
```

We can reduce complicated code blocks containing `if` and `for` statements by using logical indexing.

Take the non-vectorized (already shortened to a single loop by using linear indexing):

```
for elem = 1:numel(M)
  if M(elem) > 5
    M(elem) = M(elem) - 2;
  end
end
```

This can be shortened to the following code using logical indexing:

```
idx = M > 5;
M(idx) = M(idx) - 2;
```

Or even shorter:

```
M(M > 5) = M(M > 5) - 2;
```

More on indexing

Higher dimension matrices

All the methods mentioned above generalize into n-dimensions. If we use the three-dimensional matrix `M3 = rand(3,3,3)` as an example, then you can access all the rows and columns of the second slice of the third dimension by writing

```
>> M(:,:,2)
```

You can access the first element of the second slice using linear indexing. Linear indexing will only move on to the second slice after all the rows and all the columns of the first slice. So the linear index for that element is

```
>> M(size(M,1)*size(M,2)+1)
```

事实上，在 MATLAB 中，每个矩阵都是 n 维的：只是大多数其他 n 维的大小都是 1。因此，如果 a = 2，那么 a(1) == 2（正如预期的那样），但 a(1, 1) == 2，a(1, 1, 1)== 2，a(1, 1, 1, ..., 1) == 2，依此类推。这些大小为 1 的"额外"维度称为 单例维度（singleton dimensions）。命令 squeeze 会移除它们，可以使用 permute 来交换维度的顺序（并在需要时引入单例维度）。

一个 n 维矩阵也可以用 m 个下标索引（其中 m<=n）。规则是前 m-1 个下标按常规行为，而最后一个（第 m 个）下标引用剩余的 (n-m+1) 个维度，就像线性索引引用一个 (n-m+1) 维数组一样。下面是一个例子：

```
>> M = reshape(1:24,[2,3,4]);
>> M(1,1)
ans =
     1
>> M(1,10)
ans =
    19
>> M(:,:)
ans =
     1     3     5     7     9    11    13    15    17    19    21    23
     2     4     6     8    10    12    14    16    18    20    22    24
```

返回元素范围

使用下标索引时，如果在多个维度中指定多个元素，MATLAB 会返回每一个可能的坐标对。例如，如果你尝试 M([1,2],[1,3])，MATLAB 会返回M(1,1)和M(2,3)，但它也会返回M(1,3)和M(2,1)。当你想要一组坐标对的元素时，这种行为可能看起来不直观，但考虑一个更大的矩阵，例如A = rand(20)（注意A现在是20行20列），你想获取右上象限。在这种情况下，不用指定该象限中每一个坐标对（这里有100对），只需指定你想要的10行和10列即可

A(1:10, 11:end)。像这样切片矩阵比要求一组坐标对更常见。

如果你确实想要获取一组坐标对，最简单的解决方案是转换为线性索引。考虑这样一个问题：你有一个列索引向量，每一行包含你想要返回的矩阵对应行的列号。例如

```
colIdx = [3;2;1]
```

所以在这种情况下，你实际上想要获取元素(1,3)、(2,2)和(3,1)。使用线性索引：

```
>> colIdx = [3;2;1];
>> rowIdx = 1:length(colIdx);
>> idx = sub2ind(size(M), rowIdx, colIdx);
>> M(idx)

ans =

     6     5     4
```

多次返回一个元素

通过下标和线性索引，你也可以通过重复索引多次返回同一个元素，因此

```
>> M([1,1,1,2,2,2])

ans =
```


In fact, in MATLAB, *every* matrix is n-dimensional: it just happens to be that the size of most of the other n-dimensions are one. So, if `a = 2` then `a(1) == 2` (as one would expect), *but also* `a(1, 1) == 2`, as does `a(1, 1, 1) == 2`, `a(1, 1, 1, ..., 1) == 2` and so on. These "extra" dimensions (of size 1), are referred to as *singleton dimensions*. The command `squeeze` will remove them, and one can use `permute` to swap the order of dimensions around (and introduce singleton dimensions if required).

An n-dimensional matrix can also be indexed using an m subscripts (where m<=n). The rule is that the first m-1 subscripts behave ordinarily, while the last (m'th) subscript references the remaining (n-m+1) dimensions, just as a linear index would reference an (n-m+1) dimensional array. Here is an example:

```
>> M = reshape(1:24,[2,3,4]);
>> M(1,1)
ans =
     1
>> M(1,10)
ans =
    19
>> M(:,:)
ans =
     1     3     5     7     9    11    13    15    17    19    21    23
     2     4     6     8    10    12    14    16    18    20    22    24
```

Returning ranges of elements

With subscript indexing, if you specify more than one element in more than one dimension, MATLAB returns each possible pair of coordinates. For example, if you try M([1,2],[1,3]) MATLAB will return `M(1,1)` and `M(2,3)` but it will **also** return `M(1,3)` and `M(2,1)`. This can seem unintuitive when you are looking for the elements for a list of coordinate pairs but consider the example of a larger matrix, `A = rand(20)` (note `A` is now `20`-by-`20`), where you want to get the top right hand quadrant. In this case instead of having to specify every coordinate pair in that quadrant (and this this case that would be `100` pairs), you just specify the `10` rows and the `10` columns you want so `A(1:10, 11:end)`. *Slicing* a matrix like this is far more common than requiring a list of coordinate pairs.

In the event that you do want to get a list of coordinate pairs, the simplest solution is to convert to linear indexing. Consider the problem where you have a vector of column indices you want returned, where each row of the vector contains the column number you want returned for the *corresponding* row of the matrix. For example

```
colIdx = [3;2;1]
```

So in this case you actually want to get back the elements at `(1,3)`, `(2,2)` and `(3,1)`. So using linear indexing:

```
>> colIdx = [3;2;1];
>> rowIdx = 1:length(colIdx);
>> idx = sub2ind(size(M), rowIdx, colIdx);
>> M(idx)

ans =

     6     5     4
```

Returning an element multiple times

With subscript and linear indexing you can also return an element multiple times by repeating it's index so

```
>> M([1,1,1,2,2,2])

ans =
```

```
       8     8     8     3     3     3
```

你可以用它来复制整行和整列，例如重复第一行和最后一列

```
>> M([1, 1:end], [1:end, end])

ans =

       8     1     6     6
       8     1     6     6
       3     5     7     7
       4     9     2     2
```

更多信息，请参见 here。

第1.2节：匿名函数和函数句柄

基础知识

匿名函数是 MATLAB 语言中的一个强大工具。它们是局部存在的函数，也就是说：存在于当前工作区中。然而，它们不像常规函数那样存在于 MATLAB 路径上，例如 m 文件中。因此它们被称为匿名函数，尽管它们可以像工作区中的变量一样有一个名称。

@ 运算符

使用 `@` 运算符来创建匿名函数和函数句柄。例如，创建一个指向 `sin` 函数（正弦函数）的句柄，并将其用作 `f`：

```
>> f = @sin
f =
@sin
```

现在 f 是指向 sin 函数的句柄。就像现实生活中门把手是使用门的一种方式一样，函数句柄是使用函数的一种方式。使用 f 时，参数传递给它，就像传递给 sin 函数一样：

```
>> f(pi/2)
ans =
     1
```

f 接受 sin 函数接受的任何输入参数。如果 sin 是一个接受零输入参数的函数（它不是，但其他函数是，例如 peaks 函数），则可以使用 f() 来调用它而不带输入参数。

自定义匿名函数
单变量匿名函数

为现有函数创建一个句柄（如上例中的 sin）显然没有什么用处。在该例中，这有点多余。然而，创建执行自定义操作的匿名函数是有用的，否则这些操作需要重复多次或为此创建单独的函数。作为一个接受一个变量输入的自定义匿名函数的例子，计算信号的正弦平方与余弦平方之和：

```
>> f = @(x) sin(x)+cos(x).^2
f =
@(x)sin(x)+cos(x).^2
```



```
       8     8     8     3     3     3
```

You can use this to duplicate entire rows and column for example to repeat the first row and last column

```
>> M([1, 1:end], [1:end, end])

ans =

       8     1     6     6
       8     1     6     6
       3     5     7     7
       4     9     2     2
```

For more information, see here.

Section 1.2: Anonymous functions and function handles

Basics

Anonymous functions are a powerful tool of the MATLAB language. They are functions that exist locally, that is: in the current workspace. However, they do not exist on the MATLAB path like a regular function would, e.g. in an m-file. That is why they are called anonymous, although they can have a name like a variable in the workspace.

The @ operator

Use the `@` operator to create anonymous functions and function handles. For example, to create a handle to the `sin` function (sine) and use it as `f`:

```
>> f = @sin
f =
    @sin
```

Now `f` is a handle to the `sin` function. Just like (in real life) a door handle is a way to use a door, a function handle is a way to use a function. To use `f`, arguments are passed to it as if it were the `sin` function:

```
>> f(pi/2)
ans =
     1
```

`f` accepts any input arguments the `sin` function accepts. If `sin` would be a function that accepts zero input arguments (which it does not, but others do, e.g. the `peaks` function), `f()` would be used to call it without input arguments.

Custom anonymous functions
Anonymous functions of one variable

It is not obviously useful to create a handle to an existing function, like `sin` in the example above. It is kind of redundant in that example. However, it is useful to create anonymous functions that do custom things that otherwise would need to be repeated multiple times or created a separate function for. As an example of a custom anonymous function that accepts one variable as its input, sum the sine and cosine squared of a signal:

```
>> f = @(x) sin(x)+cos(x).^2
f =
    @(x)sin(x)+cos(x).^2
```

现在f接受一个名为 x 的输入参数。这个参数是通过紧跟在@
操作符后面的括号(...)指定的。函数f现在是一个关于 x 的匿名函数：f(x)。通过传递一个 x 的值给f来使用它：

```
>> f(pi)
ans =
    1.0000
```

也可以将一个数值向量或变量传递给f，只要它们在f中被有效使用：

```
>> f(1:3) % 将向量传递给 f
ans =
    1.1334    1.0825    1.1212
>> n = 5:7;
>> f(n) % 将 n 传递给 f
ans =
-0.8785    0.6425    1.2254
```

多变量匿名函数

同样，可以创建接受多个变量的匿名函数。下面是一个接受三个变量的匿名函数示例：

```
>> f = @(x,y,z) x.^2 + y.^2 - z.^2
f =
@(x,y,z)x.^2+y.^2-z.^2
>> f(2,3,4)
ans =
-3
```

匿名函数的参数化

工作区中的变量可以在匿名函数的定义中使用，这称为参数化。
例如，要在匿名函数中使用常量 c = 2：

```
>> c = 2;
>> f = @(x) c*x
f =
@(x)c*x
>> f(3)
ans =
     6
```

f(3) 使用变量 c 作为参数与提供的 x 相乘。请注意，如果此时 c 的值被设置为不同的值，那么调用 f(3) 时，结果将不会不同。 c 的值是 at
匿名函数创建时间：

```
>> c = 2;
>> f = @(x) c*x;
>> f(3)
ans =
     6
>> c = 3;
>> f(3)
ans =
     6
```

匿名函数的输入参数不引用工作区变量

请注意，使用工作区变量名作为匿名函数的输入参数之一

Now f accepts one input argument called x. This was specified using parentheses (...) directly after the @ operator. f now is an anonymous function of x: f(x). It is used by passing a value of x to f:

```
>> f(pi)
ans =
    1.0000
```

A vector of values or a variable can also be passed to f, as long as they are used in a valid way within f:

```
>> f(1:3) % pass a vector to f
ans =
    1.1334    1.0825    1.1212
>> n = 5:7;
>> f(n) % pass n to f
ans =
   -0.8785    0.6425    1.2254
```

Anonymous functions of more than one variable

In the same fashion anonymous functions can be created to accept more than one variable. An example of an anonymous function that accepts three variables:

```
>> f = @(x,y,z) x.^2 + y.^2 - z.^2
f =
    @(x,y,z)x.^2+y.^2-z.^2
>> f(2,3,4)
ans =
    -3
```

Parameterizing anonymous functions

Variables in the workspace can be used within the definition of anonymous functions. This is called parameterizing. For example, to use a constant c = 2 in an anonymous function:

```
>> c = 2;
>> f = @(x) c*x
f =
    @(x)c*x
>> f(3)
ans =
     6
```

f(3) used the variable c as a parameter to multiply with the provided x. Note that if the value of c is set to something different at this point, then f(3) is called, the result would **not** be different. The value of c is the value *at the time of creation* of the anonymous function:

```
>> c = 2;
>> f = @(x) c*x;
>> f(3)
ans =
     6
>> c = 3;
>> f(3)
ans =
     6
```

Input arguments to an anonymous function do not refer to workspace variables

Note that using the name of variables in the workspace as one of the input arguments of an anonymous function

(即使用 @(...)) 不会 使用 这些变量的值。相反，它们被视为匿名函数作用域内的不同变量，也就是说：匿名函数有自己的私有工作区，输入变量永远不会引用主工作区的变量。主工作区和匿名函数的工作区彼此不了解对方的内容。以下示例说明这一点：

```
>> x = 3 % 主工作区中的 x
x =
     3
>> f = @(x) x+1; % 这里的 x 指的是私有的 x 变量
>> f(5)
ans =
     6
>> x
x =
     3
```

主工作区中的 x 的值未在 f 内使用。此外，主工作区中的 x 保持不变。
在 f 的作用域内，@ 操作符后括号中的变量名与主工作区变量是独立的。

匿名函数存储在变量中

匿名函数（或者更准确地说，指向匿名函数的函数句柄）像当前工作区中的其他值一样存储：存储在变量中（如上所示）、存储在单元数组中（{@(x)x.^2,@(x)x+1}），甚至存储在属性中（如交互式图形的 h.ButtonDownFcn）。这意味着匿名函数可以像其他值一样被处理。存储在变量中时，它在当前工作区有一个名称，可以像存储数字的变量一样被修改和清除。

换句话说：函数句柄（无论是@sin形式还是匿名函数）只是一个值，可以存储在变量中，就像数值矩阵一样。

高级用法
将函数句柄传递给其他函数

由于函数句柄被视为变量，它们可以作为输入参数传递给接受函数句柄的函数。

示例：在一个 m 文件中创建了一个函数，该函数接受一个函数句柄和一个标量数字。然后通过传递3调用该函数句柄，并将标量数字加到结果上。最终返回结果。

funHandleDemo.m 的内容：

```
函数 y = funHandleDemo (fun, x)
y = fun(3);
y = y + x;
```

将其保存到路径中的某个位置，例如 MATLAB 当前文件夹。现在 funHandleDemo 可以按如下方式使用，例如：

```
>> f = @(x) x^2; % 一个匿名函数
>> y = funHandleDemo(f, 10) % 将 f 和一个标量传递给 funHandleDemo
y =
    19
```

可以将另一个已存在函数的句柄传递给 funHandleDemo：

(i.e., using `@(...)`) will **not** use those variables' values. Instead, they are treated as different variables within the scope of the anonymous function, that is: the anonymous function has its private workspace where the input variables never refer to the variables from the main workspace. The main workspace and the anonymous function's workspace do not know about each other's contents. An example to illustrate this:

```
>> x = 3 % x in main workspace
x =
     3
>> f = @(x) x+1; % here x refers to a private x variable
>> f(5)
ans =
     6
>> x
x =
     3
```

The value of `x` from the main workspace is not used within `f`. Also, in the main workspace `x` was left untouched. Within the scope of `f`, the variable names between parentheses after the `@` operator are independent from the main workspace variables.

Anonymous functions are stored in variables

An anonymous function (or, more precisely, the function handle pointing at an anonymous function) is stored like any other value in the current workspace: In a variable (as we did above), in a cell array (`{@(x)x.^2,@(x)x+1}`), or even in a property (like `h.ButtonDownFcn` for interactive graphics). This means the anonymous function can be treated like any other value. When storing it in a variable, it has a name in the current workspace and can be changed and cleared just like variables holding numbers.

Put differently: A function handle (whether in the `@sin` form or for an anonymous function) is simply a value that can be stored in a variable, just like a numerical matrix can be.

Advanced use
Passing function handles to other functions

Since function handles are treated like variables, they can be passed to functions that accept function handles as input arguments.

An example: A function is created in an m-file that accepts a function handle and a scalar number. It then calls the function handle by passing 3 to it and then adds the scalar number to the result. The result is returned.

Contents of `funHandleDemo.m`:

```
function y = funHandleDemo(fun,x)
y = fun(3);
y = y + x;
```

Save it somewhere on the path, e.g. in MATLAB's current folder. Now `funHandleDemo` can be used as follows, for example:

```
>> f = @(x) x^2; % an anonymous function
>> y = funHandleDemo(f,10) % pass f and a scalar to funHandleDemo
y =
    19
```

The handle of another existing function can be passed to `funHandleDemo`:

```
>> y = funHandleDemo(@sin, -5)
y =
-4.8589
```

注意 @sin 是快速访问 sin 函数的一种方式，无需先用 f = @sin 将其存储在变量中。

使用 bsxfun、cellfun 及类似函数与匿名函数MATLAB 有一些内置函数接受匿

名函数作为输入。这是一种用最少代码行数执行大量计算的方法。例如 bsxfun，它执行逐元素的二元操作，即：对两个向量或矩阵逐元素应用函数。

通常，这需要使用for循环，通常需要预分配以提高速度。使用bsxfun可以加快这个过程。以下示例使用ic和toc两个函数来计时代码运行时间。它计算每个矩阵元素与矩阵列均值的差值。

```
A = rand(50); % 50×50的随机值矩阵，值在0到1之间

% 方法1：慢且代码行数多
tic
meanA = mean(A); % 每列的均值：一个行向量
% 为了速度预分配结果，去掉这行性能会更差
result = zeros(size(A));
for j = 1:size(A,1)
    result(j,:) = A(j,:) - meanA;
end
toc
clear result % 确保方法2创建自己的结果

% 方法2：快速且只有一行代码
tic
result = bsxfun(@minus,A,mean(A));
toc
```

运行上述示例会产生两个输出：

```
用时为0.015153秒。
耗时为0.007884秒。
```

这些行来自oc函数，它们打印自上次调用ic函数以来经过的时间。

bsxfun调用将第一个输入参数中的函数应用于另外两个输入参数。@minus是与减号操作相同操作的长名称。也可以指定不同的匿名函数或函数句柄（@）给任何其他函数，只要它接受A和mean(A)作为输入以生成有意义的结果。

特别是对于大型矩阵中的大量数据，bsxfun可以大大加快处理速度。它还使代码看起来更简洁，尽管对于不熟悉MATLAB或bsxfun的人来说可能更难理解。（注意，在MATLAB R2016a及以后版本，许多以前使用bsxfun的操作不再需要它们；A-mean(A)可以直接使用，有时甚至更快。）

第1.3节：矩阵和数组

在MATLAB中，最基本的数据类型是数值数组。它可以是标量、一维向量、二维矩阵或N维多维数组。


```
>> y = funHandleDemo(@sin,-5)
y =
   -4.8589
```

Notice how `@sin` was a quick way to access the `sin` function without first storing it in a variable using `f = @sin`.

Using `bsxfun`, `cellfun` and similar functions with anonymous functions

MATLAB has some built-in functions that accept anonymous functions as an input. This is a way to perform many calculations with a minimal number of lines of code. For example `bsxfun`, which performs element-by-element binary operations, that is: it applies a function on two vectors or matrices in an element-by-element fashion. Normally, this would require use of `for`-loops, which often requires preallocation for speed. Using `bsxfun` this process is sped up. The following example illustrates this using `tic` and `toc`, two functions that can be used to time how long code takes. It calculates the difference of every matrix element from the matrix column mean.

```
A = rand(50); % 50-by-50 matrix of random values between 0 and 1

% method 1: slow and lots of lines of code
tic
meanA = mean(A); % mean of every matrix column: a row vector
% pre-allocate result for speed, remove this for even worse performance
result = zeros(size(A));
for j = 1:size(A,1)
    result(j,:) = A(j,:) - meanA;
end
toc
clear result % make sure method 2 creates its own result

% method 2: fast and only one line of code
tic
result = bsxfun(@minus,A,mean(A));
toc
```

Running the example above results in two outputs:

```
Elapsed time is 0.015153 seconds.
Elapsed time is 0.007884 seconds.
```

These lines come from the `toc` functions, which print the elapsed time since the last call to the `tic` function.

The `bsxfun` call applies the function in the first input argument to the other two input arguments. `@minus` is a long name for the same operation as the minus sign would do. A different anonymous function or handle (`@`) to any other function could have been specified, as long as it accepts `A` and `mean(A)` as inputs to generate a meaningful result.

Especially for large amounts of data in large matrices, `bsxfun` can speed up things a lot. It also makes code look cleaner, although it might be more difficult to interpret for people who don't know MATLAB or `bsxfun`. (Note that in MATLAB R2016a and later, many operations that previously used `bsxfun` no longer need them; `A-mean(A)` works directly and can in some cases be even faster.)

Section 1.3: Matrices and Arrays

In MATLAB, the most basic data type is the numeric array. It can be a scalar, a 1-D vector, a 2-D matrix, or an N-D multidimensional array.

```
% 一个1×1的标量值
x = 1;
```

要创建行向量，在方括号内输入元素，元素之间用空格或逗号分隔：

```
% 一个1×4的行向量
v = [1, 2, 3, 4];
v = [1 2 3 4];
```

要创建列向量，用分号分隔元素：

```
% 一个4行1列的列向量
v = [1; 2; 3; 4];
```

要创建一个矩阵，我们像之前一样输入行，行之间用分号分隔：

```
% 一个2行4列的矩阵
M = [1 2 3 4; 5 6 7 8];

% 一个4行2列的矩阵
M = [1 2; ...
     4 5; ...
     6 7; ...
     8 9];
```

注意，不能创建行列大小不等的矩阵。所有行必须长度相同，所有列也必须长度相同：

```
% 一个行列大小不等的矩阵
M = [1 2 3 ; 4 5 6 7]; % 这是无效的，会返回错误

% 另一个行列大小不等的矩阵
M = [1 2 3; ...
     4   5; ...
     6 7 8; ...
     9   10];     % 这不是有效的语法，会返回错误
```

要转置向量或矩阵，我们使用.'运算符，或者使用'运算符来取其厄米共轭，即转置的复共轭。对于实矩阵，这两者是相同的：

```
% 创建一个行向量并转置成列向量
v = [1 2 3 4].';                % v 等于 [1; 2; 3; 4];

% 创建一个 2 行 4 列的矩阵并转置成 4 行 2 列的矩阵
M = [1 2 3 4; 5 6 7 8].';       % M 等于 [1 5; 2 6; 3 7; 4 8]

% 转置存储在变量中的向量或矩阵
A = [1 2; 3 4];
B = A.';                        % B 等于 [1 3; 2 4]
```

对于超过二维的数组，没有直接的语言语法可以字面输入它们。相反，我们必须使用函数来构造它们（例如 ones、zeros、rand），或者通过操作其他数组（使用函数如 cat、reshape、permute）。一些示例：

```
% 一个 5 行 2 列 4 层 3 维数组（4 维）
arr = ones(5, 2, 4, 3);
```



```
% a 1-by-1 scalar value
x = 1;
```

To create a row vector, enter the elements inside brackets, separated by spaces or commas:

```
% a 1-by-4 row vector
v = [1, 2, 3, 4];
v = [1 2 3 4];
```

To create a column vector, separate the elements with semicolons:

```
% a 4-by-1 column vector
v = [1; 2; 3; 4];
```

To create a matrix, we enter the rows as before separated by semicolons:

```
% a 2 row-by-4 column matrix
M = [1 2 3 4; 5 6 7 8];

% a 4 row-by-2 column matrix
M = [1 2; ...
     4 5; ...
     6 7; ...
     8 9];
```

Notice you cannot create a matrix with unequal row / column size. All rows must be the same length, and all columns must be the same length:

```
% an unequal row / column matrix
M = [1 2 3 ; 4 5 6 7]; % This is not valid and will return an error

% another unequal row / column matrix
M = [1 2 3; ...
     4   5; ...
     6 7 8; ...
     9   10];     % This is not valid and will return an error
```

To transpose a vector or a matrix, we use the `.'`-operator, or the `'` operator to take its Hermitian conjugate, which is the complex conjugate of its transpose. For real matrices, these two are the same:

```
% create a row vector and transpose it into a column vector
v = [1 2 3 4].';              % v is equal to [1; 2; 3; 4];

% create a 2-by-4 matrix and transpose it to get a 4-by-2 matrix
M = [1 2 3 4; 5 6 7 8].';     % M is equal to [1 5; 2 6; 3 7; 4 8]

% transpose a vector or matrix stored as a variable
A = [1 2; 3 4];
B = A.';                      % B is equal to [1 3; 2 4]
```

For arrays of more than two-dimensions, there is no direct language syntax to enter them literally. Instead we must use functions to construct them (such as `ones`, `zeros`, `rand`) or by manipulating other arrays (using functions such as `cat`, `reshape`, `permute`). Some examples:

```
% a 5-by-2-by-4-by-3 array (4-dimensions)
arr = ones(5, 2, 4, 3);
```

```
% 一个 2 行 3 列 2 层数组（3 维）
arr = cat(3, [1 2 3; 4 5 6], [7 8 9; 0 1 2]);

% 一个 5 行 4 列 3 层 2 维数组（4 维）
arr = reshape(1:120, [5 4 3 2]);
```

第1.4节：单元数组

同一类的元素通常可以连接成数组（少数例外，如函数句柄）。数值标量，默认属于double类，可以存储在矩阵中。

```
>> A = [1, -2, 3.14, 4/5, 5^6; pi, inf, 7/0, nan, log(0)]
A =
   1.0e+04 *
    0.0001   -0.0002    0.0003    0.0001    1.5625
    0.0003       Inf       Inf       NaN      -Inf
```

字符，在MATLAB中属于char类，也可以用类似的语法存储在数组中。这样的数组类似于许多其他编程语言中的字符串。

```
>> s = ['MATLAB ','is ','fun']
s =
MATLAB is fun
```

请注意，尽管它们都使用了括号[和]，但结果类别是不同的。因此，可以对它们进行的操作也不同。

```
>> whos
名称      大小            字节数  类     属性

  A         2x5                80  double
  s         1x13               26  char
```

实际上，数组 s 并不是字符串 'MATLAB '、'is ' 和 'fun' 的数组，它只是一个字符串——一个包含13个字符的数组。如果用以下任意一种方式定义，结果都是一样的：

```
>> s = ['MAT','LAB ','is f','u','n'];
>> s = ['M','A','T','L','A','B,' ','i','s',' ','f','u','n'];
```

普通的 MATLAB 向量不允许你存储不同类的变量混合，或者几个不同的字符串。这时 cell 数组就很有用。它是一个单元格数组，每个单元格可以包含某个 MATLAB 对象，如果需要，每个单元格中的类可以不同。使用大括号 { 和 } 将元素包裹起来以存储到单元格数组中。

```
>> C = {A; s}
C =
    [2x5 double]
    'MATLAB is fun'
>> whos C
名称      大小            字节数  类     属性

  C         2x1              330  cell
```

任何类的标准 MATLAB 对象都可以存储在单元数组中。请注意，单元数组需要更多的内存来存储其内容。

访问单元内容使用花括号 { 和 }。


```
% a 2-by-3-by-2 array (3-dimensions)
arr = cat(3, [1 2 3; 4 5 6], [7 8 9; 0 1 2]);

% a 5-by-4-by-3-by-2 (4-dimensions)
arr = reshape(1:120, [5 4 3 2]);
```

Section 1.4: Cell arrays

Elements of the same class can often be concatenated into arrays (with a few rare exceptions, e.g. function handles). Numeric scalars, by default of class `double`, can be stored in a matrix.

```
>> A = [1, -2, 3.14, 4/5, 5^6; pi, inf, 7/0, nan, log(0)]
A =
   1.0e+04 *
    0.0001   -0.0002    0.0003    0.0001    1.5625
    0.0003       Inf       Inf       NaN      -Inf
```

Characters, which are of class `char` in MATLAB, can also be stored in array using similar syntax. Such an array is similar to a string in many other programming languages.

```
>> s = ['MATLAB ','is ','fun']
s =
MATLAB is fun
```

Note that despite both of them are using brackets [and], the result classes are different. Therefore the operations that can be done on them are also different.

```
>> whos
  Name      Size            Bytes  Class     Attributes

  A         2x5                80  double
  s         1x13               26  char
```

In fact, the array s is not an array of the strings `'MATLAB '`,`'is '`, and `'fun'`, it is just one string - an array of 13 characters. You would get the same results if it were defined by any of the following:

```
>> s = ['MAT','LAB ','is f','u','n'];
>> s = ['M','A','T','L','A','B,' ','i','s',' ','f','u','n'];
```

A regular MATLAB vector does not let you store a mix of variables of different classes, or a few different strings. This is where the `cell` array comes in handy. This is an array of cells that each can contain some MATLAB object, whose class can be different in every cell if needed. Use curly braces { and } around the elements to store in a cell array.

```
>> C = {A; s}
C =
    [2x5 double]
    'MATLAB is fun'
>> whos C
  Name      Size            Bytes  Class    Attributes

  C         2x1               330  cell
```

Standard MATLAB objects of any classes can be stored together in a cell array. Note that cell arrays require more memory to store their contents.

Accessing the contents of a cell is done using curly braces { and }.

```
>> C{1}
ans =
   1.0e+04 *
    0.0001   -0.0002    0.0003    0.0001    1.5625
    0.0003       Inf       Inf       NaN      -Inf
```

请注意，C(1) 与 C{1} 不同。后者返回单元的内容（在我们的例子中类为 double），而前者返回一个单元数组，它是 C 的子数组。类似地，如果 D 是一个 10 行 5 列的单元数组，那么 D(4:8,1:3) 会返回 D 的一个大小为 5 行 3 列且类为 cell 的子数组。语法 C{1:2} 不会返回单个对象，而是返回两个不同的对象（类似于 MATLAB 函数的多个返回值）：

```
>> [x,y] = C{1:2}
x =
                        1-2                      3.14
  0.8                 15625
          3.14159265358979                       Inf                       Inf
  NaN                      -Inf
y =
MATLAB is fun
```

第1.5节：你好，世界

在MATLAB编辑器中打开一个新的空白文档（在较新版本的MATLAB中，通过选择工具条的“主页”选项卡，点击“新建脚本”来实现）。创建新脚本的默认快捷键是 `Ctrl-n`.

或者，输入edit myscriptname.m 将打开文件myscriptname.m进行编辑，如果该文件在MATLAB路径中不存在，则会提示创建该文件。

在编辑器中，输入以下内容：

```
disp('Hello, World!');
```

选择工具条的“编辑器”选项卡，点击“另存为”。将文档保存到当前目录下，文件名为 helloworld.m。保存未命名文件时，会弹出对话框让你命名文件。

在MATLAB命令窗口中，输入以下内容：

```
>> helloworld
```

你应该在 MATLAB 命令窗口中看到以下响应：

```
你好，世界！
```

我们看到在命令窗口中，可以输入我们编写的函数或脚本文件的名称，或者 MATLAB 自带的文件名来运行它们。

这里，我们运行了“helloworld”脚本。注意，输入扩展名（.m）是不必要的。脚本文件中的指令由 MATLAB 执行，这里使用 disp函数打印“你好，世界！”。

脚本文件可以这样编写，以保存一系列命令以供以后（重新）使用。

第1.6节：脚本和函数

MATLAB 代码可以保存在 m 文件中以便重用。m 文件具有.m扩展名，并自动与 MATLAB 关联。


```
>> C{1}
ans =
   1.0e+04 *
    0.0001   -0.0002    0.0003    0.0001    1.5625
    0.0003       Inf       Inf       NaN      -Inf
```

Note that `C(1)` is different from `C{1}`. Whereas the latter returns the cell's content (and has class `double` in out example), the former returns a cell array which is a sub-array of `C`. Similarly, if `D` were an 10 by 5 cell array, then `D(4:8,1:3)` would return a sub-array of `D` whose size is 5 by 3 and whose class is `cell`. And the syntax `C{1:2}` does not have a single returned object, but rather it returns 2 different objects (similar to a MATLAB function with multiple return values):

```
>> [x,y] = C{1:2}
x =
                        1                       -2                     3.14
  0.8                 15625
          3.14159265358979                      Inf                      Inf
  NaN                      -Inf
y =
MATLAB is fun
```

Section 1.5: Hello World

Open a new blank document in the MATLAB Editor (in recent versions of MATLAB, do this by selecting the Home tab of the toolstrip, and clicking on New Script). The default keyboard shortcut to create a new script is `Ctrl-n`.

Alternatively, typing `edit myscriptname.m` will open the file `myscriptname.m` for editing, or offer to create the file if it does not exist on the MATLAB path.

In the editor, type the following:

```
disp('Hello, World!');
```

Select the Editor tab of the toolstrip, and click Save As. Save the document to a file in the current directory called `helloworld.m`. Saving an untitled file will bring up a dialog box to name the file.

In the MATLAB Command Window, type the following:

```
>> helloworld
```

You should see the following response in the MATLAB Command Window:

```
Hello, World!
```

We see that in the Command Window, we are able to type the names of functions or script files that we have written, or that are shipped with MATLAB, to run them.

Here, we have run the 'helloworld' script. Notice that typing the extension (`.m`) is unnecessary. The instructions held in the script file are executed by MATLAB, here printing 'Hello, World!' using the `disp` function.

Script files can be written in this way to save a series of commands for later (re)use.

Section 1.6: Scripts and Functions

MATLAB code can be saved in m-files to be reused. m-files have the `.m` extension which is automatically associated

m 文件可以包含脚本或函数。

脚本

脚本只是按预定顺序执行一系列 MATLAB 命令的程序文件。

脚本不接受输入，也不返回输出。从功能上讲，脚本等同于直接在 MATLAB 命令窗口中输入命令并能够重放这些命令。

脚本示例：

```
length = 10;
width = 3;
area = length * width;
```

该脚本将在当前工作区定义length、width和area，值分别为10、3和30。

如前所述，上述脚本在功能上等同于直接在命令窗口中输入相同的命令。

```
>> length = 10;
>> width = 3;
>> area = length * width;
```

函数

与脚本相比，函数更加灵活和可扩展。与脚本不同，函数可以接受输入并返回输出给调用者。函数有自己的工作区，这意味着函数的内部操作不会改变调用者的变量。

所有函数都采用相同的头部格式定义：

```
function [output] = myFunctionName(input)
```

function关键字开始每个函数头。接下来是输出列表。输出列表也可以是以逗号分隔的返回变量列表。

```
function [a, b, c] = myFunctionName(input)
```

接下来是用于调用的函数名。通常与文件名相同。例如，我们会将此函数保存为myFunctionName.m。

函数名后面是输入列表。与输出类似，这也可以是以逗号分隔的列表。

```
function [a, b, c] = myFunctionName(x, y, z)
```

我们可以将之前的示例脚本重写为如下可重用的函数：

```
function [area] = calcRecArea(length, width)
   area = length * width;
end
```

我们可以从其他函数或脚本文件中调用函数。下面是上述函数在脚本文件中被调用的示例。

with MATLAB. An m-file can contain either a script or functions.

Scripts

Scripts are simply program files that execute a series of MATLAB commands in a predefined order.

Scripts do not accept input, nor do scripts return output. Functionally, scripts are equivalent to typing commands directly into the MATLAB command window and being able to replay them.

An example of a script:

```
length = 10;
width = 3;
area = length * width;
```

This script will define `length`, `width`, and `area` in the current workspace with the value `10`, `3`, and `30` respectively.

As stated before, the above script is functionally equivalent to typing the same commands directly into the command window.

```
>> length = 10;
>> width = 3;
>> area = length * width;
```

Functions

Functions, when compared to scripts, are much more flexible and extensible. Unlike scripts, functions can accept input and return output to the caller. A function has its own workspace, this means that internal operations of the functions will not change the variables from the caller.

All functions are defined with the same header format:

```
function [output] = myFunctionName(input)
```

The `function` keyword begins every function header. The list of outputs follows. The list of outputs can also be a comma separated list of variables to return.

```
function [a, b, c] = myFunctionName(input)
```

Next is the name of the function that will be used for calling. This is generally the same name as the filename. For example, we would save this function as `myFunctionName.m`.

Following the function name is the list of inputs. Like the outputs, this can also be a comma separated list.

```
function [a, b, c] = myFunctionName(x, y, z)
```

We can rewrite the example script from before as a reusable function like the following:

```
function [area] = calcRecArea(length, width)
   area = length * width;
end
```

We can call functions from other functions, or even from script files. Here is an example of our above function being used in a script file.

```
l = 100;
w = 20;
a = calcRecArea(l, w);
```

如前所述，我们在工作区创建了l、w和a，分别赋值为100、20和2000。

第1.7节：自助帮助

MATLAB自带许多内置脚本和函数，范围从简单的乘法到图像
识别工具箱。要获取你想使用的函数的信息，请在
命令行输入：help functionname。以help函数为例。

可以通过输入以下命令获取使用方法：

```
>> help help
```

在命令窗口中。这将返回函数help的用法信息。如果你查找的信息仍不清楚，可以尝试该函数的documentation页面。只需输入：

```
>> doc help
```

在命令窗口中。这将打开函数help的可浏览文档页面，提供你理解“help”如何工作的所有信息。

此方法适用于所有内置函数和符号。

在开发自己的函数时，可以通过在函数文件顶部或函数声明之后添加注释，让函数拥有自己的帮助部分。

一个简单函数的示例 multiplyby2，保存在文件 multiplyby2.m 中

```
function [prod]=multiplyby2(num)
% 函数 MULTIPLYBY2 接受一个数值矩阵 NUM 并返回输出 PROD
% 使所有数字都乘以 2

    prod=num*2;
end
```

或者

```
% 函数 MULTIPLYBY2 接受一个数值矩阵 NUM 并返回输出 PROD
% 使所有数字都乘以 2

function [prod]=multiplyby2(num)
    prod=num*2;
end
```

当你在写完代码数周、数月甚至数年后重新使用时，这非常有用。

help 和 doc 函数提供了大量信息，学习如何使用这些功能将帮助你快速进步并高效使用 MATLAB。

第1.8节：数据类型

MATLAB 中有16 种基本数据类型，或称为类。每种类都是以矩阵或
数组的形式存在。除函数句柄外，该矩阵或数组的最小尺寸为 0×0，并且可以增长到


```
l = 100;
w = 20;
a = calcRecArea(l, w);
```

As before, we create `l`, `w`, and `a` in the workspace with the values of `100`, `20`, and `2000` respectively.

Section 1.7: Helping yourself

MATLAB comes with many built-in scripts and functions which range from simple multiplication to image recognition toolboxes. In order to get information about a function you want to use type: `help functionname` in the command line. Let's take the `help` function as an example.

Information on how to use it can be obtained by typing:

```
>> help help
```

in the command window. This will return information of the usage of function `help`. If the information you are looking for is still unclear you can try the **documentation** page of the function. Simply type:

```
>> doc help
```

in the command window. This will open the browsable documentation on the page for function `help` providing all the information you need to understand how the 'help' works.

This procedure works for all built-in functions and symbols.

When developing your own functions you can let them have their own help section by adding comments at the top of the function file or just after the function declaration.

Example for a simple function `multiplyby2` saved in file `multiplyby2.m`

```
function [prod]=multiplyby2(num)
% function MULTIPLYBY2 accepts a numeric matrix NUM and returns output PROD
% such that all numbers are multiplied by 2

    prod=num*2;
end
```

or

```
% function MULTIPLYBY2 accepts a numeric matrix NUM and returns output PROD
% such that all numbers are multiplied by 2

function [prod]=multiplyby2(num)
    prod=num*2;
end
```

This is very useful when you pick up your code weeks/months/years after having written it.

The `help` and `doc` function provide a lot of information, learning how to use those features will help you progress rapidly and use MATLAB efficiently.

Section 1.8: Data Types

There are 16 fundamental data types, or classes, in MATLAB. Each of these classes is in the form of a matrix or array. With the exception of function handles, this matrix or array is a minimum of 0-by-0 in size and can grow to an

任意大小的n维数组。函数句柄始终是标量（1×1）。

MATLAB中的一个重要点是，默认情况下你不需要使用任何类型声明或维度语句。当你定义新变量时，MATLAB会自动创建它并分配适当的内存空间。

示例：

```
a = 123;
b = [1 2 3];
c = '123';

>> whos
名称      大小            字节数  类型     属性

  a         1x1                 8  double
  b         1x3                24  double
  c         1x3                 6  char
```

如果变量已存在，MATLAB会用新数据替换原始数据，并在必要时分配新的存储空间。

基本数据类型

基本数据类型有：数值型、逻辑型、字符型、单元格、结构体、表格和函数句柄。

数值数据类型：

- 浮点数（默认）

  MATLAB以双精度或单精度格式表示浮点数。默认是双精度，但你可以通过简单的转换函数将任何数字转换为单精度：

```
a = 1.23;
b = single(a);

>> whos
名称      大小            字节数  类别     属性

  a         1x1                 8  double
  b         1x1                 4  single
```

- 整数

  MATLAB有四种有符号和四种无符号整数类。有符号类型允许你处理负整数和正整数，但由于一个位用于表示数字的正负号，其数值范围不如无符号类型宽。无符号类型提供更宽的数值范围，但这些数字只能是零或正数。

  MATLAB支持1、2、4和8字节的整数存储。如果你使用能容纳数据的最小整数类型，可以节省程序的内存和执行时间。例如，存储数值100不需要32位整数。

```
a = int32(100);
b = int8(100);

>> whos
```


n-dimensional array of any size. A function handle is always scalar (1-by-1).

Important moment in MATLAB is that you don't need to use any type declaration or dimension statements by default. When you define new variable MATLAB creates it automatically and allocates appropriate memory space.

Example:

```
a = 123;
b = [1 2 3];
c = '123';

>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1                 8  double
  b         1x3                24  double
  c         1x3                 6  char
```

If the variable already exists, MATLAB replaces the original data with new one and allocates new storage space if necessary.

Fundamental data types

Fundamental data types are: numeric, `logical`, `char`, `cell`, `struct`, `table` and `function_handle`.

Numeric data types:

- Floating-Point numbers (*default*)

  MATLAB represents floating-point numbers in either double-precision or single-precision format. The default is double precision, but you can make any number single precision with a simple conversion function:

```
a = 1.23;
b = single(a);

>> whos
  Name      Size            Bytes  Class     Attributes

  a         1x1                 8  double
  b         1x1                 4  single
```

- Integers

  MATLAB has four signed and four unsigned integer classes. Signed types enable you to work with negative integers as well as positive, but cannot represent as wide a range of numbers as the unsigned types because one bit is used to designate a positive or negative sign for the number. Unsigned types give you a wider range of numbers, but these numbers can only be zero or positive.

  MATLAB supports 1-, 2-, 4-, and 8-byte storage for integer data. You can save memory and execution time for your programs if you use the smallest integer type that accommodates your data. For example, you do not need a 32-bit integer to store the value 100.

```
a = int32(100);
b = int8(100);

>> whos
```

```
名称      大小            字节数  类型    属性

  a         1x1                4  int32
  b         1x1                1  int8
```

要将数据存储为整数，需要将双精度数转换为所需的整数类型。如果被转换为整数的数字有小数部分，MATLAB 会四舍五入到最接近的整数。如果小数部分恰好是0.5，则在两个同样接近的整数中，MATLAB 会选择绝对值较大的那个。

```
a  = int16(456);
```

- char

字符数组为 MATLAB 中的文本数据提供存储。按照传统编程术语，字符数组（序列）定义为字符串。MATLAB 的零售版本中没有显式的字符串类型。

- logical：逻辑值为1或0，分别表示真和假。用于关系条件和数组索引。因为它只有真或假，所以大小为1字节。

```
a = logical(1);
```

- 结构体。结构体数组是一种数据类型，使用称为字段的数据容器将不同数据类型的变量组合在一起。每个字段可以包含任何类型的数据。使用点符号访问结构体中的数据，格式为structName.fieldName。

```
field1 = 'first';
field2 = 'second';
value1 = [1 2 3 4 5];
value2 = 'sometext';
s = struct(field1,value1,field2,value2);
```

为了访问 value1，以下每种语法都是等价的

```
s.first 或 s.(field1) 或 s.('first')
```

我们可以用第一种方法显式访问我们知道存在的字段，或者在第二个例子中传入字符串或创建字符串来访问字段。第三个例子演示了点符号加括号的写法接受一个字符串，这个字符串与存储在 field1 变量中的字符串相同。

- 表格变量可以具有不同的大小和数据类型，但所有变量必须具有相同的行数。

```
Age = [15 25 54]';
Height = [176 190 165]';
Name = {'Mike', 'Pete', 'Steeve'}';
T = table(Name,Age, Height);
```

- cell。它是非常有用的 MATLAB 数据类型：cell 数组是一个数组，其每个元素可以是不同的数据类型和大小。它是一个非常强大的工具，可以按需操作数据。

```
a = { [1 2 3], 56, 'art'};
```



```
  Name      Size            Bytes  Class    Attributes

  a         1x1                 4  int32
  b         1x1                 1  int8
```

To store data as an integer, you need to convert from double to the desired integer type. If the number being converted to an integer has a fractional part, MATLAB rounds to the nearest integer. If the fractional part is exactly `0.5`, then from the two equally nearby integers, MATLAB chooses the one for which the absolute value is larger in magnitude.

```
a  = int16(456);
```

- char

Character arrays provide storage for text data in MATLAB. In keeping with traditional programming terminology, an array (sequence) of characters is defined as a string. There is no explicit string type in retail releases of MATLAB.

- logical: logical values of 1 or 0, represent true and false respectively. Use for relational conditions and array indexing. Because it's just TRUE or FALSE it has size of 1 byte.

```
a = logical(1);
```

- structure. A structure array is a data type that groups variables of different data types using data containers called *fields*. Each field can contain any type of data. Access data in a structure using dot notation of the form structName.fieldName.

```
field1 = 'first';
field2 = 'second';
value1 = [1 2 3 4 5];
value2 = 'sometext';
s = struct(field1,value1,field2,value2);
```

In order to access value1, each of the following syntax are equivalent

```
s.first or s.(field1) or s.('first')
```

We can explicitly access a field we know will exist with the first method, or either pass a string or create a string to access the field in the second example. The third example is demonstrating that the dot parentheses notation takes a string, which is the same one stored in the field1 variable.

- table variables can be of different sizes and data types, but all variables must have the same number of rows.

```
Age = [15 25 54]';
Height = [176 190 165]';
Name = {'Mike', 'Pete', 'Steeve'}';
T = table(Name,Age, Height);
```

- cell. It's very useful MATLAB data type: cell array is an array each element of it can be of different data type and size. It's very strong instrument for manipulating data as you wish.

```
a = { [1 2 3], 56, 'art'};
```

或者

```
a = cell(3);
```

- 函数句柄存储指向函数的指针（例如，指向匿名函数）。它允许你将一个函数传递给另一个函数，或从主函数外部调用局部函数。

有许多工具可用于处理每种数据类型，还有内置的数据类型转换函数（`str2double`，`table2cell`）。

附加数据类型

还有几种附加的数据类型，在某些特定情况下非常有用。它们是：

- 日期和时间：用于表示日期、时间和持续时间的数组。datetime（'now'）返回2016年7月21日 16:30:16。
- 分类数组：用于存储来自一组离散类别值的数据的数据类型。适合存储非数值数据（节省内存）。可以用于表格中选择行组。

```
a = categorical({'a' 'b' 'c'});
```

- 映射容器是一种数据结构，具有独特的能力，不仅可以通过任何标量数值，还可以通过字符向量进行索引。映射元素的索引称为键。这些键及其关联的数据值存储在映射中。
- 时间序列是按时间顺序采样的数据向量，通常以规则间隔采样。它适合存储与时间步相关的数据，并且有许多有用的方法可供使用。

第1.9节：读取输入与写入输出

就像所有编程语言一样，MATLAB 设计用于读取和写入多种格式。其原生库支持大量的文本、图像、视频、音频、数据格式，并且每个版本更新都会包含更多格式——点击这里查看支持的文件格式完整列表以及导入它们所用的函数。

在尝试加载文件之前，你必须先问自己想让数据变成什么样子，以及你期望计算机如何为你组织数据。假设你有一个格式如下的 txt/csv 文件：

```
水果，总单位，销售后剩余单位，单价
苹果，200，67，$0.14
香蕉，300，172，$0.11
菠萝，50，12，$1.74
```

我们可以看到第一列是字符串格式，第二列和第三列是数值型，最后一列是货币形式。假设我们想用 MATLAB 计算今天的收入，首先要加载这个 txt/csv 文件。查看链接后，我们发现字符串和数值类型的 txt 文件由textscan处理。因此我们可以尝试：

```
fileID = fopen('dir/test.txt'); %从目录加载文件
C = textscan(fileID,'%s %f %f %s','Delimiter',',','HeaderLines',1); %解析 txt/csv 文件
```

其中%s表示元素是字符串类型，%f表示元素是浮点数类型，文件以“,”分隔。HeaderLines 选项告诉 MATLAB 跳过前 N 行，而紧跟其后的 1

or

```
a = cell(3);
```

- function handles stores a pointer to a function (for example, to anonymous function). It allows you to pass a function to another function, or call local functions from outside the main function.

There are a lot of instruments to work with each data type and also built-in data type conversion functions (`str2double`, `table2cell`).

Additional data types

There are several additional data types which are useful in some specific cases. They are:

- Date and time: arrays to represent dates, time, and duration. `datetime('now')` returns `21-Jul-2016 16:30:16`.
- Categorical arrays: it's data type for storing data with values from a set of discrete categories. Useful for storing nonnumeric data (memory effective). Can be used in a table to select groups of rows.

```
a = categorical({'a' 'b' 'c'});
```

- Map containers is a data structure that has unique ability to indexing not only through the any scalar numeric values but character vector. Indices into the elements of a Map are called keys. These keys, along with the data values associated with them, are stored within the Map.
- Time series are data vectors sampled over time, in order, often at regular intervals. It's useful to store the data connected with timesteps and it has a lot of useful methods to work with.

Section 1.9: Reading Input & Writing Output

Just like all programming language, MATLAB is designed to read and write in a large variety of formats. The native library supports a large number of Text,Image,Video,Audio,Data formats with more formats included in each version update - check here to see the full list of supported file formats and what function to use to import them.

Before you attempt to load in your file, you must ask yourself what do you want the data to become and how you expect the computer to organize the data for you. Say you have a txt/csv file in the following format:

```
Fruit,TotalUnits,UnitsLeftAfterSale,SellingPricePerUnit
Apples,200,67,$0.14
Bananas,300,172,$0.11
Pineapple,50,12,$1.74
```

We can see that the first column is in the format of Strings, while the second, third are Numeric, the last column is in the form of Currency. Let's say we want to find how much revenue we made today using MATLAB and first we want to load in this txt/csv file. After checking the link, we can see that String and Numeric type of txt files are handled by `textscan`. So we could try:

```
fileID = fopen('dir/test.txt'); %Load file from dir
C = textscan(fileID,'%s %f %f %s','Delimiter',',','HeaderLines',1); %Parse in the txt/csv
```

where `%s` suggest that the element is a String type, `%f` suggest that the element is a Float type, and that the file is Delimited by ",". The HeaderLines option asks MATLAB to skip the First N lines while the 1 immediately after it

表示跳过第一行（表头行）。

现在 C 是我们加载的数据，形式是一个包含 4 个单元的单元数组，每个单元包含 txt/csv 文件中对应列的数据。

首先，我们想通过用第二列减去第三列来计算今天卖出了多少水果，具体操作如下：

```
sold = C{2} - C{3}; %C{2} 给出第二个单元格（或第二列）内的元素
```

现在我们想将这个向量乘以单价，所以首先需要将那一列字符串转换成数字列，然后使用 MATLAB 的 cell2mat 将其转换为数值矩阵，第一步是去掉“$”符号，有很多方法可以做到。最直接的方法是使用简单的正则表达式：

```
D = cellfun(@(x)(str2num(regexprep(x, '\$',''))), C{4}, 'UniformOutput', false);%cellfun 让我们避免对单元格中的每个元素进行循环。
```

或者你可以使用循环：

```
for t=1:size(C{4},1)
D{t} = str2num(regexprep(C{4}{t}, '\$',''));
end

E = cell2mat(D)% 将单元格数组转换为矩阵
```

str2num 函数将去掉“$”符号的字符串转换为数值类型，cell2mat 将包含数值元素的单元格转换为数字矩阵

现在我们可以将售出单位数乘以单价：

```
revenue = sold .* E; % 在 MATLAB 中，元素逐个相乘用 .* 表示

totalrevenue = sum(revenue);
```


means to skip the first line (the header line).

Now C is the data we have loaded which is in the form of a Cell Array of 4 cells, each containing the column of data in the txt/csv file.

So first we want to calculate how many fruits we sold today by subtracting the third column from the second column, this can be done by:

```
sold = C{2} - C{3}; %C{2} gives the elements inside the second cell (or the second column)
```

Now we want to multiply this vector by the Price per unit, so first we need to convert that column of Strings into a column of Numbers, then convert it into a Numeric Matrix using MATLAB's `cell2mat` the first thing we need to do is to strip-off the "$" sign, there are many ways to do this. The most direct way is using a simple regex:

```
D = cellfun(@(x)(str2num(regexprep(x, '\$',''))), C{4}, 'UniformOutput', false);%cellfun allows us to avoid looping through each element in the cell.
```

Or you can use a loop:

```
for t=1:size(C{4},1)
   D{t} = str2num(regexprep(C{4}{t}, '\$',''));
end

E = cell2mat(D)% converts the cell array into a Matrix
```

The `str2num` function turns the string which had "$" signs stripped into numeric types and `cell2mat` turns the cell of numeric elements into a matrix of numbers

Now we can multiply the units sold by the cost per unit:

```
revenue = sold .* E; %element-wise product is denoted by .* in MATLAB

totalrevenue = sum(revenue);
```

第2章：初始化矩阵或数组

参数	详细信息
大小	n（用于 n x n 矩阵）
大小	n，m（用于 n x m 矩阵）
大小	m，n，...，k（用于 m×n×...×k 矩阵）
数据类型	‘double’（默认），‘single’，‘int8’，‘uint8’，‘int16’，‘uint16’，‘int32’，‘uint32’，‘int64’或‘uint64’
数组类型	‘distributed’
数组类型	'codistributed'
arraytype	'gpuArray'

MATLAB 有三个重要函数用于创建矩阵并将其元素设置为零、全一或单位矩阵。（单位矩阵在主对角线上为1，其余位置为0。）

第2.1节：创建全零矩阵

```
z1 = zeros(5); % 创建一个5×5的全零矩阵
z2 = zeros(2,3); % 创建一个2×3的矩阵
```

第2.2节：创建全一矩阵

```
o1 = ones(5); % 创建一个5×5的全一矩阵
o2 = ones(1,3); % 创建一个1×3的矩阵/大小为3的向量
```

第2.3节：创建单位矩阵

```
i1 = eye(3); % 创建一个3×3的单位矩阵
i2 = eye(5,6); % 创建一个5×6的单位矩阵
```


Chapter 2: Initializing Matrices or arrays

Parameter	Details
sz	n (for an n x n matrix)
sz	n, m (for an n x m matrix)
sz	m,n,...,k (for an m-by-n-by-...-by-k matrix)
datatype	'double' (default), 'single', 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64', or 'uint64'
arraytype	'distributed'
arraytype	'codistributed'
arraytype	'gpuArray'

MATLAB has three important functions to create matrices and set their elements to zeroes, ones, or the identity matrix. (The identity matrix has ones on the main diagonal and zeroes elsewhere.)

Section 2.1: Creating a matrix of 0s

```
z1 = zeros(5); % Create a 5-by-5 matrix of zeroes
z2 = zeros(2,3); % Create a 2-by-3 matrix
```

Section 2.2: Creating a matrix of 1s

```
o1 = ones(5); % Create a 5-by-5 matrix of ones
o2 = ones(1,3); % Create a 1-by-3 matrix / vector of size 3
```

Section 2.3: Creating an identity matrix

```
i1 = eye(3); % Create a 3-by-3 identity matrix
i2 = eye(5,6); % Create a 5-by-6 identity matrix
```

第3章：条件

参数	描述
表达式具有逻辑意义的表达式	

第3.1节：IF条件

条件是几乎所有代码中不可或缺的一部分。它们用于仅在某些情况下执行代码的某些部分，而在其他情况下不执行。让我们来看基本语法：

```
a = 5;
if a > 10    % 该条件不满足，因此不会发生任何事情
    disp('OK')
end

if a < 10    % 该条件满足，因此执行if...end之间的语句
    disp('Not OK')
end
```

输出：

```
Not OK
```

在这个例子中，我们看到if由两部分组成：条件，以及条件为真时要执行的代码。代码是指条件之后、该if的end之前的所有内容。第一个条件不满足，因此其中的代码未被执行。

这是另一个例子：

```
a = 5;
if a ~= a+1        % “~=“ 表示“不等于”
    disp('是真的！') % 我们使用两个单引号告诉 MATLAB 这里的 ' 是字符串的一部分
end
```

上述条件将始终为真，并且会显示输出It's true!。

我们也可以这样写：

```
a = 5;
if a == a+1    % "==" 表示“等于”，它不是赋值运算符("=")
    disp('相等')
end
```

这次条件始终为假，因此我们永远不会得到输出相等。

不过，对于始终为真或始终为假的条件用处不大，因为如果它们始终为假，我们可以直接删除这部分代码；如果它们始终为真，那么条件就不需要了。

第3.2节：IF-ELSE 条件

在某些情况下，如果条件为假，我们希望运行另一段代码，为此我们使用可选的else部分：

```
a = 20;
if a < 10
    disp('小于10')
```


Chapter 3: Conditions

Parameter	Description
expression	an expression that has logical meaning

Section 3.1: IF condition

Conditions are a fundamental part of almost any part of code. They are used to execute some parts of the code only in some situations, but not other. Let's look at the basic syntax:

```
a = 5;
if a > 10    % this condition is not fulfilled, so nothing will happen
    disp('OK')
end

if a < 10    % this condition is fulfilled, so the statements between the if...end are executed
    disp('Not OK')
end
```

Output:

```
Not OK
```

In this example we see the `if` consists of 2 parts: the condition, and the code to run if the condition is true. The code is everything written after the condition and before the `end` of that `if`. The first condition was not fulfilled and hence the code within it was not executed.

Here is another example:

```
a = 5;
if a ~= a+1        % "~=" means "not equal to"
    disp('It''s true!') % we use two apostrophes to tell MATLAB that the ' is part of the string
end
```

The condition above will always be true, and will display the output `It's true!`.

We can also write:

```
a = 5;
if a == a+1    % "==" means "is equal to", it is NOT the assignment ("=") operator
    disp('Equal')
end
```

This time the condition is always false, so we will never get the output `Equal`.

There is not much use for conditions that are always true or false, though, because if they are always false we can simply delete this part of the code, and if they are always true then the condition is not needed.

Section 3.2: IF-ELSE condition

In some cases we want to run an alternative code if the condition is false, for this we use the optional `else` part:

```
a = 20;
if a < 10
    disp('a smaller than 10')
```

```
否则
    disp('大于10')
结束
```

这里我们看到，因为 a 不小于 10，代码的第二部分，即 else 后的部分被执行，输出为 大于 10。现在我们来看另一个例子：

```
a = 10;
如果 a > 10
    disp('大于10')
否则
    disp('小于10')
结束
```

这个例子表明我们没有检查 a 是否确实小于10，导致输出了错误的信息，因为条件只检查表达式本身，任何不等于真的情况（如 a = 10）都会执行第二部分。

这种类型的错误是初学者和有经验的程序员都常犯的陷阱，尤其是在条件变得复杂时，应始终牢记这一点。

第3.3节：IF-ELSEIF条件

使用else我们可以在条件不满足时执行某些任务。但如果我们想在第一个条件为假时检查第二个条件怎么办呢？我们可以这样做：

```
a = 9;
if mod(a,2)==0    % MOD - 取模运算，返回'a'除以2的余数
    disp('a是偶数')
否则
    if mod(a,3)==0
        disp('3是a的因数')
    end
end
```

```
输出：
3是a的因数
```

这也称为“嵌套条件”，但这里有一种特殊情况可以提高代码的可读性，并减少出错的可能——我们可以这样写：

```
a = 9;
if mod(a,2)==0
    disp('a是偶数')
elseif mod(a,3)==0
    disp('3是a的因数')
end
```

```
输出：
3是a的因数
```

使用elseif我们能够在同一个条件块中检查另一个表达式，这不限于一次尝试：

```
a = 25;
```



```
else
    disp('a bigger than 10')
end
```

Here we see that because a is not smaller than 10 the second part of the code, after the else is executed and we get the output a bigger than 10. Now let's look at another try:

```
a = 10;
if a > 10
    disp('a bigger than 10')
else
    disp('a smaller than 10')
end
```

In this example shows that we did not checked if a is indeed smaller than 10, and we get a wrong message because the condition only check the expression as it is, and ANY case that does not equals true (a = 10) will cause the second part to be executed.

This type of error is a very common pitfall for both beginners and experienced programmers, especially when conditions become complex, and should be always kept in mind

Section 3.3: IF-ELSEIF condition

Using else we can perform some task when the condition is not satisfied. But what if we want to check a second condition in case that the first one was false. We can do it this way:

```
a = 9;
if mod(a,2)==0   % MOD - modulo operation, return the remainder after division of 'a' by 2
    disp('a is even')
else
    if mod(a,3)==0
        disp('3 is a divisor of a')
    end
end
```

```
OUTPUT:
3 is a divisor of a
```

This is also called "nested condition", but here we have a special case that can improve code readability, and reduce the chance for an error - we can write:

```
a = 9;
if mod(a,2)==0
    disp('a is even')
elseif mod(a,3)==0
    disp('3 is a divisor of a')
end
```

```
OUTPUT:
3 is a divisor of a
```

using the elseif we are able to check another expression within the same block of condition, and this is not limited to one try:

```
a = 25;
```

```
if mod(a,2)==0
    disp('a 是偶数')
elseif mod(a,3)==0
    disp('3 是 a 的因数')
elseif mod(a,5)==0
    disp('5 是 a 的因数')
end
```

输出：
5 是 a 的因数

在连续使用elseif时应格外小心，因为在整个if到end块中，只有一个会被执行。因此，在我们的例子中，如果想显示所有a的因数（在我们明确检查的范围内），上面的例子就不合适：

```
a = 15;
if mod(a,2)==0
    disp('a 是偶数')
elseif mod(a,3)==0
    disp('3 是 a 的因数')
elseif mod(a,5)==0
    disp('5 是 a 的因数')
end
```

输出：
3是a的因数

不仅3，5也是15的因数，但如果上面任何表达式为真，检查5是否为因数的部分就不会被执行。

最后，我们可以在所有elseif条件之后添加一个else（且只能有一个），当上述条件都不满足时执行代码：

```
a = 11;
if mod(a,2)==0
    disp('a 是偶数')
elseif mod(a,3)==0
    disp('3 是 a 的因数')
elseif mod(a,5)==0
    disp('5 是 a 的因数')
else
    disp('2、3 和 5 都不是 a 的因数')
end
```

输出：
2、3 和 5 都不是 a 的因数

第3.4节：嵌套条件

当我们在一个条件中使用另一个条件时，我们称这些条件是“嵌套”的。嵌套条件的一个特殊情况是elseif选项，但还有许多其他使用嵌套条件的方法。让我们来看以下代码：

```
a = 2;
if mod(a,2)==0    % MOD - 取模运算，返回 a 除以 2 的余数
```



```
if mod(a,2)==0
    disp('a is even')
elseif mod(a,3)==0
    disp('3 is a divisor of a')
elseif mod(a,5)==0
    disp('5 is a divisor of a')
end
```

OUTPUT:
5 is a divisor of a

Extra care should be taken when choosing to use `elseif` in a row, since **only one** of them will be executed from all the `if` to `end` block. So, in our example if we want to display all the divisors of a (from those we explicitly check) the example above won't be good:

```
a = 15;
if mod(a,2)==0
    disp('a is even')
elseif mod(a,3)==0
    disp('3 is a divisor of a')
elseif mod(a,5)==0
    disp('5 is a divisor of a')
end
```

OUTPUT:
3 is a divisor of a

not only 3, but also 5 is a divisor of 15, but the part that check the division by 5 is not reached if any of the expressions above it was true.

Finally, we can add one `else` (and only **one**) after all the `elseif` conditions to execute a code when none of the conditions above are met:

```
a = 11;
if mod(a,2)==0
    disp('a is even')
elseif mod(a,3)==0
    disp('3 is a divisor of a')
elseif mod(a,5)==0
    disp('5 is a divisor of a')
else
    disp('2, 3 and 5 are not divisors of a')
end
```

OUTPUT:
2, 3 and 5 are not divisors of a

Section 3.4: Nested conditions

When we use a condition within another condition we say the conditions are "nested". One special case of nested conditions is given by the `elseif` option, but there are numerous other ways to use nested conditions. Let's examine the following code:

```
a = 2;
if mod(a,2)==0    % MOD - modulo operation, return the remainder after division of 'a' by 2
```

```
    disp('a 是偶数')
    if mod(a,3)==0
        disp('3 是 a 的因数')
        if mod(a,5)==0
            disp('5 是 a 的因数')
        end
    end
否则
    disp('a 是奇数')
end
```

对于 a=2，输出将是 a 是 偶数，这是正确的。对于 a=3，输出将是 a 是 奇数，这也正确，但遗漏了检查 3 是否是 a 的因数。这是因为条件是嵌套的，所以只有当第一个条件为 真 时，才会进入内部条件，如果 a 是奇数，内部条件根本不会被检查。这与使用 elseif 有些相反，elseif 只有在第一个条件为 假 时才会检查下一个条件。那检查是否能被 5 整除呢？只有能被 6 整除（即同时被 2 和 3 整除）的数字才会被检查是否能被 5 整除，我们可以测试并看到对于 a=30，输出是：

```
a 是偶数
3 是 a 的因数
5 是 a 的因数
```

我们还应该注意两点：

1. end 位置正确对应每个 if 对于条件集按预期工作至关重要，
   所以缩进不仅仅是一个好的建议。
2. else语句的位置也很关键，因为我们需要知道它属于哪个if（并且可能有多个）
   在表达式为false的情况下，我们想做一些事情（其中有几个）。

让我们看另一个例子：

```
对于 a = 5:10    % FOR循环对从5到10的每个a执行其内的所有代码
    ch = num2str(a);    % NUM2STR将整数a转换为字符
    if mod(a,2)==0
        if mod(a,3)==0
            disp(['3是' ch'的因数'])
        elseif  mod(a,4)==0
            disp(['4是' ch'的因数'])
        else
            disp([ch '是偶数'])
        end
    elseif mod(a,3)==0
        disp(['3是' ch'的因数'])

    else
        disp([ch '是奇数'])
    end
end
```

输出将是：

```
5 是奇数
3 是 6 的因数
7 是奇数
4 是 8 的因数
3 是 9 的因数
10 是偶数
```



```
    disp('a is even')
    if mod(a,3)==0
        disp('3 is a divisor of a')
        if mod(a,5)==0
            disp('5 is a divisor of a')
        end
    end
else
    disp('a is odd')
end
```

For `a=2`, the output will be `a is even`, which is correct. For `a=3`, the output will be `a is odd`, which is also correct, but misses the check if 3 is a divisor of `a`. This is because the conditions are nested, so **only** if the first is `true`, than we move to the inner one, and if `a` is odd, none of the inner conditions are even checked. This is somewhat opposite to the use of `elseif` where only if the first condition is `false` than we check the next one. What about checking the division by 5? only a number that has 6 as a divisor (both 2 and 3) will be checked for the division by 5, and we can test and see that for `a=30` the output is:

```
a is even
3 is a divisor of a
5 is a divisor of a
```

We should also notice two things:

1. The position of the `end` in the right place for each `if` is crucial for the set of conditions to work as expected, so indentation is more than a good recommendation here.
2. The position of the `else` statement is also crucial, because we need to know in which `if` (and there could be several of them) we want to do something in case the expression if `false`.

Let's look at another example:

```
for a = 5:10    % the FOR loop execute all the code within it for every a from 5 to 10
    ch = num2str(a);    % NUM2STR converts the integer a to a character
    if mod(a,2)==0
        if mod(a,3)==0
            disp(['3 is a divisor of ' ch])
        elseif  mod(a,4)==0
            disp(['4 is a divisor of ' ch])
        else
            disp([ch ' is even'])
        end
    elseif mod(a,3)==0
        disp(['3 is a divisor of ' ch])

    else
        disp([ch ' is odd'])
    end
end
```

And the output will be:

```
5 is odd
3 is a divisor of 6
7 is odd
4 is a divisor of 8
3 is a divisor of 9
10 is even
```

我们看到，对于 6 个数字，我们只得到了 6 行，因为条件是嵌套的，确保每个数字只打印一次，且（虽然从输出中看不到）没有执行额外的检查，所以如果一个数字不是偶数，就没有必要检查 4 是否是它的因数。

we see that we got only 6 lines for 6 numbers, because the conditions are nested in a way that ensure only one print per number, and also (although can't be seen directly from the output) no extra checks are preformed, so if a number is not even there is no point to check if 4 is one of it divisors.

第4章：函数

第4.1节：nargin，nargout

在函数体内，nargin 和 nargout 分别表示调用时实际提供的输入和输出数量。

例如，我们可以根据提供的输入数量来控制函数的执行。

myVector.m：

```
function [res] = myVector(a, b, c)
    % 大致模拟冒号运算符

    switch nargin
        case 1
res = [0:a];
        case 2
res = [a:b];
        case 3
res = [a:b:c];
        otherwise
            error('参数数量错误');
    end
end
```

终端：

```
>> myVector(10)

ans =

    0    1    2    3    4    5    6    7    8    9   10

>> myVector(10, 20)

ans =

   10   11   12   13   14   15   16   17   18   19   20

>> myVector(10, 2, 20)

ans =

   10   12   14   16   18   20
```

我们也可以根据输出参数的数量来控制函数的执行。

myIntegerDivision:

```
function [qt, rm] = myIntegerDivision(a, b)
    qt = floor(a / b);

    if nargout == 2
        rm = rem(a, b);
    end
end
```


Chapter 4: Functions

Section 4.1: nargin, nargout

In the body of a function `nargin` and `nargout` indicate respectively the actual number of input and output supplied in the call.

We can for example control the execution of a function based on the number of provided input.

myVector.m:

```
function [res] = myVector(a, b, c)
    % Roughly emulates the colon operator

    switch nargin
        case 1
            res = [0:a];
        case 2
            res = [a:b];
        case 3
            res = [a:b:c];
        otherwise
            error('Wrong number of params');
    end
end
```

terminal:

```
>> myVector(10)

ans =

    0    1    2    3    4    5    6    7    8    9   10

>> myVector(10, 20)

ans =

   10   11   12   13   14   15   16   17   18   19   20

>> myVector(10, 2, 20)

ans =

   10   12   14   16   18   20
```

In a similar way we can control the execution of a function based on the number of output parameters.

myIntegerDivision:

```
function [qt, rm] = myIntegerDivision(a, b)
    qt = floor(a / b);

    if nargout == 2
        rm = rem(a, b);
    end
end
```

terminal:

```
>> q = myIntegerDivision(10, 7)

q = 1

>> [q, r] = myIntegerDivision(10, 7)

q = 1
r = 3
```


terminal:

```
>> q = myIntegerDivision(10, 7)

q = 1

>> [q, r] = myIntegerDivision(10, 7)

q = 1
r = 3
```

第5章：集合运算

参数	详细信息
A,B	集合，可能是矩阵或向量
x	集合的可能元素

第5.1节：基本集合运算

可以使用MATLAB执行基本的集合运算。假设我们有两个向量或数组

```
A = randi([0 10],1,5);
B = randi([-1 9], 1,5);
```

我们想找到所有同时属于A和B的元素。为此我们可以使用

```
C = intersect(A,B);
```

C 将包含所有既属于A又属于B的数字。如果我们还想找到这些元素的位置，我们称之为

```
[C,pos] = intersect(A,B);
```

pos 是这些元素的位置，使得 C == A(pos)成立。

另一个基本操作是两个集合的并集

```
D = union(A,B);
```

这里的 D 包含了 A 和 B 的所有元素。

注意，这里 A 和 B 被视为集合，这意味着元素在 A 或 B 中出现的次数无关紧要。为澄清这一点，可以检查 D == union(D,C)。

如果我们想获得存在于 'A' 但不存在于 'B' 的数据，可以使用以下函数

```
E = setdiff(A,B);
```

我们再次强调，这些是集合，因此以下等式成立 D == union(E,B)。

假设我们想要检查是否

```
x = randi([-10 10],1,1);
```

如果元素属于A或B，我们可以执行命令

```
a = ismember(A,x);
b = ismember(B,x);
```

如果 a==1，则 x 是 A 的元素，且当 a==0 时 x 不是元素。对于 B 也是同理。如果 a==1 && b==1，则 x 也是 C 的元素。如果 a == 1 || b == 1，则 x 是 D 的元素；如果 a == 1 || b == 0，则 x 也是 E 的元素。

Chapter 5: Set operations

Parameter	Details
A,B	sets, possibly matrices or vectors
x	possible element of a set

Section 5.1: Elementary set operations

It's possible to perform elementary set operations with MATLAB. Let's assume we have given two vectors or arrays

```
A = randi([0 10],1,5);
B = randi([-1 9], 1,5);
```

and we want to find all elements which are in A and in B. For this we can use

```
C = intersect(A,B);
```

C will include all numbers which are part of A and part of B. If we also want to find the position of these elements we call

```
[C,pos] = intersect(A,B);
```

pos is the position of these elements such that C == A(pos).

Another basic operation is the union of two sets

```
D = union(A,B);
```

Herby contains D all elements of A and B.

Note that A and B are hereby treated as sets which means that it does not matter how often an element is part of A or B. To clarify this one can check D == union(D,C).

If we want to obtain the data that is in 'A' but not in 'B' we can use the following function

```
E = setdiff(A,B);
```

We want to note again that this are sets such that following statement holds D == union(E,B).

Suppose we want to check if

```
x = randi([-10 10],1,1);
```

is an element of either A or B we can execute the command

```
a = ismember(A,x);
b = ismember(B,x);
```

If a==1 then x is element of A and x is no element is a==0. The same goes for B. If a==1 && b==1 x is also an element of C. If a == 1 || b == 1 x is element of D and if a == 1 || b == 0 it's also element of E.

第6章：函数文档编写

第6.1节：获取函数签名

通常让MATLAB打印函数的第一行是很有帮助的，因为这通常包含函数签名，
包括输入和输出：

```
dbtype <functionName> 1
```

示例：

```
>> dbtype fit 1

1  function [fitobj,goodness,output,warnstr,errstr,convmsg] =
fit(xdatain,ydatain,fittypeobj,varargin)
```

第6.2节：简单函数文档

```
function output = mymult(a, b)
% MYMULT 乘以两个数字。
%   output = MYMULT(a, b) 计算 a 和 b 的乘积。
%
%   另见 fft、foo、sin。
%
%   更多信息，请参见 <a href="matlab:web('https://google.com')">Google</a>。
output = a * b;
end
```

help mymult 然后显示：

> mymult 乘以两个数字。
>
> output = mymult(a, b) 计算 a 和 b 的乘积。
>
> 另见 fft、foo、sin。
>
> 有关更多信息，请参见Google。

fft和sin会自动链接到各自的帮助文本，而Google是指向google.com的链接。foo在这种情况下不会链接到任何文档，只要搜索路径上没有名为foo的已记录函数/类。

第6.3节：本地函数文档

在此示例中，本地函数baz（定义在foo.m中）的文档可以通过help foo中的生成链接访问，或直接通过help foo>baz访问。

```
function bar = foo
%这是FOO的文档。
% 另见 foo>baz

% 这不会被打印，因为有一行没有%符号。
end
```


Chapter 6: Documenting functions

Section 6.1: Obtaining a function signature

It is often helpful to have MATLAB print the 1st line of a function, as this usually contains the function signature, including inputs and outputs:

```
dbtype <functionName> 1
```

Example:

```
>> dbtype fit 1

1  function [fitobj,goodness,output,warnstr,errstr,convmsg] =
fit(xdatain,ydatain,fittypeobj,varargin)
```

Section 6.2: Simple Function Documentation

```
function output = mymult(a, b)
% MYMULT Multiply two numbers.
%   output = MYMULT(a, b) multiplies a and b.
%
%   See also fft, foo, sin.
%
%   For more information, see <a href="matlab:web('https://google.com')">Google</a>.
    output = a * b;
end
```

`help mymult` then provides:

> **mymult** Multiply two numbers.
>
> output = **mymult**(a, b) multiplies a and b.
>
> See also fft, foo, sin.
>
> For more information, see Google.

`fft` and `sin` automatically link to their respective help text, and Google is a link to google.com. `foo` will not link to any documentation in this case, as long as there is not a documented function/class by the name of `foo` on the search path.

Section 6.3: Local Function Documentation

In this example, documentation for the local function `baz` (defined in `foo.m`) can be accessed either by the resulting link in `help foo`, or directly through `help foo>baz`.

```
function bar = foo
%This is documentation for FOO.
% See also foo>baz

% This won't be printed, because there is a line without % on it.
end
```

```
function baz
% 这是BAZ的文档。
end
```

第6.4节：用示例脚本记录函数

为了记录一个函数，通常有一个使用该函数的示例脚本会很有帮助。MATLAB中的publish函数可以用来生成带有嵌入图片、代码、链接等的帮助文件。记录代码的语法可以在这里找到。

函数 该函数使用MATLAB中的“校正”FFT。

```
function out_sig = myfft(in_sig)

out_sig = fftshift(fft(ifftshift(in_sig)));

end
```

示例脚本 这是一个单独的脚本，解释输入、输出，并给出一个示例说明为什么需要校正。感谢吴侃（Wu，Kan），该函数的原作者。

```
%% myfft
% 该函数使用MATLAB中的“正确”fft。注意fft需要
% 乘以dt才具有物理意义。
% 关于为什么FFT应这样计算的完整说明，请参阅
% 本代码帮助文件夹中包含的Why_use_fftshift(fft(fftshift(x)))__in_Matlab.pdf。
% 还可以找到更多信息：
%
<https://www.mathworks.com/matlabcentral/fileexchange/25473-why-use-fftshift-fft-fftshift-x----in-m
atlab-instead-of-fft-x-->
%
%% 输入
% *in_sig* - 一维信号
%
%% 输出
% *out_sig* - *in_sig* 的校正 FFT
%
%% 示例
% 生成一个具有解析解的信号。然后将解析解与 fft 以及 myfft 进行比较。此示例是

% 由吴侃（Wu, Kan）在上述链接中给出的修改版本。
%%
% 设置参数
fs = 500;            %采样频率
dt = 1/fs;           %时间步长
T=1;                 %总时间窗口
t = -T/2:dt:T/2-dt; %时间网格
df = 1/T;           %频率步长
Fmax = 1/2/dt;       %频率窗口
f=-Fmax:df:Fmax-df; %频率网格，未在我们的示例中使用，可用于 plot(f, X)
%%
% 生成高斯曲线
Bx = 10; A = sqrt(log(2))/(2*pi*Bx);     %高斯曲线的特性
x = exp(-t.^2/2/A^2);                    %创建高斯曲线
%%
% 生成解析解
Xan = A*sqrt(2*pi)*exp(-2*pi^2*f.^2*A^2); %X(f)，x(t)的解析傅里叶变换的实部
```



```
function baz
% This is documentation for BAZ.
end
```

Section 6.4: Documenting a Function with an Example Script

To document a function, it is often helpful to have an example script which uses your function. The publish function in MATLAB can then be used to generate a help file with embedded pictures, code, links, etc. The syntax for documenting your code can be found here.

The Function This function uses a "corrected" FFT in MATLAB.

```
function out_sig = myfft(in_sig)

out_sig = fftshift(fft(ifftshift(in_sig)));

end
```

The Example Script This is a separate script which explains the inputs, outputs, and gives an example explaining why the correction is necessary. Thanks to Wu, Kan, the original author of this function.

```
%% myfft
% This function uses the "proper" fft in MATLAB. Note that the fft needs to
% be multiplied by dt to have physical significance.
% For a full description of why the FFT should be taken like this, refer
% to: Why_use_fftshift(fft(fftshift(x)))__in_Matlab.pdf included in the
% help folder of this code. Additional information can be found:
%
<https://www.mathworks.com/matlabcentral/fileexchange/25473-why-use-fftshift-fft-fftshift-x----in-m
atlab-instead-of-fft-x-->
%
%% Inputs
% *in_sig* - 1D signal
%
%% Outputs
% *out_sig* - corrected FFT of *in_sig*
%
%% Examples
% Generate a signal with an analytical solution. The analytical solution is
% then compared to the fft then to myfft. This example is a modified
% version given by Wu, Kan given in the link above.
%%
% Set parameters
fs = 500;            %sampling frequency
dt = 1/fs;           %time step
T=1;                 %total time window
t = -T/2:dt:T/2-dt; %time grids
df = 1/T;           %freq step
Fmax = 1/2/dt;       %freq window
f=-Fmax:df:Fmax-df; %freq grids, not used in our examples, could be used by plot(f, X)
%%
% Generate Gaussian curve
Bx = 10; A = sqrt(log(2))/(2*pi*Bx);     %Characteristics of Gaussian curve
x = exp(-t.^2/2/A^2);                    %Create Gaussian Curve
%%
% Generate Analytical solution
Xan = A*sqrt(2*pi)*exp(-2*pi^2*f.^2*A^2); %X(f), real part of the analytical Fourier transform of
x(t)
```

```
%%
% 进行FFT和修正后的FFT并进行比较
Xfft = dt *fftshift(fft(x));      %FFT
Xfinal = dt * myfft(x);           %修正后的FFT
hold on
plot(f,Xan);
plot(f,real(Xfft));
plot(f,real(Xfinal),'ro');
title('校正与未校正FFT的比较');
legend('解析解','未校正FFT','校正FFT');
xlabel('频率'); ylabel('幅度');
DT = max(f) - min(f);
xlim([-DT/4,DT/4]);
```

输出 发布选项可以在“发布”标签下找到，如下图“简单函数文档”中所示。

MATLAB 将运行脚本，并保存显示的图像以及命令行生成的文本。输出可以保存为多种不同格式，包括 HTML、Latex 和 PDF。

上述示例脚本的输出可以在下图中看到。


```
%%
% Take FFT and corrected FFT then compare
Xfft = dt *fftshift(fft(x));    %FFT
Xfinal = dt * myfft(x);         %Corrected FFT
hold on
plot(f,Xan);
plot(f,real(Xfft));
plot(f,real(Xfinal),'ro');
title('Comparison of Corrected and Uncorrected FFT');
legend('Analytical Solution','Uncorrected FFT','Corrected FFT');
xlabel('Frequency'); ylabel('Amplitude');
DT = max(f) - min(f);
xlim([-DT/4,DT/4]);
```

The Output The publish option can be found under the "Publish" tab, highlighted in the imageSimple Function Documentation below.

MATLAB will run the script, and save the images which are displayed, as well as the text generated by the command line. The output can be saved to many different types of formats, including HTML, Latex, and PDF.

The output of the example script given above can be seen in the image below.

myfft

This function uses the "proper" fft in matlab. Note that the fft needs to be multiplied by dt to have physical significance. For a full description of why the FFT should be taken like this, refer to: Why_use_fftshift(fft(fftshift(x)))__in_Matlab.pdf included in the help folder of this code. Additional information can be found: https://www.mathworks.com/matlabcentral/fileexchange/25473-why-use-fftshift-fft-fftshift-x----in-matlab-instead-of-fft-x--

Contents

Inputs

in_sig - 1D signal

Outputs

out_sig - corrected FFT of **in_sig**

Examples

Generate a signal with an analytical solution. The analytical solution is then compared to the fft then to myfft. This example is a modified version given by Wu, Kan given in the link aboce.

Set parameters

```
fs = 500;           %sampling frequency
dt = 1/fs;          %time step
T=1;                %total time window
t = -T/2:dt:T/2-dt; %time grids
df = 1/T;           %freq step
Fmax = 1/2/dt;      %freq window
f=-Fmax:df:Fmax-df; %freq grids, not used in our examples, could be used by plot(f, X)
```

Generate Gaussian curve

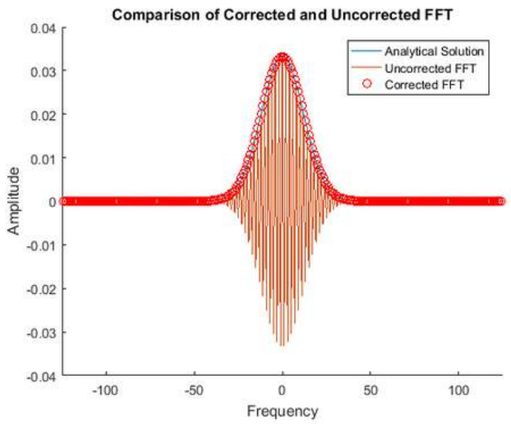
```
Bx = 10; A = sqrt(log(2))/(2*pi*Bx);    %Characteristics of Gaussian curve
x = exp(-t.^2/2/A^2);                   %Create Gaussian Curve
```

Generate Analytical solution

```
Xan = A*sqrt(2*pi)*exp(-2*pi^2*f.^2*A^2); %X(f), real part of the analytical Fourier transform of x(t)
```

Take FFT and corrected FFT then compare

```
Xfft = dt *fftshift(fft(x));    %FFT
Xfinal = dt * myfft(x);         %Corrected FFT
hold on
plot(f,Xan);
plot(f,real(Xfft));
plot(f,real(Xfinal),'ro');
title('Comparison of Corrected and Uncorrected FFT');
legend('Analytical Solution','Uncorrected FFT','Corrected FFT');
xlabel('Frequency'); ylabel('Amplitude');
DT = max(f) - min(f);
xlim([-DT/4,DT/4]);
```



Published with MATLAB® R2016a

myfft

This function uses the "proper" fft in matlab. Note that the fft needs to be multiplied by dt to have physical significance. For a full description of why the FFT should be taken like this, refer to: Why_use_fftshift(fft(fftshift(x)))__in_Matlab.pdf included in the help folder of this code. Additional information can be found: https://www.mathworks.com/matlabcentral/fileexchange/25473-why-use-fftshift-fft-fftshift-x----in-matlab-instead-of-fft-x--

Contents

- Inputs
- Outputs
- Examples

Inputs

in_sig - 1D signal

Outputs

out_sig - corrected FFT of **in_sig**

Examples

Generate a signal with an analytical solution. The analytical solution is then compared to the fft then to myfft. This example is a modified version given by Wu, Kan given in the link aboce.

Set parameters

```
fs = 500;           %sampling frequency
dt = 1/fs;          %time step
T=1;                %total time window
t = -T/2:dt:T/2-dt; %time grids
df = 1/T;           %freq step
Fmax = 1/2/dt;      %freq window
f=-Fmax:df:Fmax-df; %freq grids, not used in our examples, could be used by plot(f, X)
```

Generate Gaussian curve

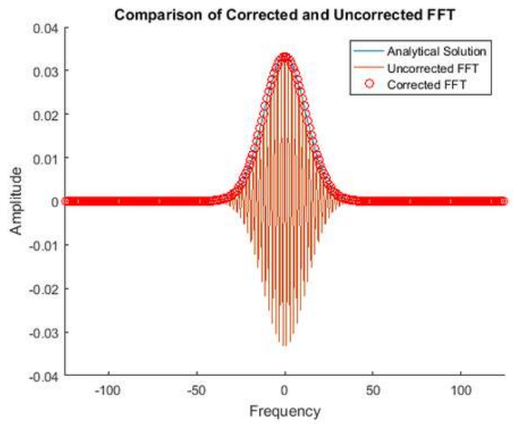
```
Bx = 10; A = sqrt(log(2))/(2*pi*Bx);    %Characteristics of Gaussian curve
x = exp(-t.^2/2/A^2);                   %Create Gaussian Curve
```

Generate Analytical solution

```
Xan = A*sqrt(2*pi)*exp(-2*pi^2*f.^2*A^2); %X(f), real part of the analytical Fourier transform of x(t)
```

Take FFT and corrected FFT then compare

```
Xfft = dt *fftshift(fft(x));    %FFT
Xfinal = dt * myfft(x);         %Corrected FFT
hold on
plot(f,Xan);
plot(f,real(Xfft));
plot(f,real(Xfinal),'ro');
title('Comparison of Corrected and Uncorrected FFT');
legend('Analytical Solution','Uncorrected FFT','Corrected FFT');
xlabel('Frequency'); ylabel('Amplitude');
DT = max(f) - min(f);
xlim([-DT/4,DT/4]);
```



Published with MATLAB® R2016a

第7章：使用带有逻辑输出的函数

第7.1节：空数组的 All 和 Any

当数组可能变为空数组时，使用逻辑运算符需要特别注意。通常预期如果 all(A) 为真，则 any(A) 必须为真；如果 any(A) 为假，则 all(A) 也必须为假。但在 MATLAB 中，空数组的情况并非如此。

```
>> any([])
ans =
     0
>> all([])
ans =
     1
```

例如，如果你正在将数组的所有元素与某个阈值进行比较，你需要注意数组为空的情况：

```
>> A=1:10;
>> all(A>5)
ans =
     0
>> A=1:0;
>> all(A>5)
ans =
     1
```

使用内置函数isempty来检查空数组：

```
a = [];
isempty(a)
ans =
1
```


Chapter 7: Using functions with logical output

Section 7.1: All and Any with empty arrays

Special care needs to be taken when there is a possibility that an array become an empty array when it comes to logical operators. It is often expected that if `all(A)` is true then `any(A)` must be true and if `any(A)` is false, `all(A)` must also be false. That is not the case in MATLAB with empty arrays.

```
>> any([])
ans =
     0
>> all([])
ans =
     1
```

So if for example you are comparing all elements of an array with a certain threshold, you need to be aware of the case where the array is empty:

```
>> A=1:10;
>> all(A>5)
ans =
     0
>> A=1:0;
>> all(A>5)
ans =
     1
```

Use the built-in function `isempty` to check for empty arrays:

```
a = [];
isempty(a)
ans =
1
```

第8章：For循环

第8.1节：遍历矩阵的列

如果赋值的右侧是一个矩阵，那么在每次迭代中，变量将被赋值为该矩阵的后续列。

```
some_matrix = [1, 2, 3; 4, 5, 6]; % 2 行 3 列矩阵
for some_column = some_matrix
display(some_column)
end
```

（行向量版本是这种情况的正常形式，因为在 MATLAB 中，行向量只是列数为 1 的矩阵。）

输出将显示

```
1
4
2
5
3
6
```

即迭代矩阵的每一列被显示，每次调用时打印每一列 display。

第 8.2 节：注意：奇怪的相同计数器嵌套循环

这在其他编程环境中是看不到的。我几年前遇到过这种情况，当时不明白为什么会发生，但在使用 MATLAB 一段时间后，我终于弄明白了。
请看下面的代码片段：

```
for x = 1:10
    for x = 1:10
        fprintf('%d,', x);
    end
    fprintf('');end
```

你可能不会期望这能正常工作，但它确实可以，产生以下输出：

```
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
```

原因是，和MATLAB中的其他所有事物一样，x计数器也是一个矩阵—准确来说是一个向量。因此，x只是对一个“数组”（一个连贯的、连续的内存结构）的引用，该引用会在每次循环（无论是否嵌套）中被适当

Chapter 8: For loops

Section 8.1: Iterate over columns of matrix

If the right-hand side of the assignment is a matrix, then in each iteration the variable is assigned subsequent columns of this matrix.

```
some_matrix = [1, 2, 3; 4, 5, 6]; % 2 by 3 matrix
for some_column = some_matrix
    display(some_column)
end
```

(The row vector version is a normal case of this, because in MATLAB a row vector is just a matrix whose columns are size 1.)

The output would display

```
1
4
2
5
3
6
```

i.e. each column of the iterated matrix displayed, each column printed on each call of `display`.

Section 8.2: Notice: Weird same counter nested loops

This is not something you will see in other programming environments. I came across it some years back and I couldn't understand why it was happening, but after working with MATLAB for some time I was able to figure it out. Look at the code snippet below:

```
for x = 1:10
    for x = 1:10
        fprintf('%d,', x);
    end
    fprintf('\n');
end
```

you wouldn't expect this to work properly but it does, producing the following output:

```
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
1,2,3,4,5,6,7,8,9,10,
```

The reason is that, as with everything else in MATLAB, the x counter is also a matrix—a vector to be precise. As such, x is only a reference to an 'array' (a coherent, consecutive memory structure) which is appropriately

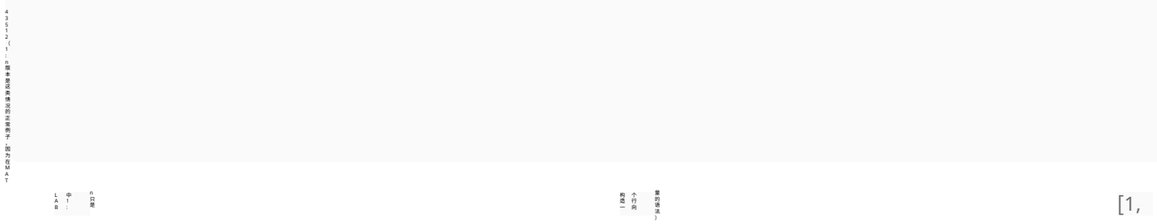
引用。嵌套循环使用相同标识符这一事实并不影响该数组中值的引用方式。唯一的问题是，在嵌套循环中，外层的x被嵌套（局部）x隐藏，因此无法被引用。然而，嵌套循环结构的功能仍然保持完整。

第8.3节：遍历向量元素

for循环中赋值的右侧可以是任何行向量。赋值的左侧可以是任何有效的变量名。for循环每次运行时将该向量的不同元素赋值给变量。

```
other_row_vector = [4, 3, 5, 1, 2];
for any_name = other_row_vector
    display(any_name)
end
```

输出将显示



[1, 2, ..., n].)

因此，以下两个代码块是相同的：

```
A = [1 2 3 4 5];
for x = A
  disp(x);
end
```

和

```
for x = 1:5
  disp(x);
end
```

以下内容也相同：

```
A = [1 3 5 7 9];
for x = A
  disp(x);
end
```

和

```
for x = 1:2:9
  disp(x);
end
```

任何行向量都可以。它们不必是数字。

```
my_characters = 'abcde';
```


referenced with every consequent loop (nested or not). The fact that the nested loop uses the same identifier makes no difference to how values from that array are referenced. The only problem is that within the nested loop the outer `x` is hidden by the nested (local) `x` and therefore cannot be referenced. However, the functionality of the nested loop structure remains intact.

Section 8.3: Iterate over elements of vector

The right-hand side of the assignment in a `for` loop can be any row vector. The left-hand side of the assignment can be any valid variable name. The `for` loop assigns a different element of this vector to the variable each run.

```
other_row_vector = [4, 3, 5, 1, 2];
for any_name = other_row_vector
    display(any_name)
end
```

The output would display

```
4
3
5
1
2
```

(The `1:n` version is a normal case of this, because in MATLAB `1:n` is just syntax for constructing a row vector of `[1, 2, ..., n]`.)

Hence, the two following blocks of code are identical:

```
A = [1 2 3 4 5];
for x = A
  disp(x);
end
```

and

```
for x = 1:5
  disp(x);
end
```

And the following are identical as well:

```
A = [1 3 5 7 9];
for x = A
  disp(x);
end
```

and

```
for x = 1:2:9
  disp(x);
end
```

Any row vector will do. They don't have to be numbers.

```
my_characters = 'abcde';
```

```
for my_char = my_characters
    disp(my_char)
end
```

将输出

```
for my_char = my_characters
    disp(my_char)
end
```

will output

```
a
b
c
d
e
```

Section 8.4: Nested Loops

Loops can be nested, to preform iterated task within another iterated task. Consider the following loops:

```
ch = 'abc';
m = 3;
for c = ch
    for k = 1:m
        disp([c num2str(k)]) % NUM2STR converts the number stored in k to a character,
                             % so it can be concatenated with the letter in c
    end
end
```

we use 2 iterators to display all combinations of elements from `abc` and `1:m`, which yields:

```
a1
a2
a3
b1
b2
b3
c1
c2
c3
```

We can also use nested loops to combine between tasks to be done each time, and tasks to be done once in a several iterations:

```
N = 10;
n = 3;
a1 = 0; % the first element in Fibonacci series
a2 = 1; % the second element in Fibonacci series
for j = 1:N
    for k = 1:n
        an = a1 + a2; % compute the next element in Fibonacci series
        a1 = a2;      % save the previous element for the next iteration
        a2 = an;      % save ht new element for the next iteration
    end
    disp(an) % display every n'th element
end
```

Here we want to compute all the Fibonacci series, but to display only the nth element each time, so we get

```
```

```
```

)

元素之间时才换行。我们在外层循环中遍历间隔宽度，并在内层循环中使用它来
遍历向量：

```
1 2 3 4 5 6 7 8 9 10 11 12
1 3 5 7 9 11
1 4 7 10
1 5 9
1 7
```

第8.5节：循环1到n

最简单的情况就是执行一个固定已知次数的任务。比如我们想显示数字
从1到n，可以写成：

```
n = 5;
for k = 1:n
    display(k)
end
```

循环将执行内部语句，即for和end之间的所有内容，执行 n次（本例中为5次）：

```
1 2 3 4 5 这是另一个例子：n = 5; for k = 1:n
```



```
3
13
55
233
987
4181
17711
75025
317811
1346269
```

Another thing we can do is to use the first (outer) iterator within the inner loop. Here is another example:

```
N = 12;
gap = [1 2 3 4 6];
for j = gap
    for k = 1:j:N
        fprintf('%d ',k) % FPRINTF prints the number k proceeding to the next the line
    end
    fprintf('\n')        % go to the next line
end
```

This time we use the nested loop to format the output, and brake the line only when a new gap (`j`) between the elements was introduced. We loop through the gap width in the outer loop and use it within the inner loop to iterate through the vector:

```
1 2 3 4 5 6 7 8 9 10 11 12
1 3 5 7 9 11
1 4 7 10
1 5 9
1 7
```

Section 8.5: Loop 1 to n

The simplest case is just preforming a task for a fixed known number of times. Say we want to display the numbers between 1 to n, we can write:

```
n = 5;
for k = 1:n
    display(k)
end
```

The loop will execute the inner statement(s), everything between the `for` and the `end`, for n times (5 in this example):

```
1
2
3
4
5
```

Here is another example:

```
n = 5;
for k = 1:n
```

```
    disp(n-k+1:-1:1) % DISP 使用更“干净”的方式在屏幕上打印
end
```

这次我们在循环中同时使用了 n和 k，来创建一个“嵌套”的显示：

```
5    4    3    2    1

4    3    2    1

3    2    1

2    1

1
```

第8.6节：索引循环

```
my_vector = [0, 2, 1, 3, 9];
for i = 1:numel(my_vector)
    my_vector(i) = my_vector(i) + 1;
end
```

使用for循环完成的大多数简单操作都可以通过向量化操作更快更容易地完成。例如，
上面的循环可以用`my_vector = my_vector + 1`替代。


```
    disp(n-k+1:-1:1) % DISP uses more "clean" way to print on the screen
end
```

this time we use both the n and k in the loop, to create a "nested" display:

```
5    4    3    2    1

4    3    2    1

3    2    1

2    1

1
```

Section 8.6: Loop over indexes

```
my_vector = [0, 2, 1, 3, 9];
for i = 1:numel(my_vector)
    my_vector(i) = my_vector(i) + 1;
end
```

Most simple things done with `for` loops can be done faster and easier by vectorized operations. For example, the above loop can be replaced by `my_vector = my_vector + 1`.

第9章：面向对象编程

第9.1节：值类与句柄类

MATLAB中的类分为两大类：值类和句柄类。主要区别在于复制值类的实例时，底层数据会被复制到新实例，而对于句柄类，新实例指向原始数据，修改新实例中的值会影响原始数据。通过继承handle类可以定义一个句柄类。

```
classdef valueClass
    properties
data
    end
end
```

和

```
classdef handleClass < handle
    properties
data
    end
end
```

then

```
>> v1 = valueClass;
>> v1.data = 5;
>> v2 = v1;
>> v2.data = 7;
>> v1.data
ans =
     5

>> h1 = handleClass;
>> h1.data = 5;
>> h2 = h1;
>> h2.data = 7;
>> h1.data
ans =
     7
```

第9.2节：构造函数

构造函数是类中的一种特殊方法，当创建对象实例时会被调用。它是一个普通的MATLAB函数，接受输入参数，但也必须遵循某些规则。

构造函数不是必须的，因为MATLAB会创建一个默认的构造函数。然而，实际上这是定义对象状态的地方。例如，可以通过指定属性来限制属性。然后，构造函数可以通过默认值或用户定义的值来初始化这些属性，这些值实际上可以通过构造函数的输入参数传递。

调用简单类的构造函数

这是一个简单的类Person。

```
classdef Person
    properties
```


Chapter 9: Object-Oriented Programming

Section 9.1: Value vs Handle classes

Classes in MATLAB are divided into two major categories: value classes and handle classes. The major difference is that when copying an instance of a value class, the underlying data is copied to the new instance, while for handle classes the new instance points to the original data and changing values in new instance changes them in the original. A class can be defined as a handle by inheriting from the `handle` class.

```
classdef valueClass
    properties
        data
    end
end
```

and

```
classdef handleClass < handle
    properties
        data
    end
end
```

then

```
>> v1 = valueClass;
>> v1.data = 5;
>> v2 = v1;
>> v2.data = 7;
>> v1.data
ans =
     5

>> h1 = handleClass;
>> h1.data = 5;
>> h2 = h1;
>> h2.data = 7;
>> h1.data
ans =
     7
```

Section 9.2: Constructors

A constructor is a special method in a class that is called when an instance of an object is created. It is a regular MATLAB function that accepts input parameters but it also must follow certain rules.

Constructors are not required as MATLAB creates a default one. In practice, however, this is a place to define a state of an object. For example, properties can be restricted by specifying attributes. Then, a constructor can initalize such properties by default or user defined values which in fact can sent by input parameters of a constructor.

Calling a constructor of a simple class

This is a simple class `Person`.

```
classdef Person
    properties
```

```
name
        surname
        address
    end

    methods
        function obj = Person(name,surname,address)
          obj.name = name;
          obj.surname = surname;
          obj.address = address;
        end
    end
end
```

构造函数的名称与类名相同。因此，构造函数通过其类名来调用。类Person可以如下创建：

```
>> p = Person('John','Smith','London')
p =
Person，具有属性：

name: 'John'
    surname: 'Smith'
    address: 'London'
```

调用子类的构造函数

如果类共享公共属性或方法，则可以从父类继承。当一个类继承自另一个类时，通常需要调用父类的构造函数。

类Member继承自类Person，因为Member使用与类Person相同的属性，但它还在定义中添加了payment属性。

```
classdef Member < Person
    properties
付款
    结束

    methods
        function obj = Member(name,surname,address,payment)
            obj = obj@Person(name,surname,address);
          obj.payment = payment;
        end
    end
end
```

与类Person类似，Member是通过调用其构造函数创建的：

```
>> m = Member('Adam','Woodcock','Manchester',20)
m =
Member，属性如下：

payment: 20
        name: 'Adam'
    surname: 'Woodcock'
    address: 'Manchester'
```

Person的构造函数需要三个输入参数。Member必须遵守这一点，因此调用带有三个参数的Person类的构造函数。该操作由以下代码实现：


```
        name
        surname
        address
    end

    methods
        function obj = Person(name,surname,address)
            obj.name = name;
            obj.surname = surname;
            obj.address = address;
        end
    end
end
```

The name of a constructor is the same the name of a class. Consequently, constructors are called by the name of its class. A class `Person` can be created as follows:

```
>> p = Person('John','Smith','London')
p =
  Person with properties:

       name: 'John'
    surname: 'Smith'
    address: 'London'
```

Calling a constructor of a child class

Classes can be inherited from parent classes if the share common properties or methods. When a class is inherited from another, it is likely that a constructor of a parent class has to be called.

A class `Member` inherits from a class `Person` because `Member` uses the same properties as the class Person but it also adds `payment` to its definition.

```
classdef Member < Person
    properties
        payment
    end

    methods
        function obj = Member(name,surname,address,payment)
            obj = obj@Person(name,surname,address);
            obj.payment = payment;
        end
    end
end
```

Similarly to the class `Person`, `Member` is created by calling its constructor:

```
>> m = Member('Adam','Woodcock','Manchester',20)
m =
  Member with properties:

    payment: 20
       name: 'Adam'
    surname: 'Woodcock'
    address: 'Manchester'
```

A constructor of `Person` requires three input parameters. `Member` must respect this fact and therefore call a constructor of the class `Person` with three parameters. It is fulfilled by the line:

```
obj = obj@Person(name,surname,address);
```

上述示例展示了子类需要其父类信息的情况。这就是为什么Member的构造函数需要四个参数：三个用于其父类，一个用于自身。

第9.3节：定义类

类可以使用classdef在与类同名的.m文件中定义。该文件可以包含classdef...end代码块以及供类方法内部使用的本地函数。

最通用的 MATLAB 类定义具有以下结构：

```
classdef (ClassAttribute = expression, ...) ClassName < ParentClass1 & ParentClass2 & ...

   properties (PropertyAttributes)
      PropertyName
   end

methods (MethodAttributes)
      function obj = methodName(obj,arg2,...)
         ...
      end
   end

events (EventAttributes)
      EventName
   end

enumeration
      EnumName
   end

end
```

MATLAB 文档：类属性，属性属性，方法属性，事件属性，枚举 类限制。

示例类：

一个名为Car的类可以在文件Car.m中定义为

```
classdef Car < handle % handle 类以便属性持久存在
    properties
制造商
        型号
        里程 = 0;
    结束

    methods
        function obj = Car(make, model)
          obj.make = make;
          obj.model = model;
        end
        function drive(obj, milesDriven)
          obj.mileage = obj.mileage + milesDriven;
        end
    end
end
```



```
obj = obj@Person(name,surname,address);
```

The example above shows the case when a child class needs information for its parent class. This is why a constructor of `Member` requires four parameters: three for its parent class and one for itself.

Section 9.3: Defining a class

A class can be defined using `classdef` in an `.m` file with the same name as the class. The file can contain the `classdef`...`end` block and local functions for use within class methods.

The most general MATLAB class definition has the following structure:

```
classdef (ClassAttribute = expression, ...) ClassName < ParentClass1 & ParentClass2 & ...

   properties (PropertyAttributes)
      PropertyName
   end

   methods (MethodAttributes)
      function obj = methodName(obj,arg2,...)
         ...
      end
   end

   events (EventAttributes)
      EventName
   end

   enumeration
      EnumName
   end

end
```

MATLAB Documentation: Class attributes, Property attributes, Method attributes, Event attributes, Enumeration class restrictions.

Example class:

A class called `Car` can be defined in file `Car.m` as

```
classdef Car < handle % handle class so properties persist
    properties
        make
        model
        mileage = 0;
    end

    methods
        function obj = Car(make, model)
            obj.make = make;
            obj.model = model;
        end
        function drive(obj, milesDriven)
            obj.mileage = obj.mileage + milesDriven;
        end
    end
end
```

请注意，构造函数是与类同名的方法。<构造函数是面向对象编程中类或结构的一个特殊方法，用于初始化该类型的对象。构造函数是一个实例方法，通常与类同名，可用于将对象成员的值设置为默认值或用户定义的值。>

可以通过调用构造函数来创建该类的一个实例；

```
>> myCar = Car('Ford', 'Mustang'); //创建一个汽车类的实例
```

调用 drive方法将增加里程数

```
>> myCar.mileage

    ans =
            0

>> myCar.drive(450);

>> myCar.mileage

   ans =
            450
```

第9.4节：从类和抽象类继承

免责声明：此处展示的示例仅用于展示抽象类和继承的用法，可能并不具备实际应用价值。此外，MATLAB中不存在多态性，因此抽象类的使用受到限制。此示例旨在展示如何创建类、继承另一个类以及应用抽象类来定义通用接口。

抽象类在MATLAB中的使用相当有限，但在某些情况下仍然有用。

假设我们想要一个消息记录器。我们可能会创建一个类似下面的类：

```
classdef ScreenLogger
    properties(Access=protected)
       scrh;
    end

    methods
        function obj = ScreenLogger(screenhandler)
          obj.scrh = screenhandler;
        end

        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});fprintf(o
                bj.scrh, '%s', sprintf(varargin{:}));end

        end
    end
end
```

属性和方法

简而言之，属性保存对象的状态，而方法类似于接口，定义了对对象的操作。

Note that the constructor is a method with the same name as the class. <A constructor is a special method of a class or structure in object-oriented programming that initializes an object of that type. A constructor is an instance method that usually has the same name as the class, and can be used to set the values of the members of an object, either to default or to user-defined values.>

An instance of this class can be created by calling the constructor;

```
>> myCar = Car('Ford', 'Mustang'); //creating an instance of car class
```

Calling the `drive` method will increment the mileage

```
>> myCar.mileage

    ans =
            0

>> myCar.drive(450);

>> myCar.mileage

   ans =
            450
```

Section 9.4: Inheriting from classes and abstract classes

Disclaimer: the examples presented here are only for the purpose of showing the use of abstract classes and inheritance and may not necessarily be of a practical use. Also, there is no such thing as polymorphic in MATLAB and therefore the use of abstract classes is limited. This example is to show who to create a class, inherit from another class and apply an abstract class to define a common interface.

The use of abstract classes is rather limited in MATLAB but it still can come useful on a couple of occasions.

Let's say we want a message logger. We might create a class similar to the one below:

```
classdef ScreenLogger
    properties(Access=protected)
        scrh;
    end

    methods
        function obj = ScreenLogger(screenhandler)
            obj.scrh = screenhandler;
        end

        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});
                fprintf(obj.scrh, '%s\n', sprintf(varargin{:}));
            end
        end
    end
end
```

Properties and methods

In short, properties hold a state of an object whilst methods are like interface and define actions on objects.

属性scrh是受保护的。这就是为什么它必须在构造函数中初始化。还有其他方法（getter）可以访问该属性，但这超出了本例的范围。属性和方法可以通过保存对象引用的变量，使用点符号后跟方法名或属性名来访问：

```
mylogger = ScreenLogger(1);                          % 正确
mylogger.LogMessage('我的 %s %d 消息', '非常', 1); % 正确
mylogger.scrh = 2;                                   % 错误！！！访问被拒绝
```

属性和方法可以是公共的、私有的或受保护的。在这种情况下，受保护意味着我可以从继承类访问scrh，但不能从外部访问。默认情况下，所有属性和方法都是公共的。
因此，LogMessage()可以在类定义外自由使用。同时，LogMessage定义了一个接口，意味着当我们想让对象记录自定义消息时，必须调用它。

应用

假设我有一个脚本，使用我的日志记录器：

```
clc;
% ... 一段代码
logger = ScreenLogger(1);
% ... 一段代码
logger.LogMessage('某些内容');
% ... 一段代码
logger.LogMessage('某些内容');
% ... 一段代码
logger.LogMessage('某些内容');
% ... 一段代码
logger.LogMessage('something');
```

如果我在多个地方使用相同的logger，然后想将其更改为更复杂的功能，
比如将消息写入文件，我就必须创建另一个对象：

```
classdef 深度日志器
    properties(SetAccess=protected)
        文件名
    end
    methods
        function obj = 深度日志器(filename)
          obj.文件名 = filename;
        end

        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});
           fid = fopen(obj.fullfname, 'a+t');fprintf(fid, '%s', spri
                ntf(varargin{:}));fclose(fid);

            end
        end
    end
end
```

然后只需将代码中的一行改成这样：

```
clc;
% ... 一段代码
logger = 深度日志器('mymessages.log');
```


The property `scrh` is protected. This is why it must be initialized in a constructor. There are other methods (getters) to access this property but it is out of cope of this example. Properties and methods can be access via a variable that holds a reference to an object by using dot notation followed by a name of a method or a property:

```
mylogger = ScreenLogger(1);                          % OK
mylogger.LogMessage('My %s %d message', 'very', 1); % OK
mylogger.scrh = 2;                                   % ERROR!!! Access denied
```

Properties and methods can be public, private, or protected. In this case, protected means that I will be able to access to `scrh` from an inherited class but not from outside. By default all properties and methods are public. Therefore `LogMessage()` can freely be used outside the class definition. Also `LogMessage` defines an interface meaning this is what we must call when we want an object to log our custom messages.

Application

Let's say I have a script where I utilize my logger:

```
clc;
% ... a code
logger = ScreenLogger(1);
% ... a code
logger.LogMessage('something');
% ... a code
logger.LogMessage('something');
% ... a code
logger.LogMessage('something');
% ... a code
logger.LogMessage('something');
```

If I have multiple places where I use the same logger and then want to change it to something more sophisticated, such as write a message in a file, I would have to create another object:

```
classdef DeepLogger
    properties(SetAccess=protected)
        FileName
    end
    methods
        function obj = DeepLogger(filename)
            obj.FileName = filename;
        end

        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});
                fid = fopen(obj.fullfname, 'a+t');
                fprintf(fid, '%s\n', sprintf(varargin{:}));
                fclose(fid);
            end
        end
    end
end
```

and just change one line of a code into this:

```
clc;
% ... a code
logger = DeepLogger('mymessages.log');
```

上述方法只是简单地打开一个文件，在文件末尾追加一条消息，然后关闭它。目前，为了保持接口的一致性，我需要记住方法名是LogMessage()，但它也可以是其他任何名称。MATLAB可以通过使用抽象类强制开发者遵守相同的名称。假设我们为任何日志器定义一个通用接口：

```
classdef 消息日志器
    methods(Abstract=true)
        LogMessage(obj, varargin);
    end
end
```

现在，如果ScreenLogger和DeepLogger都继承自该类，且LogMessage()未定义，MATLAB将会报错。抽象类有助于构建可以使用相同接口的类似类。

为了这个例子，我将做一些稍微不同的修改。我假设 DeepLogger 会同时在屏幕和文件中记录日志信息。因为ScreenLogger已经在屏幕上记录消息，所以我打算让DeepLogger继承自ScreenLogger以避免重复。除了第一行，ScreenLogger的代码完全不变：

```
classdef ScreenLogger < MessageLogger
// 之前代码的其余部分
```

然而，DeepLogger在LogMessage方法中需要更多的修改：

```
classdef DeepLogger < MessageLogger & ScreenLogger
    properties(SetAccess=protected)
        FileName
        Path
    end
    methods
        function obj = DeepLogger(screenhandler, filename)
            [path,filen,ext] = fileparts(filename);
obj.FileName = [filen ext];
            pbj.Path     = pathn;
obj = obj@ScreenLogger(screenhandler);
        end
        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});
                LogMessage@ScreenLogger(obj, varargin{:});
                fid = fopen(obj.fullfname, 'a+t');
                fprintf(fid, '%s', sprintf(varargin{:}));fclose(fid);

            end
        end
    end
end
```

首先，我仅在构造函数中初始化属性。其次，因为该类继承自ScreenLogger，我必须同时初始化这个父类对象。这一行更为重要，因为ScreenLogger的构造函数需要一个参数来初始化它自己的对象。这一行代码：

```
obj = obj@ScreenLogger(screenhandler);
```

简单来说就是“调用ScreenLogger的构造函数，并用一个屏幕处理器来初始化它”。这里值得注意的是，我将 scrh 定义为受保护的（protected）。因此，我同样可以从DeepLogger访问这个属性。如果该属性被定义为私有（private），唯一的初始化方式就是通过构造函数。

The above method will simply open a file, append a message at the end of the file and close it. At the moment, to be consistent with my interface, I need to remember that the name of a method is `LogMessage()` but it could equally be anything else. MATLAB can force developer to stick to the same name by using abstract classes. Let's say we define a common interface for any logger:

```
classdef MessageLogger
    methods(Abstract=true)
        LogMessage(obj, varargin);
    end
end
```

Now, if both `ScreenLogger` and `DeepLogger` inherit from this class, MATLAB will generate an error if `LogMessage()` is not defined. Abstract classes help to build similar classes which can use the same interface.

For the sake of this example, I will make slightly different change. I am going to assume that DeepLogger will do both logging message on a screen and in a file at the same time. Because `ScreenLogger` already log messages on screen, I am going to inherit `DeepLogger` from the `ScreenLoggger` to avoid repetition. `ScreenLogger` doesn't change at all apart from the first line:

```
classdef ScreenLogger < MessageLogger
// the rest of previous code
```

However, `DeepLogger` needs more changes in the `LogMessage` method:

```
classdef DeepLogger < MessageLogger & ScreenLogger
    properties(SetAccess=protected)
        FileName
        Path
    end
    methods
        function obj = DeepLogger(screenhandler, filename)
            [path,filen,ext] = fileparts(filename);
            obj.FileName = [filen ext];
            pbj.Path     = pathn;
            obj = obj@ScreenLogger(screenhandler);
        end
        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});
                LogMessage@ScreenLogger(obj, varargin{:});
                fid = fopen(obj.fullfname, 'a+t');
                fprintf(fid, '%s\n', sprintf(varargin{:}));
                fclose(fid);
            end
        end
    end
end
```

Firstly, I simply initialize properties in the constructor. Secondly, because this class inherits from `ScreenLogger` I have to initialize this parent object as well. This line is even more important because `ScreenLogger` constructor requires one parameter to initalize its own object. This line:

```
obj = obj@ScreenLogger(screenhandler);
```

simply says "call the constructor of ScreenLogger and initalize it with a screen handler". It is worth noting here that I have defined `scrh` as protected. Therefore, I could equally access this property from `DeepLogger`. If the property was defined as private. The only way to intialize it would be using the constructor.

另一个变化是在methods部分。为了避免重复，我调用父类的LogMessage()来在屏幕上记录消息。如果我需要对屏幕日志功能做任何改进，现在只需在一个地方修改即可。其余代码保持不变，因为它是DeepLogger的一部分。

由于该类还继承自一个抽象类MessageLogger，我必须确保DeepLogger中的LogMessage()也被定义。继承自MessageLogger在这里有点棘手。我认为这使得重新定义LogMessage成为必须——这是我的猜测。

在应用记录器的代码方面，得益于类中通用的接口，我可以放心地认为，在整个代码中只修改这一行不会引发任何问题。相同的消息将像以前一样记录在屏幕上，此外代码还会将这些消息写入文件。

```
clc;
% ... 一段代码
logger = DeepLogger(1, 'mylogfile.log');
% ... 一段代码
logger.LogMessage('某些内容');
% ... 一段代码
logger.LogMessage('某些内容');
% ... 一段代码
logger.LogMessage('某些内容');
% ... 一段代码
logger.LogMessage('something');
```

我希望这些示例解释了类的使用、继承的使用以及抽象类的使用。

附注：上述问题的解决方案只是众多方案中的一种。另一种较为简单的解决方案是ScreenLogger作为另一个记录器组件，如FileLogger等的一个部分。ScreenLogger会被保存在某个属性中。它的LogMessage方法仅仅调用ScreenLogger的LogMessage并在屏幕上显示文本。我选择了更复杂的方法来展示MATLAB中类的工作原理。以下是示例代码：

```
classdef DeepLogger < MessageLogger
    properties(SetAccess=protected)
       FileName
        Path
ScrLogger
    end
    methods
        function obj = DeepLogger(screenhandler, filename)
            [path,filen,ext] = fileparts(filename);
obj.FileName    = [filen ext];
            obj.Path         = pathn;
obj.ScrLogger    = ScreenLogger(screenhandler);
        end
        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});
obj.LogMessage(obj.ScrLogger, varargin{:}); % <-------- 这里的改动
                fid = fopen(obj.fullfname, 'a+t');
                fprintf(fid, '%s', sprintf(varargin{:}));fclose(fid);

            end
        end
    end
end
```


Another change is in section `methods`. Again to avoid repetition, I call `LogMessage()` from a parent class to log a message on a screen. If I had to change anything to make improvements in screen logging, now I have to do it in one place. The rest code is the same as it is a part of `DeepLogger`.

Because this class also inherits from an abstract class `MessageLogger` I had to make sure that `LogMessage()` inside `DeepLogger` is also defined. Inheriting from `MessageLogger` is a little bit tricky here. I think it cases redefinition of `LogMessage` mandatory--my guess.

In terms of the code where the a logger is applied, thanks to a common interface in classes, I can rest assured that changing this one line in the whole code would not make any issues. The same messages will be log on screen as before but additionally the code will write such messages to a file.

```
clc;
% ... a code
logger = DeepLogger(1, 'mylogfile.log');
% ... a code
logger.LogMessage('something');
% ... a code
logger.LogMessage('something');
% ... a code
logger.LogMessage('something');
% ... a code
logger.LogMessage('something');
```

I hope these examples explained the use of classes, the use of inheritance, and the use of abstract classes.

PS. The solution for the above problem is one of many. Another solution, less complex, would be to make `ScreenLoger` to be a component of another logger like `FileLogger` etc. `ScreenLogger` would be held in one of the properties. Its `LogMessage` would simply call `LogMessage` of the `ScreenLogger` and show text on a screen. I have chosen more complex approach to rather show how classes work in MATLAB. The example code below:

```
classdef DeepLogger < MessageLogger
    properties(SetAccess=protected)
        FileName
        Path
        ScrLogger
    end
    methods
        function obj = DeepLogger(screenhandler, filename)
            [path,filen,ext] = fileparts(filename);
            obj.FileName     = [filen ext];
            obj.Path         = pathn;
            obj.ScrLogger    = ScreenLogger(screenhandler);
        end
        function LogMessage(obj, varargin)
            if ~isempty(varargin)
                varargin{1} = num2str(varargin{1});
                obj.LogMessage(obj.ScrLogger, varargin{:}); % <-------- the change here
                fid = fopen(obj.fullfname, 'a+t');
                fprintf(fid, '%s\n', sprintf(varargin{:}));
                fclose(fid);
            end
        end
    end
end
```

第10章：向量化

第10.1节：bsxfun的使用

代码经常使用for循环的原因是为了从“附近”的值计算新值。bsxfun函数通常可以用更简洁的方式实现这一点。

例如，假设您希望对矩阵B执行按列操作，从每列中减去该列的均值：

```
B = round(randn(5)*10);                 % 生成随机数据
A = zeros(size(B));                     % 预分配数组
for col = 1:size(B,2);                   % 遍历列
    A(:,col) = B(:,col) - mean(B(:,col));   % 减去均值
end
```

如果B很大，这种方法效率低下，通常是因为MATLAB需要在内存中移动变量内容。通过使用bsxfun，可以仅用一行代码整洁且轻松地完成相同的工作：

```
A = bsxfun(@minus, B, mean(B));
```

这里，@minus是minus运算符（-）的函数句柄，将应用于两个矩阵B和mean(B)的元素之间。其他函数句柄，甚至用户自定义的，也可以使用。

接下来，假设您想将行向量v加到矩阵A的每一行：

```
v = [1,  2,  3];

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

最简单的做法是使用循环（不要这样做）：

```
B = zeros(3);
for 行 = 1:3
B(行,:) = A(行,:) + v;
end
```

另一种选择是用 v 和 repmat 复制（也不要这样做）：

```
>> v = repmat(v,3,1)
v =
     1     2     3
     1     2     3
     1     2     3

>> B = A + v;
```

相反，可以使用 bsxfun 来完成此任务：

```
>> B = bsxfun(@plus, A, v);
B =
     9     3     9
     4     7    10
```


Chapter 10: Vectorization

Section 10.1: Use of bsxfun

Quite often, the reason why code has been written in a `for` loop is to compute values from 'nearby' ones. The function `bsxfun` can often be used to do this in a more succinct fashion.

For example, assume that you wish to perform a columnwise operation on the matrix `B`, subtracting the mean of each column from it:

```
B = round(randn(5)*10);                 % Generate random data
A = zeros(size(B));                     % Preallocate array
for col = 1:size(B,2);                   % Loop over columns
    A(:,col) = B(:,col) - mean(B(:,col));   % Subtract means
end
```

This method is inefficient if `B` is large, often due to MATLAB having to move the contents of variables around in memory. By using `bsxfun`, one can do the same job neatly and easily in just a single line:

```
A = bsxfun(@minus, B, mean(B));
```

Here, `@minus` is a function handle to the `minus` operator (`-`) and will be applied between elements of the two matrices `B` and `mean(B)`. Other function handles, even user-defined ones, are possible as well.

Next, suppose you want to add row vector `v` to each row in matrix `A`:

```
v = [1,  2,  3];

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

The naive approach is use a loop (*do **not** do this*):

```
B = zeros(3);
for row = 1:3
    B(row,:) = A(row,:) + v;
end
```

Another option would be to replicate `v` with `repmat` (*do **not** do this either*):

```
>> v = repmat(v,3,1)
v =
     1     2     3
     1     2     3
     1     2     3

>> B = A + v;
```

Instead use `bsxfun` for this task:

```
>> B = bsxfun(@plus, A, v);
B =
     9     3     9
     4     7    10
```

```
     5    11     5
```

语法

bsxfun(@fun, A, B)

其中 @fun 是支持的函数之一，且两个数组 A 和 B 满足以下两个条件。

函数名 bsxfun 有助于理解其工作原理，代表带有单例扩展的二元函数（Binary FUNction with Singleton eXpansion）。换句话说，如果：

1.两个数组除了一个维度外，其余维度相同
2. 且不匹配的维度在任一数组中是单例维度（即大小为 1）

那么具有单例维度的数组将被扩展以匹配另一个数组的维度。扩展后，二元函数将在两个数组的对应元素上逐元素应用。

例如，设 A 是一个 M-行-N-列-K 层的数组，B 是一个 M-行-N-列的数组。首先，它们的前两个维度大小对应。其次，A 有 K 层，而 B 隐式地只有 1 层，因此是单例维度。所有 条件 都满足，B 将被复制以匹配 A 的第三维度。

在其他语言中，这通常称为广播（broadcasting），并且在 Python（numpy）和 Octave 中会自动发生。

函数 @fun 必须是二元函数，意味着它必须恰好接受两个输入。

备注

内部，bsxfun 不复制数组，而是执行高效的循环。

第10.2节：隐式数组扩展（广播）[R2016b]

MATLAB R2016b 引入了其标量扩展1,2 机制的泛化，也支持不同大小数组之间的某些元素逐个操作，只要它们的维度兼容。支持隐式扩展的运算符有1：

- **元素逐个算术运算符：+，-，.*，.^，./，.\。**
- **关系运算符：<，<=，>，>=，==，~=。**
- **逻辑运算符：&，|，xor。**
- **按位函数：bitand，bitor，bitxor。**
- 基本数学函数：max，min，mod，rem，hypot，atan2，atan2d。

上述二元运算允许在数组之间进行，只要它们具有“兼容的大小”。当一个数组的每个维度要么与另一个数组的相同维度完全相等，要么等于1时，大小被视为“兼容”。注意，MATLAB会省略尾部的单例维度（即大小为1的维度），尽管理论上它们是无限的。换句话说——出现在一个数组中但不出现在另一个数组中的维度，隐式地适合自动扩展。

例如，在R2016b之前的MATLAB版本中会发生以下情况：

```
>> magic(3) + (1:3)
使用 + 时出错
矩阵维度必须一致。
```



```
     5    11     5
```

Syntax

`bsxfun(@fun, A, B)`

where `@fun` is one of the supported functions and the two arrays `A` and `B` respect the two conditions below.

The name `bsxfun` helps to understand how the function works and it stands for **B**inary **FUN**ction with **S**ingleton e**X**pansion. In other words, if:

1. two arrays share the same dimensions except for one
2. and the discordant dimension is a singleton (i.e. has a size of 1) in either of the two arrays

then the array with the singleton dimension will be expanded to match the dimension of the other array. After the expansion, a binary function is applied elementwise on the two arrays.

For example, let `A` be an `M`-by-`N`-by`K` array and `B` is an `M`-by-`N` array. Firstly, their first two dimensions have corresponding sizes. Secondly, `A` has `K` layers while `B` has implicitly only `1`, hence it is a singleton. All **conditions** are met and `B` will be replicated to match the 3rd dimension of `A`.

In other languages, this is commonly referred to as *broadcasting* and happens automatically in Python (numpy) and Octave.

The function, `@fun`, must be a binary function meaning it must take exactly two inputs.

Remarks

Internally, `bsxfun` does not replicate the array and executes an efficient loop.

Section 10.2: Implicit array expansion (broadcasting) [R2016b]

MATLAB R2016b featured a generalization of its scalar expansion1,2 mechanism, to also support certain element-wise operations between *arrays* of different sizes, as long as their dimension are compatible. The operators that support implicit expansion are1:

- **Element-wise arithmetic operators:** `+`, `-`, `.*`, `.^`, `./`, `.\`.
- **Relational operators:** `<`, `<=`, `>`, `>=`, `==`, `~=`.
- **Logical operators:** `&`, `|`, `xor`.
- **Bit-wise functions:** `bitand`, `bitor`, `bitxor`.
- **Elementary math functions:** `max`, `min`, `mod`, `rem`, `hypot`, `atan2`, `atan2d`.

The aforementioned binary operations are allowed between arrays, as long as they have "compatible sizes". Sizes are considered "compatible" when each dimension in one array is either exactly equal to the same dimension in the other array, or is equal to 1. Note that trailing singleton (that is, of size 1) dimensions are omitted by MATLAB, even though there's theoretically an infinite amount of them. In other words - dimensions that appear in one array and do not appear in the other, are implicitly fit for automatic expansion.

For example, in MATLAB versions **before R2016b** this would happen:

```
>> magic(3) + (1:3)
Error using  +
Matrix dimensions must agree.
```

而从 R2016b 开始，之前的操作将成功：

```
>> magic(3) + (1:3)
ans =

     9     3     9
     4     7    10
     5    11     5
```

兼容尺寸的示例：

描述	第1个数组尺寸	第2个数组尺寸	结果尺寸
向量和标量	[3x1]	[1x1]	[3x1]
行向量和列向量 [1x3]		[2x1]	[2x3]
向量和二维矩阵	[1x3]	[5x3]	[5x3]
N维和K维数组	[1x3x3]	[5x3x1x4x2]	[5x3x3x4x2]

不兼容尺寸示例：

描述	第一个数组尺寸	第二个数组尺寸	可能的解决方法
一个维度是另一个数组中相同维度的倍数的向量。	[1x2]	[1x8]	转置
维度彼此成倍数关系的数组。	[2x2]	[8x8]	repmat，reshape
具有正确数量的单例维度但顺序错误的N维数组（#1）。维度顺序错误的N维数组（#1）。	[2x3x4]	[2x4x3]	`permute`
具有正确数量的单例维度但顺序错误的N维数组（#2）。	[2x3x4x5]	[5x2]	`permute`

重要：

依赖此约定的代码不向后兼容任何旧版本的MATLAB。因此，如果代码需要在旧版本MATLAB上运行，应显式调用bsxfun1,2（实现相同效果）。如果不存在此类顾虑，MATLAB R2016版本说明鼓励用户从bsxfun切换：

> *与使用bsxfun相比，隐式扩展提供更快的执行速度、更好的内存使用以及更易读的代码。*

相关阅读：

- *MATLAB文档中关于"基本操作的兼容数组大小"的内容。*
- NumPy的广播1,2.
- 使用bsxfun与隐式数组扩展计算速度的比较。

第10.3节：逐元素操作

MATLAB支持（并鼓励）对向量和矩阵进行矢量化操作。
例如，假设我们有两个n行m列的矩阵A和B，我们希望C是对应元素的逐元素乘积（即，C(i,j)= A(i,j)*B(i,j)）。

非矢量化的方法，使用嵌套循环如下：

```
C = zeros(n,m);
for ii=1:n
```


Whereas **starting from R2016b** the previous operation will succeed:

```
>> magic(3) + (1:3)
ans =

     9     3     9
     4     7    10
     5    11     5
```

Examples of compatible sizes:

Description	1st Array Size	2nd Array Size	Result Size
Vector and scalar	[3x1]	[1x1]	[3x1]
Row and column vectors	[1x3]	[2x1]	[2x3]
Vector and 2D matrix	[1x3]	[5x3]	[5x3]
N-D and K-D arrays	[1x3x3]	[5x3x1x4x2]	[5x3x3x4x2]

Examples of incompatible sizes:

Description	1st Array Size	2nd Array Size	Possible Workaround
Vectors where a dimension is a multiple of the same dimension in the other array.	[1x2]	[1x8]	`transpose`
Arrays with dimensions that are multiples of each other.	[2x2]	[8x8]	`repmat`, `reshape`
N-D arrays that have the right amount of singleton dimensions but they're in the wrong order (#1).	[2x3x4]	[2x4x3]	`permute`
N-D arrays that have the right amount of singleton dimensions but they're in the wrong order (#2).	[2x3x4x5]	[5x2]	`permute`

IMPORTANT:

Code relying on this convention is **NOT** backward-compatible with *any* older versions of MATLAB. Therefore, the explicit invocation of `bsxfun`1,2 (which achieves the same effect) should be used if code needs to run on older MATLAB versions. If such a concern does not exist, MATLAB R2016 release notes encourage users to switch from `bsxfun`:

> *Compared to using `bsxfun`, implicit expansion offers faster speed of execution, better memory usage, and improved readability of code.*

Related reading:

- MATLAB documentation on "*Compatible Array Sizes for Basic Operations*".
- NumPy's Broadcasting1,2.
- A comparison between the speed of computing using `bsxfun` vs. implicit array expansion.

Section 10.3: Element-wise operations

MATLAB supports (and encourages) vectorized operations on vectors and matrices.
For example, suppose we have `A` and `B`, two `n`-by-`m` matrices and we want `C` to be the element-wise product of the corresponding elements (i.e., `C(i,j) = A(i,j)*B(i,j)`).

The un-vectorized way, using nested loops is as follows:

```
C = zeros(n,m);
for ii=1:n
```

```
    for jj=1:m
C(ii,jj) = A(ii,jj)*B(ii,jj);
    end
end
```

然而，矢量化的方法是使用逐元素运算符.*：

```
C = A.*B;
```

- 有关 MATLAB 中元素逐个相乘的更多信息，请参见 times 的文档。
- 有关数组运算和矩阵运算之间差异的更多信息，请参见 MATLAB 文档中的 数组与矩阵运算。

第10.4节：逻辑掩码

MATLAB 支持使用逻辑掩码，以便在不使用 for 循环或 if 语句的情况下对矩阵进行选择操作。

逻辑掩码定义为仅由 1 和 0 组成的矩阵。

例如：

```
mask = [1 0 0; 0 1 0; 0 0 1];
```

是表示单位矩阵的逻辑矩阵。

我们可以使用谓词对矩阵进行查询，从而生成逻辑掩码。

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

我们首先创建一个包含数字1到9的3x3矩阵A。然后查询A中大于4的值，并将结果存储在一个名为B的新矩阵中。

B是一个逻辑矩阵，形式如下：

```
B = [0 0 0
     0 1 1
     1 1 1]
```

当谓词A > 4为真时，值为1；为假时，值为0。

我们可以使用逻辑矩阵来访问矩阵的元素。如果使用逻辑矩阵选择元素，则逻辑矩阵中出现1的索引将在被选择的矩阵中被选中。

使用上面相同的B，我们可以执行以下操作：

```
C = [0 0 0; 0 0 0; 0 0 0];
C(B) = 5;
```

这将选择C中所有对应B中为1的元素。然后将C中这些索引的值设置为5。

我们的C现在看起来像这样：


```
    for jj=1:m
        C(ii,jj) = A(ii,jj)*B(ii,jj);
    end
end
```

However, the vectorized way of doing this is by using the element-wise operator `.*`:

```
C = A.*B;
```

- For more information on the element-wise multiplication in MATLAB see the documentation of `times`.
- For more information about the difference between array and matrix operations see Array vs. Matrix Operations in the MATLAB documentation.

Section 10.4: Logical Masking

MATLAB supports the use of logical masking in order to perform selection on a matrix without the use of for loops or if statements.

A logical mask is defined as a matrix composed of only `1` and `0`.

For example:

```
mask = [1 0 0; 0 1 0; 0 0 1];
```

is a logical matrix representing the identity matrix.

We can generate a logical mask using a predicate to query a matrix.

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

We first create a 3x3 matrix, `A`, containing the numbers 1 through 9. We then query `A` for values that are greater than 4 and store the result in a new matrix called `B`.

`B` is a logical matrix of the form:

```
B = [0 0 0
     0 1 1
     1 1 1]
```

Or `1` when the predicate `A > 4` was true. And `0` when it was false.

We can use logical matrices to access elements of a matrix. If a logical matrix is used to select elements, indices where a `1` appear in the logical matrix will be selected in the matrix you are selecting from.

Using the same `B` from above, we could do the following:

```
C = [0 0 0; 0 0 0; 0 0 0];
C(B) = 5;
```

This would select all of the elements of `C` where `B` has a `1` in that index. Those indices in `C` are then set to `5`.

Our `C` now looks like:

```
C = [0 0 0
     0 5 5
     5 5 5]
```

我们可以通过使用逻辑掩码来简化包含if和for的复杂代码块。

以非向量化代码为例：

```
A = [1 3 5; 7 9 11; 11 9 7];
for j = 1:length(A)
  if A(j) > 5
    A(j) = A(j) - 2;
  end
end
```

这可以用逻辑掩码缩短为以下代码：

```
A = [1 3 5; 7 9 11; 11 9 7];
B = A > 5;
A(B) = A(B) - 2;
```

甚至更短：

```
A = [1 3 5; 7 9 11; 11 9 7];
A(A > 5) = A(A > 5) - 2;
```

第10.5节：求和、均值、乘积等

给定一个随机向量

```
v = rand(10,1);
```

如果你想求其元素的和，不要使用循环

```
s = 0;
for ii = 1:10
    s = s + v(ii);
end
```

而应使用 sum() 函数的向量化功能

```
s = sum(v);
```

像 sum()、mean()、prod() 等函数，能够直接沿行、列或其他维度进行操作。

例如，给定一个随机矩阵

```
A = rand(10,10);
```

每一列的平均值是

```
m = mean(A,1);
```

每一行的平均值是


```
C = [0 0 0
     0 5 5
     5 5 5]
```

We can reduce complicated code blocks containing `if` and `for` by using logical masks.

Take the non-vectorized code:

```
A = [1 3 5; 7 9 11; 11 9 7];
for j = 1:length(A)
  if A(j) > 5
    A(j) = A(j) - 2;
  end
end
```

This can be shortened using logical masking to the following code:

```
A = [1 3 5; 7 9 11; 11 9 7];
B = A > 5;
A(B) = A(B) - 2;
```

Or even shorter:

```
A = [1 3 5; 7 9 11; 11 9 7];
A(A > 5) = A(A > 5) - 2;
```

Section 10.5: Sum, mean, prod & co

Given a random vector

```
v = rand(10,1);
```

if you want the sum of its elements, do **NOT** use a loop

```
s = 0;
for ii = 1:10
    s = s + v(ii);
end
```

but use the vectorized capability of the `sum()` function

```
s = sum(v);
```

Functions like `sum()`, `mean()`, `prod()` and others, have the ability to operate directly along rows, columns or other dimensions.

For instance, given a random matrix

```
A = rand(10,10);
```

the average for each **column** is

```
m = mean(A,1);
```

the average for each **row** is

```
m = mean(A,2)
```

以上所有函数都只在一个维度上工作，但如果你想对整个矩阵求和呢？你可以使用：

```
s = sum(sum(A))
```

但是如果有一个多维数组？对 sum 连续使用多次似乎不是最佳选择，改用 :
操作符来向量化你的数组：

```
s = sum(A(:))
```

这将得到一个数字，即你数组中所有元素的和，无论它有多少维度。

第10.6节：获取两个或多个参数函数的值

在许多应用中，需要计算两个或多个参数的函数值。

传统上，我们使用for循环。例如，如果我们需要计算f = exp(-x^2-y^2)（如果需要快速模拟，请勿使用此方法）：

```
% code1
x = -1.2:0.2:1.4;
y = -2:0.25:3;
for nx=1:lenght(x)
    for ny=1:lenght(y)
f(nx,ny) = exp(-x(nx)^2-y(ny)^2);
    end
end
```

但向量化版本更优雅且更快：

```
% code2
[x,y] = ndgrid(-1.2:0.2:1.4, -2:0.25:3);
f = exp(-x.^2-y.^2);
```

然后我们可以将其可视化：

```
surf(x,y,f)
```

注1 - 网格：通常，矩阵存储是按行优先组织的。但在MATLAB中，存储方式是类似FORTRAN的列优先存储。因此，MATLAB中有两个类似的函数ndgrid和meshgrid来实现上述两种模型。对于meshgrid的函数可视化，我们可以使用：

```
surf(y,x,f)
```

注2 - 内存消耗：设 x或 y的大小为1000。因此，对于非矢量化的code1，我们需要存储1000*1000+2*1000约1e6个元素。但对于矢量化的code2，需要存储3*(1000*1000)= 3e6个元素。在三维情况下（设 z与 x或 y大小相同），内存消耗显著增加：矢量化的code2需要4*(1000*1000*1000)（约32GB，双精度）而code1仅需约1000*1000*1000（约8GB）。因此，我们必须在内存和速度之间做出选择。


```
m = mean(A,2)
```

All the functions above work only on one dimension, but what if you want to sum the whole matrix? You could use:

```
s = sum(sum(A))
```

But what if have an ND-array? applying `sum` on `sum` on `sum`... don't seem like the best option, instead use the : operator to vectorize your array:

```
s = sum(A(:))
```

and this will result in one number which is the sum of all your array, doesn't matter how many dimensions it have.

Section 10.6: Get the value of a function of two or more arguments

In many application it is necessary to compute the function of two or more arguments.

Traditionally, we use `for`-loops. For example, if we need to calculate the `f = exp(-x^2-y^2)` (do not use this if you need **fast simulations**):

```
% code1
x = -1.2:0.2:1.4;
y = -2:0.25:3;
for nx=1:lenght(x)
   for ny=1:lenght(y)
      f(nx,ny) = exp(-x(nx)^2-y(ny)^2);
   end
end
```

But vectorized version is more elegant and faster:

```
% code2
[x,y] = ndgrid(-1.2:0.2:1.4, -2:0.25:3);
f = exp(-x.^2-y.^2);
```

than we can visualize it:

```
surf(x,y,f)
```

Note1 - Grids: Usually, the matrix storage is organized *row-by-row*. But in the MATLAB, it is the *column-by-column* storage as in FORTRAN. Thus, there are two similar functions `ndgrid` and `meshgrid` in MATLAB to implement the two aforementioned models. To visualise the function in the case of `meshgrid`, we can use:

```
surf(y,x,f)
```

Note2 - Memory consumption: Let size of x or y is 1000. Thus, we need to store `1000*1000+2*1000 ~ 1e6` elements for non-vectorized ***code1***. But we need `3*(1000*1000) = 3e6` elements in the case of vectorized ***code2***. In the 3D case (let z has the same size asx or y), memory consumption increases dramatically: `4*(1000*1000*1000)` (~32GB for doubles) in the case of the vectorized ***code2*** vs ~`1000*1000*1000` (just ~8GB) in the case of ***code1***. Thus, we have to choose either the memory or speed.

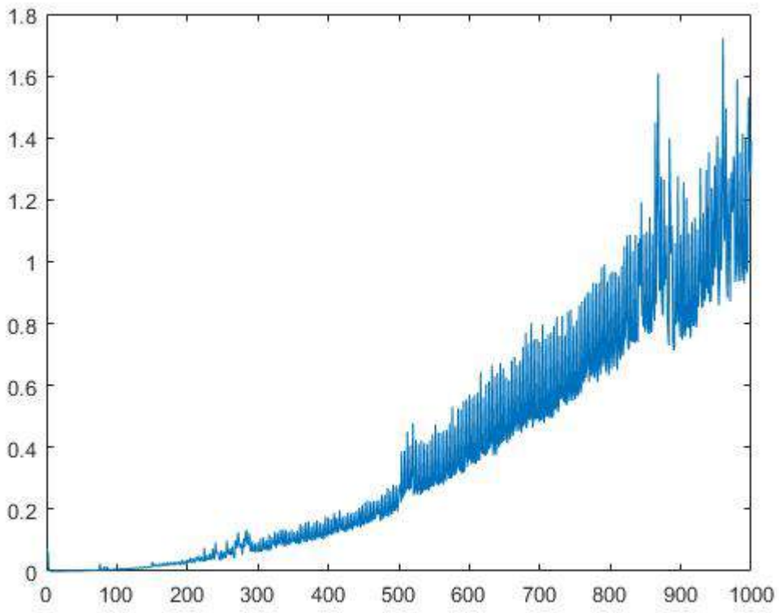
第11章：矩阵分解

第11.1节：Schur分解

如果A是复数方阵，则存在一个酉矩阵Q，使得Q*AQ = T = D + N，其中D是由特征值组成的对角矩阵，N是严格的上三对角矩阵。

```
A = [3 6 1
     23 13 1
     0 3 4];
T = schur(A);
```

我们还展示了 schur 的运行时间，取决于矩阵元素的平方根：



第11.2节：Cholesky分解

Cholesky分解是一种将埃尔米特（Hermitean）正定矩阵分解为上三角矩阵及其转置的方法。它可用于求解线性方程组，速度大约是LU分解的两倍。

```
A = [4 12 -16
     12 37 -43
-16 -43 98];
R = chol(A);
```

这将返回上三角矩阵。下三角矩阵通过转置获得。

```
L = R';
```

我们终于可以检查分解是否正确。

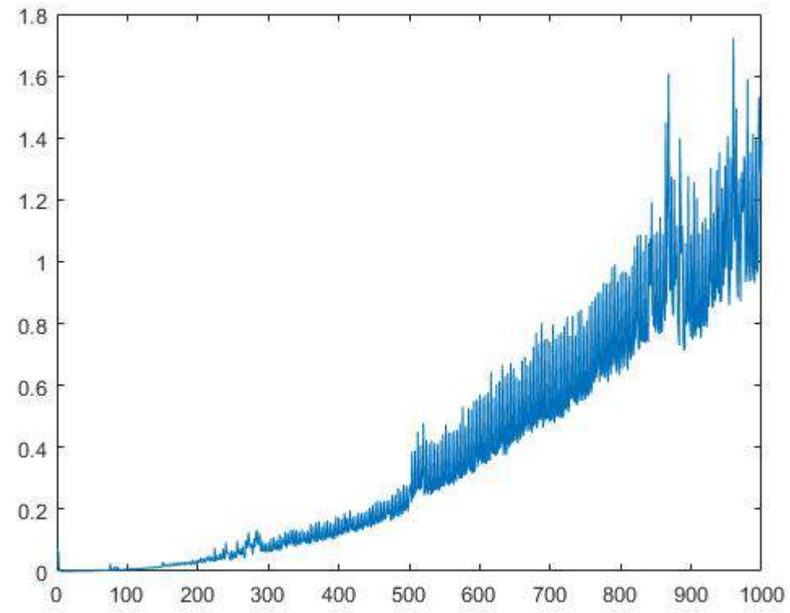
Chapter 11: Matrix decompositions

Section 11.1: Schur decomposition

If A is a complex and quadratic matrix there exists a unitary Q such that Q*AQ = T = D + N with D being the diagonal matrix consisting of the eigenvalues and N being strictly upper tridiagonal.

```
A = [3 6 1
     23 13 1
     0 3 4];
T = schur(A);
```

We also display the runtime of `schur` dependent on the square root of matrix elements:



Section 11.2: Cholesky decomposition

The Cholesky decomposition is a method to decompose an Hermitean, positive definite matrix into an upper triangular matrix and its transpose. It can be used to solve linear equations systems and is around twice as fast as LU-decomposition.

```
A = [4 12 -16
     12 37 -43
     -16 -43 98];
R = chol(A);
```

This returns the upper triangular matrix. The lower one is obtained by transposition.

```
L = R';
```

We finally can check whether the decomposition was correct.

```
b = (A == L*R);
```

第11.3节：QR分解

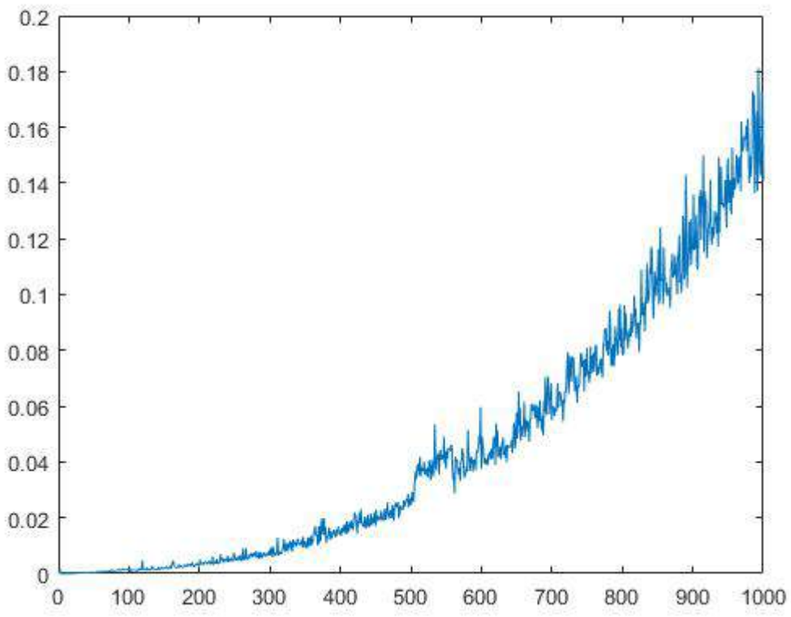
该方法将把矩阵分解为一个上三角矩阵和一个正交矩阵。

```
A = [4 12 -16
     12 37 -43
-16 -43 98];
R = qr(A);
```

这将返回上三角矩阵，而下面的将返回两个矩阵。

```
[Q,R] = qr(A);
```

下面的图将显示qr运行时间，依赖于矩阵元素平方根。



第11.4节：LU分解

这里矩阵将被分解为一个上三角矩阵和一个下三角矩阵。通常它将被用来提高高斯消元法的性能和稳定性（如果使用置换的话）。

然而，这种方法经常不起作用或效果很差，因为它不稳定。例如

```
A = [8 1 6
     3 5 7
     4 9 2];
[L,U] = lu(A);
```

只需添加一个置换矩阵，使得 PA=LU：

```
[L,U,P]=lu(A);
```



```
b = (A == L*R);
```

Section 11.3: QR decomposition

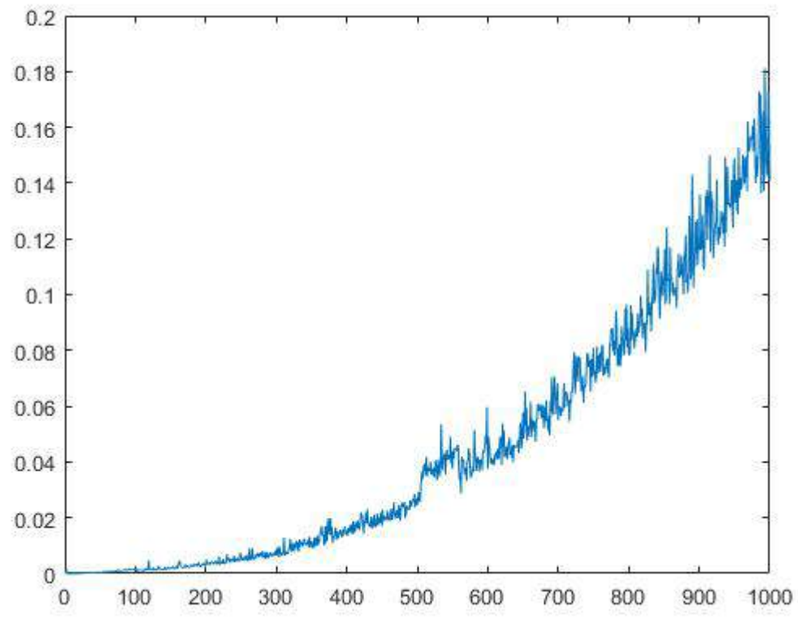
This method will decompose a matrix into an upper triangular and an orthogonal matrix.

```
A = [4 12 -16
     12 37 -43
     -16 -43 98];
R = qr(A);
```

This will return the upper triangular matrix while the following will return both matrices.

```
[Q,R] = qr(A);
```

The following plot will display the runtime of `qr` dependent of the square root of elements of the matrix.



Section 11.4: LU decomposition

Hereby a matrix will be decomposed into an upper triangular and an lower triangular matrix. Often it will be used to increase the performance and stability (if it's done with permutation) of Gauß elimination.

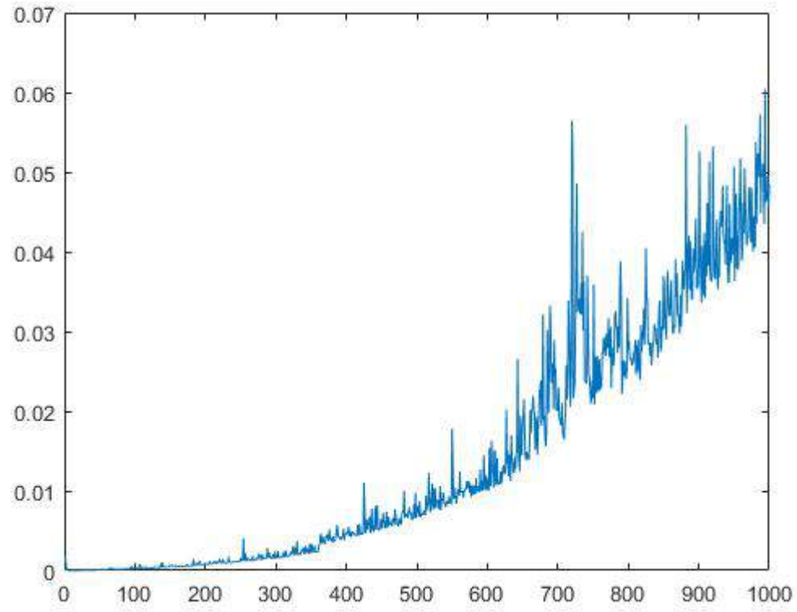
However, quite often does this method not or badly work as it is not stable. For example

```
A = [8 1 6
     3 5 7
     4 9 2];
[L,U] = lu(A);
```

It is sufficient to add an permutation matrix such that PA=LU:

```
[L,U,P]=lu(A);
```

接下来我们将绘制 `lu` 的运行时间，作为矩阵元素平方根的函数。



第11.5节：奇异值分解

给定一个 m 行 n 列的矩阵 A，且 n 大于 m。奇异值分解为

```
[U,S,V] = svd(A);
```

计算矩阵 U、S、V。

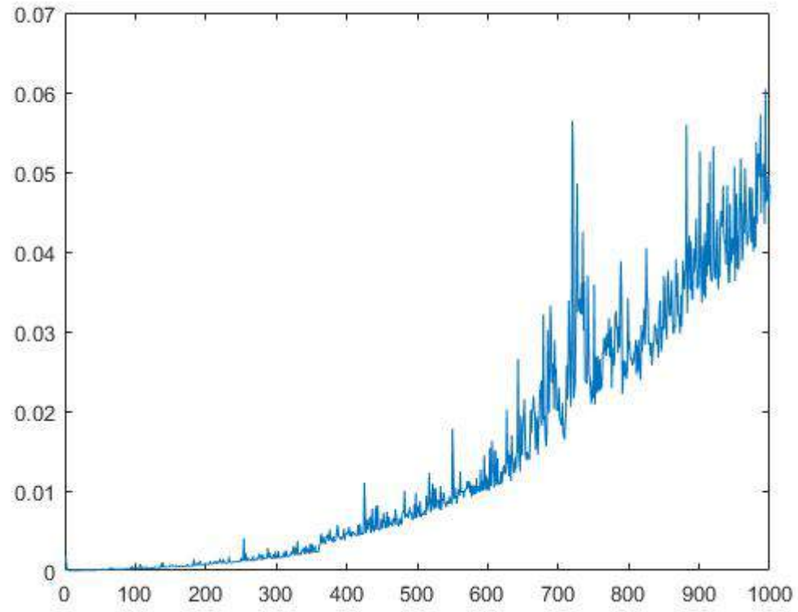
矩阵U由左奇异特征向量组成，这些向量是A*A.'的特征向量，而矩阵V由右奇异特征值组成，这些是A.'*A的特征向量。矩阵S的对角线上是A*A.'和A.'*A的特征值的平方根。

如果m大于n，可以使用

```
[U,S,V] = svd(A,'econ');
```

来执行经济尺寸的奇异值分解。

In the following we will now plot the runtime of `lu' dependent of the square root of elements of the matrix.



Section 11.5: Singular value decomposition

Given an m times n matrix A with n larger than m. The singular value decomposition

```
[U,S,V] = svd(A);
```

computes the matrices U,S,V.

The matrix U consists of the left singular eigenvectors which are the eigenvectors of `A*A.'` while V consists of the right singular eigenvalues which are the eigenvectors of `A.'*A`. The matrix `S` has the square roots of the eigenvalues of `A*A.'` and `A.'*A` on its diagonal.

If m is larger than n one can use

```
[U,S,V] = svd(A,'econ');
```

to perform economy sized singular value decomposition.

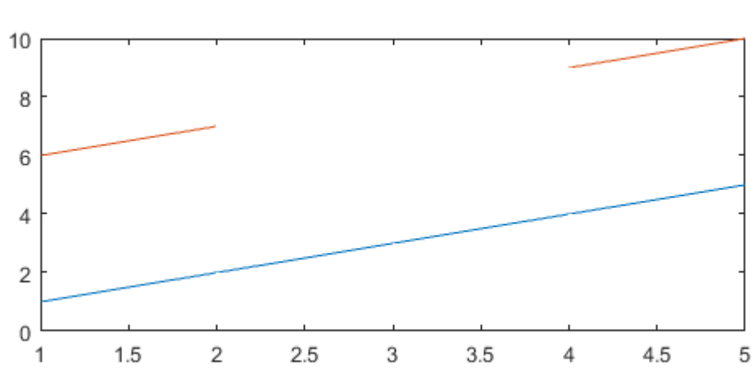
第12章：图形：二维折线图

参数	详细信息
X	x值
Y	y值
LineSpec	线型、标记符号和颜色，指定为字符串
名称，值	可选的名称-值对参数，用于自定义线条属性
h	线条图形对象的句柄

第12.1节：用NaN分割线条

将你的y或x值与NaN交错排列

```
x = [1:5; 6:10]';

x(3,2) = NaN
x =
     1     6
     2     7
     3   NaN
     4     9
     5    10

plot(x)
```



第12.2节：单个图中的多条线

在本例中，我们将在同一坐标轴上绘制多条线。此外，我们为这些线选择不同的外观并创建图例。

```
% 创建示例数据
x = linspace(-2,2,100);                 % 从-2到2的100个线性间隔点
y1 = x.^2;
y2 = 2*x.^2;
y3 = 4*x.^2;

% 创建图形
figure;                                 % 打开新图形
plot(x,y1, x,y2,'--', x,y3,'-.');       % 绘制曲线
grid minor;                             % 添加次要网格
title('不同曲率的二次函数');
xlabel('x');
```

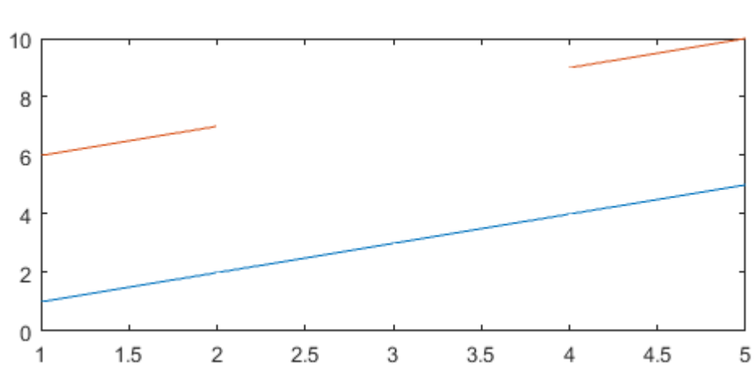

Chapter 12: Graphics: 2D Line Plots

Parameter	Details
X	x-values
Y	y-values
LineSpec	Line style, marker symbol, and color, specified as a string
Name,Value	Optional pairs of name-value arguments to customize line properties
h	handle to line graphics object

Section 12.1: Split line with NaNs

Interleave your y or x values with NaNs

```
x = [1:5; 6:10]';

x(3,2) = NaN
x =
     1     6
     2     7
     3   NaN
     4     9
     5    10

plot(x)
```



Section 12.2: Multiple lines in a single plot

In this example we are going to plot multiple lines onto a single axis. Additionally, we choose a different appearance for the lines and create a legend.

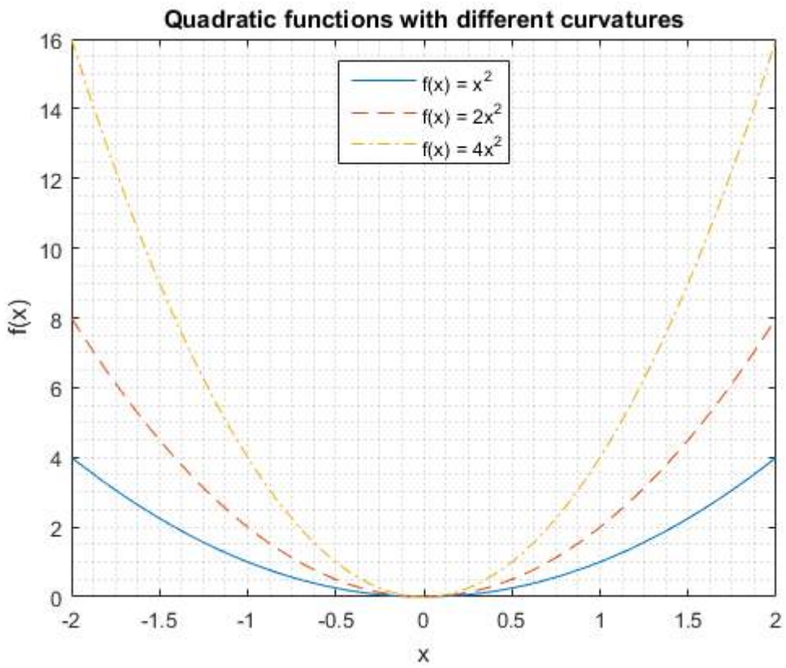
```
% create sample data
x = linspace(-2,2,100);                 % 100 linearly spaced points from -2 to 2
y1 = x.^2;
y2 = 2*x.^2;
y3 = 4*x.^2;

% create plot
figure;                                 % open new figure
plot(x,y1, x,y2,'--', x,y3,'-.');       % plot lines
grid minor;                             % add minor grid
title('Quadratic functions with different curvatures');
xlabel('x');
```

```
ylabel('f(x)');
legend('f(x) = x^2', 'f(x) = 2x^2', 'f(x) = 4x^2', 'Location','North');
```

在上面的例子中，我们使用单个plot命令绘制了多条线。另一种方法是为每条线使用单独的命令。在添加第二条线之前，我们需要使用hold on来保持图形内容。否则，之前绘制的线条将从图形中消失。要创建与上述相同的图形，我们可以使用以下命令：

```
figure; hold on;
plot(x,y1);
plot(x,y2,'--');
plot(x,y3,'-.');
```

两种情况下生成的图形如下所示：



第12.3节：自定义颜色和线型顺序

在MATLAB中，我们可以设置新的default自定义顺序，例如颜色顺序和线型顺序。这意味着新的顺序将应用于这些设置应用后创建的任何图形。新设置将一直保持，直到MATLAB会话关闭或进行新的设置。

默认颜色和线型顺序

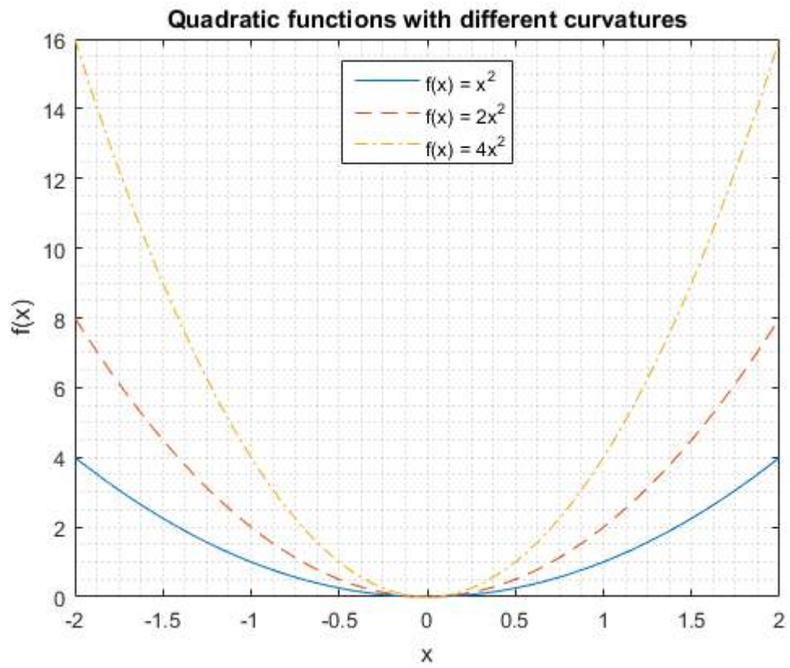
默认情况下，MATLAB使用几种不同的颜色且仅使用实线样式。因此，如果调用plot绘制多条线，MATLAB会通过颜色顺序交替绘制不同颜色的线条。


```
ylabel('f(x)');
legend('f(x) = x^2', 'f(x) = 2x^2', 'f(x) = 4x^2', 'Location','North');
```

In the above example, we plotted the lines with a single `plot`-command. An alternative is to use separate commands for each line. We need to *hold* the contents of a figure with `hold on` the latest before we add the second line. Otherwise the previously plotted lines will disappear from the figure. To create the same plot as above, we can use these following commands:

```
figure; hold on;
plot(x,y1);
plot(x,y2,'--');
plot(x,y3,'-.');
```

The resulting figure looks like this in both cases:

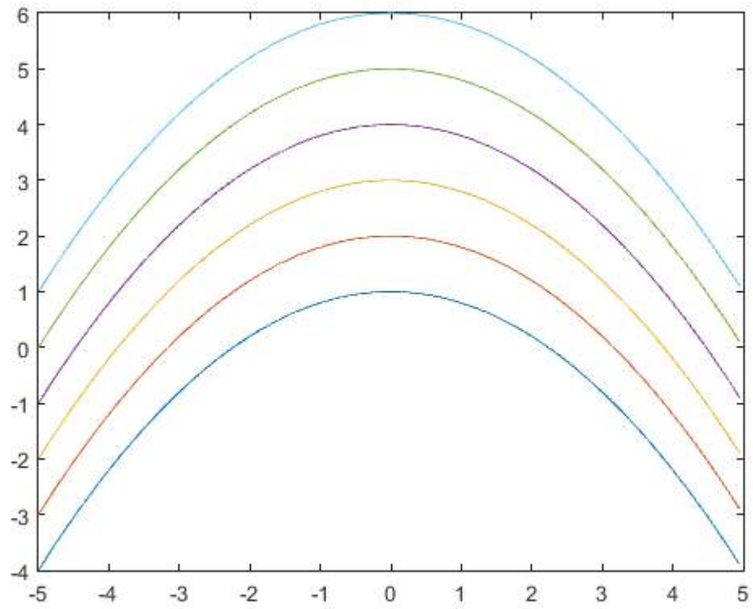


Section 12.3: Custom colour and line style orders

In MATLAB, we can set new *default* custom orders, such as a colour order and a line style order. That means new orders will be applied to any figure that is created after these settings have been applied. The new settings remains until MATLAB session is closed or new settings has been made.

Default colour and line style order

By default, MATLAB uses a couple of different colours and only a solid line style. Therefore, if `plot` is called to draw multiple lines, MATLAB alternates through a colour order to draw lines in different colours.



我们可以通过调用带有全局句柄0的get函数，然后使用此属性来获取默认颜色顺序 DefaultAxesColorOrder:

```
>> get(0, 'DefaultAxesColorOrder')
ans =
         0    0.4470    0.7410
    0.8500    0.3250    0.0980
    0.9290    0.6940    0.1250
    0.4940    0.1840    0.5560
    0.4660    0.6740    0.1880
    0.3010    0.7450    0.9330
    0.6350    0.0780    0.1840
```

自定义颜色和线型顺序

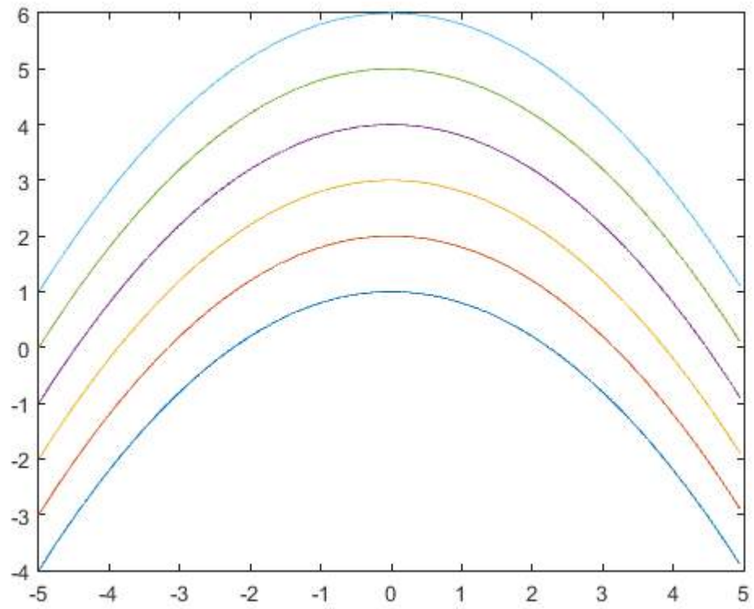
一旦我们决定设置自定义颜色顺序和线型顺序，MATLAB必须同时交替使用两者。MATLAB应用的第一个变化是颜色。当所有颜色用尽后，MATLAB会从定义的线型顺序中应用下一个线型，并将颜色索引重置为1。这意味着MATLAB将再次开始交替使用所有颜色，但使用顺序中的下一个线型。当所有线型和颜色都用尽时，显然MATLAB会从头开始循环，使用第一个颜色和第一个线型。

在这个例子中，我定义了一个输入向量和一个匿名函数，以便绘图更方便一些：

```
F = @(a,x) bsxfun(@plus, -0.2*x(:).^2, a);
x = (-5:5/100:5-5/100)';
```

要设置新的颜色或线型顺序，我们调用带有全局句柄0的set函数，后跟一个属性 DefaultAxesXXXXXXX; XXXXXXX可以是ColorOrder或LineStyleOrder。以下命令将颜色顺序分别设置为黑色、红色和蓝色：

```
set(0, 'DefaultAxesColorOrder', [0 0 0; 1 0 0; 0 0 1]);
```

We can obtain the default colour order by calling `get` with a global handle `0` followed by this attribute `DefaultAxesColorOrder`:

```
>> get(0, 'DefaultAxesColorOrder')
ans =
         0    0.4470    0.7410
    0.8500    0.3250    0.0980
    0.9290    0.6940    0.1250
    0.4940    0.1840    0.5560
    0.4660    0.6740    0.1880
    0.3010    0.7450    0.9330
    0.6350    0.0780    0.1840
```

Custom colour and line style order

Once we have decided to set a custom colour order AND line style order, MATLAB must alternate through both. The first change MATLAB applies is a colour. When all colours are exhausted, MATLAB applies the next line style from a defined line style order and set a colour index to 1. That means MATLAB will begin to alternate through all colours again but using the next line style in its order. When all line styles and colours are exhausted, obviously MATLAB begins to cycle from the beginning using the first colour and the first line style.

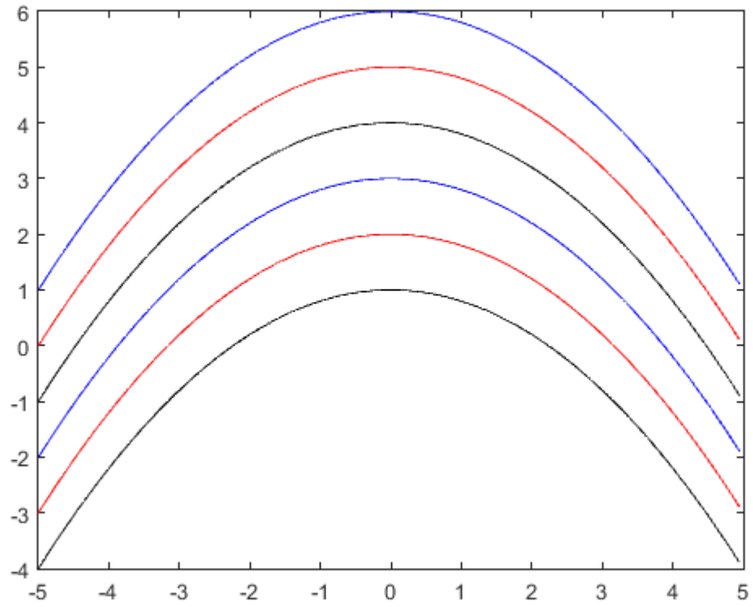
For this example, I have defined an input vector and an anonymous function to make plotting figures a little bit easier:

```
F = @(a,x) bsxfun(@plus, -0.2*x(:).^2, a);
x = (-5:5/100:5-5/100)';
```

To set a new colour or a new line style orders, we call `set` function with a global handle `0` followed by an attribute `DefaultAxesXXXXXXX`; `XXXXXXX` can either be `ColorOrder` or `LineStyleOrder`. The following command sets a new colour order to black, red and blue, respectively:

```
set(0, 'DefaultAxesColorOrder', [0 0 0; 1 0 0; 0 0 1]);
```

```
plot(x, F([1 2 3 4 5 6],x));
```



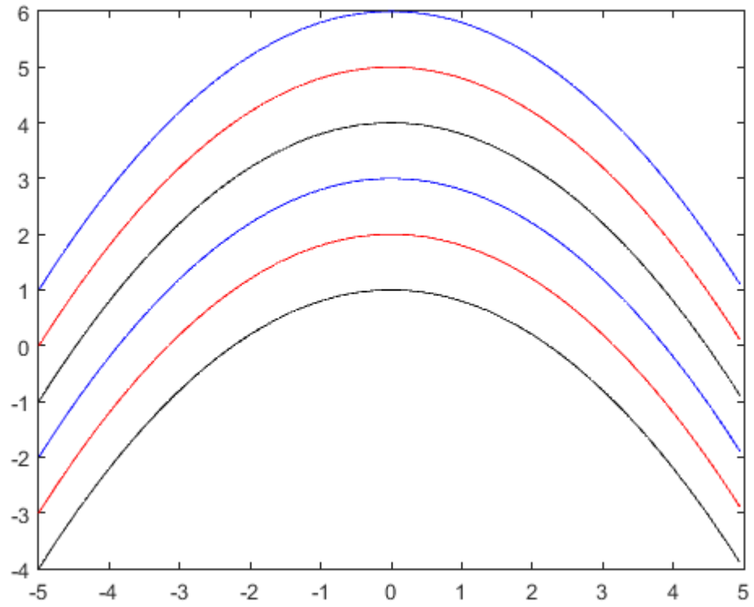
如您所见，MATLAB 仅通过颜色交替，因为线型顺序默认设置为实线。
当一组颜色用尽时，MATLAB 会从颜色顺序的第一个颜色开始。

下面的命令设置颜色和线型顺序：

```
set(0, 'DefaultAxesColorOrder', [0 0 0; 1 0 0; 0 0 1]);
set(0, 'DefaultAxesLineStyleOrder', {'-' '--'});
plot(x, F([1 2 3 4 5 6],x));
```



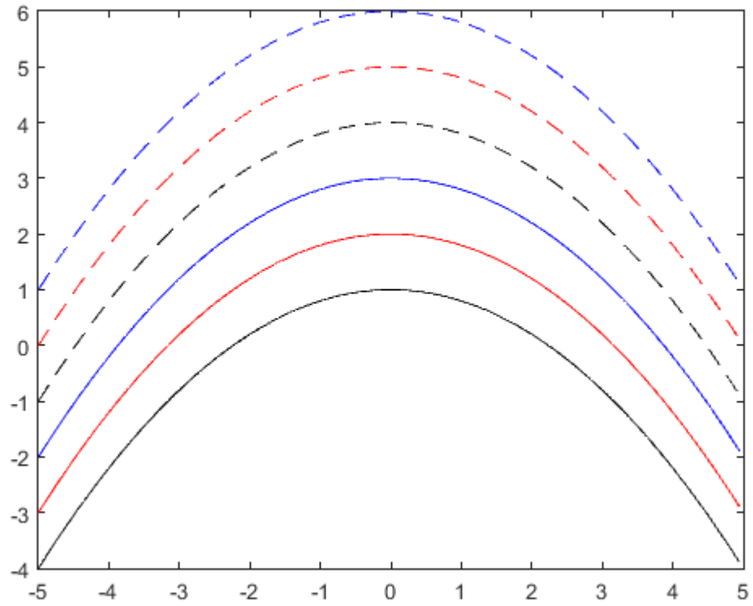
```
plot(x, F([1 2 3 4 5 6],x));
```



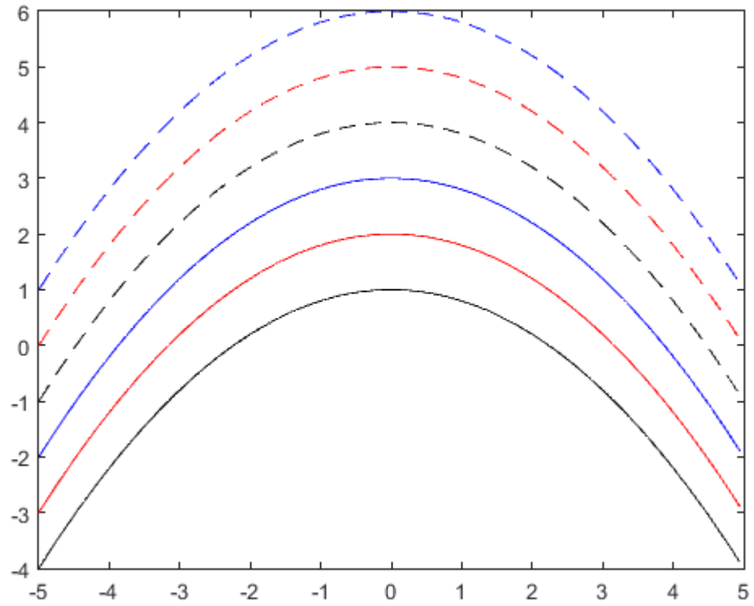
As you can see, MATLAB alternates only through colours because line style order is set to a solid line by default.
When a set of colours is exhausted, MATLAB starts from the first colour in the colour order.

The following commands set both colour and line style orders:

```
set(0, 'DefaultAxesColorOrder', [0 0 0; 1 0 0; 0 0 1]);
set(0, 'DefaultAxesLineStyleOrder', {'-' '--'});
plot(x, F([1 2 3 4 5 6],x));
```



现在，MATLAB 使用颜色作为最频繁的属性，在不同颜色和不同线型之间交替。



Now, MATLAB alternates through different colours and different line styles using colour as most frequent attribute.

第13章：图形：二维和三维变换

第13.1节：二维变换

在本例中，我们将绘制一个方形线条，使用line命令，并对其进行变换。
然后我们将使用相同的变换，但顺序不同，看看它如何影响结果。

首先我们打开一个图形并设置一些初始参数（正方形点坐标和变换参数）

```
%打开图形并创建坐标轴
Figureh=figure('NumberTitle','off','Name','变换示例',...
     'Position',[200 200 700 700]); %背景设置为红色，这样我们知道只能看到坐标轴
Axesh=axes('XLim',[-8 8],'YLim',[-8,8]);

 %初始化变量
square=[-0.5 -0.5;-0.5 0.5;0.5 0.5;0.5 -0.5]; %由其顶点表示
    Sx=0.5;
Sy=2;
     Tx=2;
     Ty=2;
     teta=pi/4;
```

接下来我们构造变换矩阵（缩放、旋转和平移）：

```
%生成变换矩阵
S=makehgtform('scale',[Sx Sy 1]);
R=makehgtform('zrotate',teta);
T=makehgtform('translate',[Tx Ty 0]);
```

接下来我们绘制蓝色正方形：

```
%% 绘制原始蓝色正方形
OriginalSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','b','LineWidth',3);
grid on;     % 在图形上应用网格
hold all;   % 保持当前坐标轴，允许后续图形叠加
```

接下来我们将用不同颜色（红色）再次绘制，并应用变换：

```
%% 绘制红色正方形
% 计算矩形顶点
HrectTRS=T*R*S;
RedSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','r','LineWidth',3);
% 坐标轴的变换
AxesTransformation=hgtransform('Parent',gca,'matrix',HrectTRS);
% 设置线条为变换后坐标轴的子对象
set(RedSQ,'Parent',AxesTransformation);
```

结果应如下所示：

Chapter 13: Graphics: 2D and 3D Transformations

Section 13.1: 2D Transformations

In this Example we are going to take a square shaped line plotted using `line` and perform transformations on it. Then we are going to use the same transformations but in different order and see how it influences the results.

First we open a figure and set some initial parameters (square point coordinates and transformation parameters)

```
%Open figure and create axis
Figureh=figure('NumberTitle','off','Name','Transformation Example',...
    'Position',[200 200 700 700]); %bg is set to red so we know that we can only see the axes
Axesh=axes('XLim',[-8 8],'YLim',[-8,8]);

 %Initializing Variables
    square=[-0.5 -0.5;-0.5 0.5;0.5 0.5;0.5 -0.5]; %represented by its vertices
    Sx=0.5;
    Sy=2;
    Tx=2;
    Ty=2;
    teta=pi/4;
```

Next we construct the transformation matrices (scale, rotation and translation):

```
%Generate Transformation Matrix
S=makehgtform('scale',[Sx Sy 1]);
R=makehgtform('zrotate',teta);
T=makehgtform('translate',[Tx Ty 0]);
```

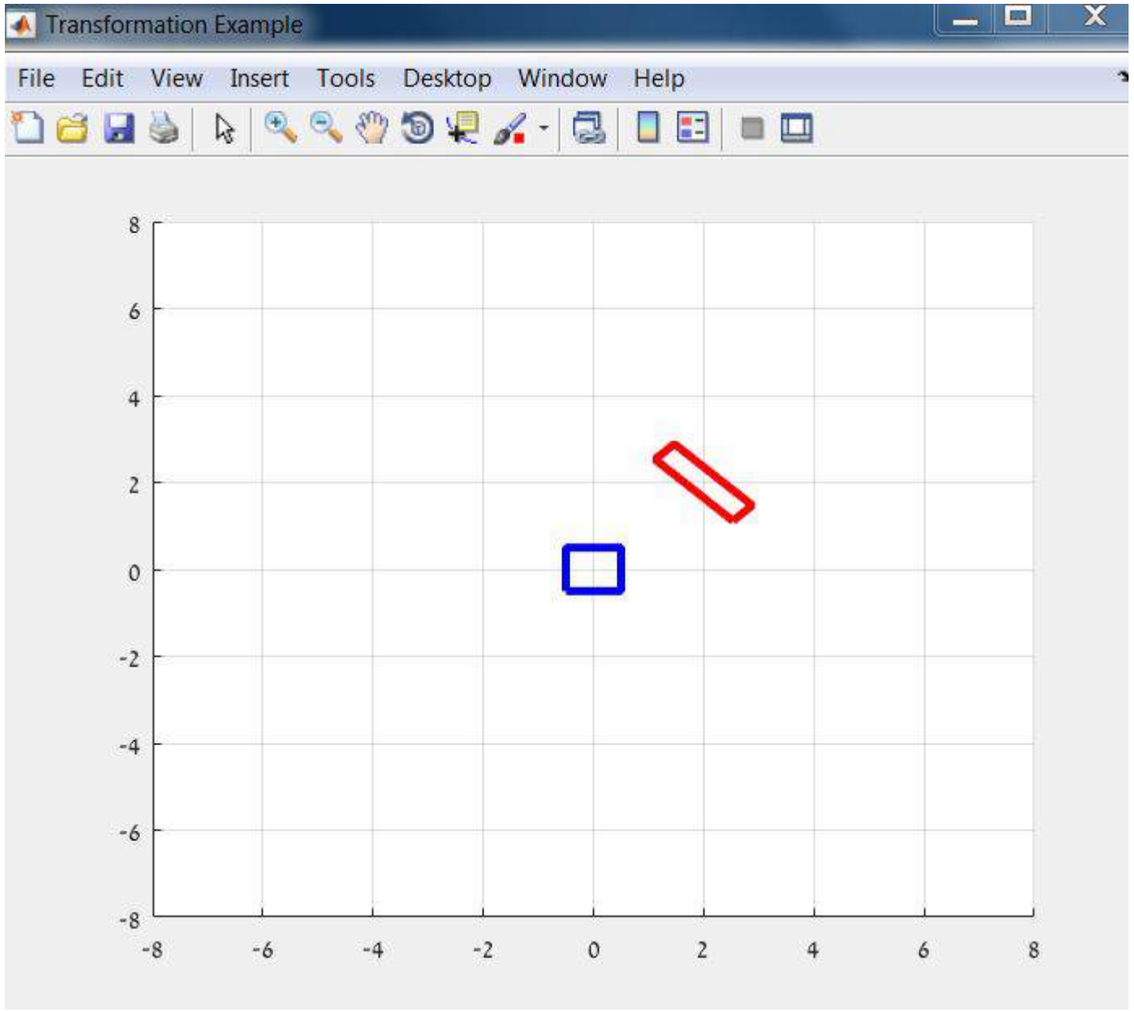
Next we plot the blue square:

```
%% Plotting the original Blue Square
OriginalSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','b','LineWidth',3);
grid on;    % Applying grid on the figure
hold all;   % Holding all Following  graphs to current axes
```

Next we will plot it again in a different color (red) and apply the transformations:

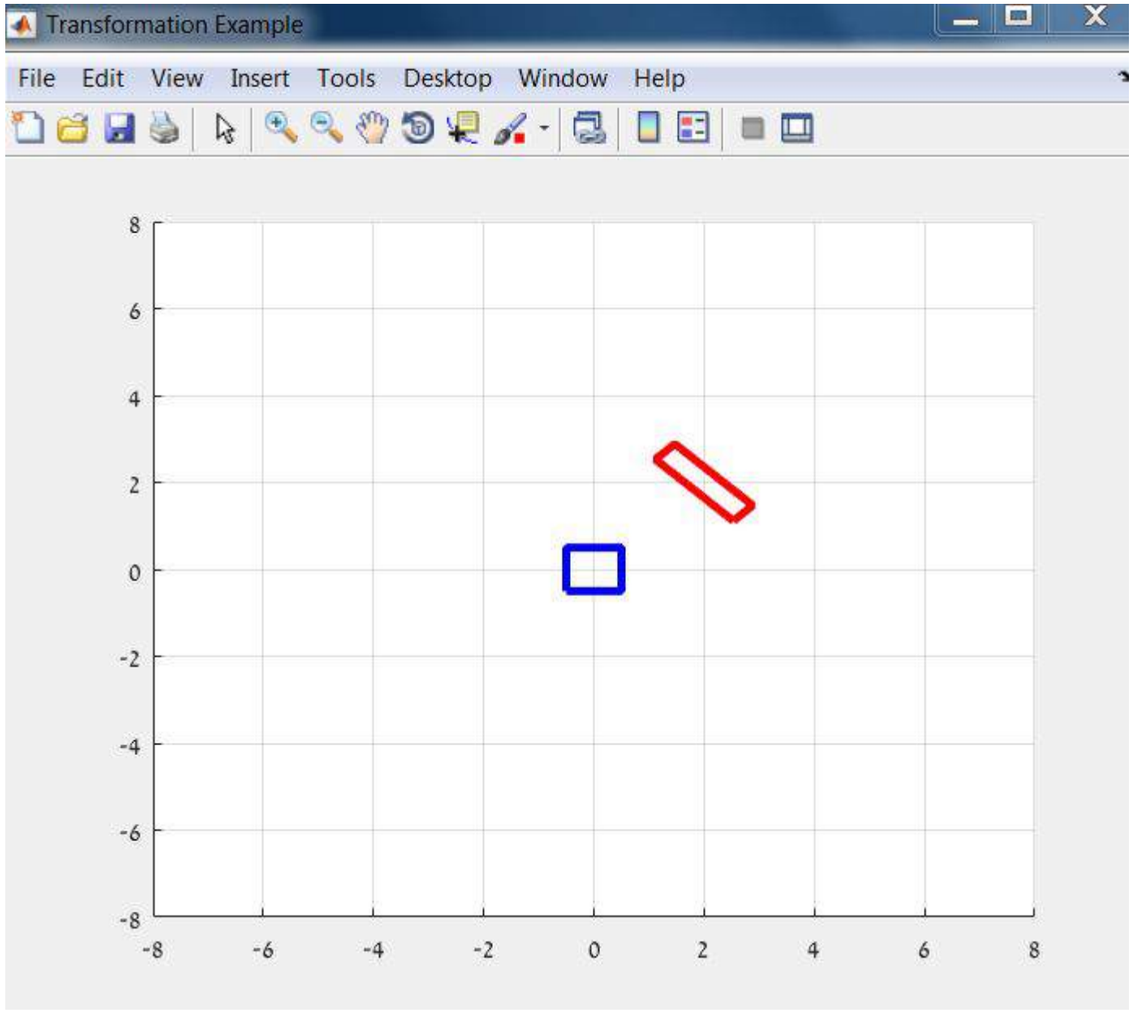
```
%% Plotting the Red Square
%Calculate rectangle vertices
HrectTRS=T*R*S;
RedSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','r','LineWidth',3);
%transformation of the axes
AxesTransformation=hgtransform('Parent',gca,'matrix',HrectTRS);
%setting the line to be a child of transformed axes
set(RedSQ,'Parent',AxesTransformation);
```

The result should look like this:



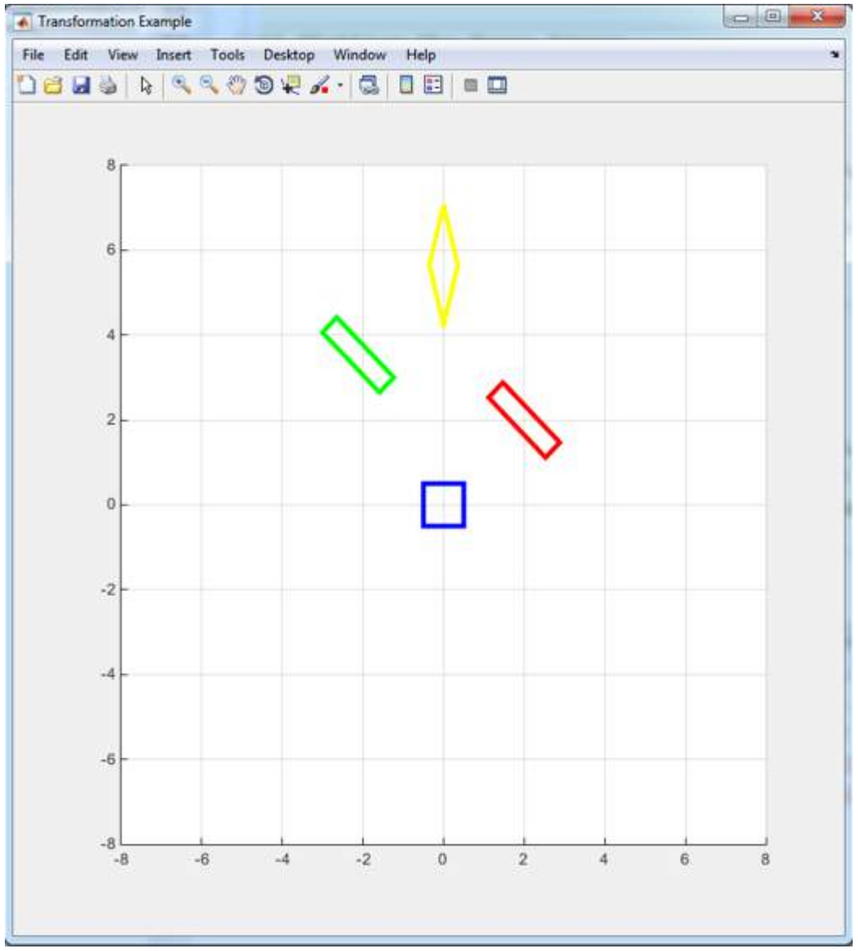
现在让我们看看改变变换顺序时会发生什么：

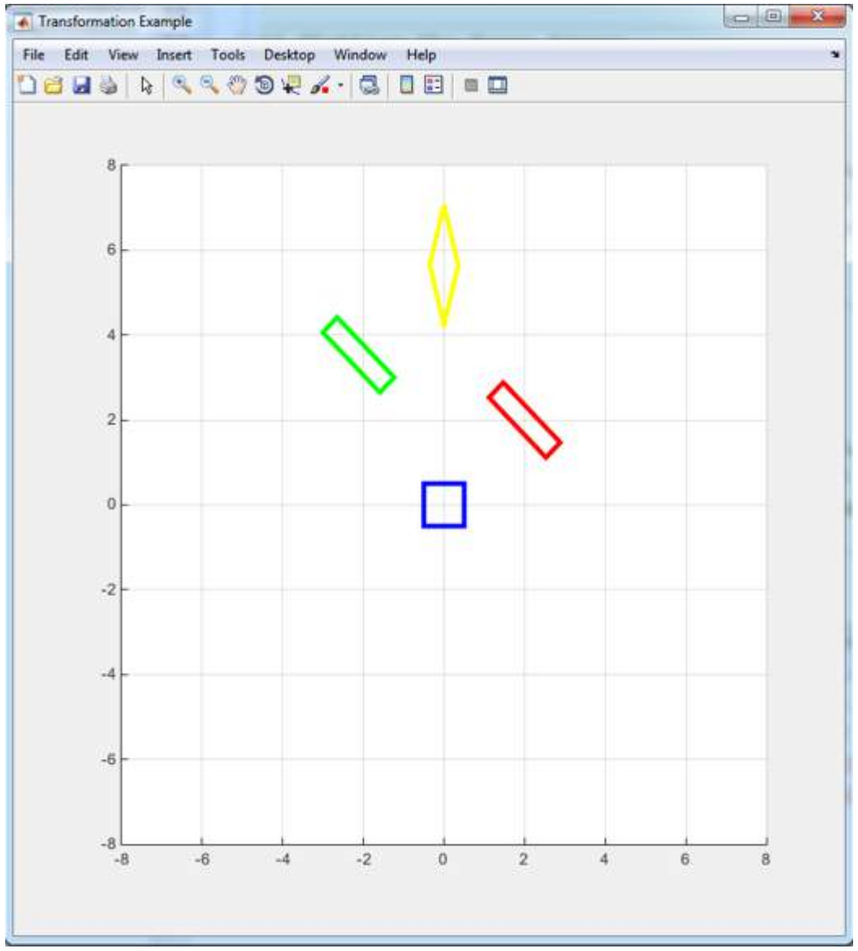
```
%% 绘制绿色正方形
HrectRST=R*S*T;
GreenSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','g','LineWidth',3);
AxesTransformation=hgtransform('Parent',gca,'matrix',HrectRST);
set(GreenSQ,'Parent',AxesTransformation);

%% 绘制黄色正方形
HrectSRT=S*R*T;
YellowSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','y','LineWidth',3);
AxesTransformation=hgtransform('Parent',gca,'matrix',HrectSRT);
set(YellowSQ,'Parent',AxesTransformation);
```

Now let's see what happens when we change the transformation order:

```
%% Plotting the Green Square
HrectRST=R*S*T;
GreenSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','g','LineWidth',3);
AxesTransformation=hgtransform('Parent',gca,'matrix',HrectRST);
set(GreenSQ,'Parent',AxesTransformation);

%% Plotting the Yellow Square
HrectSRT=S*R*T;
YellowSQ=line([square(:,1);square(1,1)],[square(:,2);square(1,2)],'Color','y','LineWidth',3);
AxesTransformation=hgtransform('Parent',gca,'matrix',HrectSRT);
set(YellowSQ,'Parent',AxesTransformation);
```


Transformation Example
File Edit View Insert Tools Desktop Window Help
8
6
4
2
0
-2
-4
-6
-8
-8 -6 -4 -2 0 2 4 6 8


Transformation Example
File Edit View Insert Tools Desktop Window Help
8
6
4
2
0
-2
-4
-6
-8
-8 -6 -4 -2 0 2 4 6 8

第14章：在MATLAB中控制子图着色

由于我不止一次为此苦恼，而且网上关于该如何操作的说明并不清晰，我决定整理现有资料，加入一些自己的内容，以解释如何创建带有单个颜色条且根据该颜色条进行缩放的子图。

我已经用最新的MATLAB测试过，但我很确定它在旧版本中也能运行。

第14.1节：如何实现

这是一个简单的代码，创建了6个三维子图，最后同步它们显示的颜色。

```
c_fin = [0,0];
[X,Y] = meshgrid(1:0.1:10,1:0.1:10);

figure; hold on;
for i = 1 : 6
Z(:,:,i) = i * (sin(X) + cos(Y));

    ax(i) = subplot(3,2,i); hold on; grid on;
    surf(X, Y, Z(:,:,i));
    view(-26,30);
    colormap('jet');
ca = caxis;
c_fin = [min(c_fin(1),ca(1)), max(c_fin(2),ca(2))];
end

%%你可以在这里停止，看看效果如何，然后再进行颜色操作

c = colorbar('eastoutside');
c.Label.String = '单位';
set(c, '位置', [0.9, 0.11, 0.03, 0.815]); %%你可能想调整这些数值
pause(2); %%需要暂停以允许最后一张图像在改变坐标轴前调整大小
for i = 1 : 6
pos=get(ax(i), '位置');
     axes(ax(i));
    set(ax(i), '位置', [pos(1) pos(2) 0.85*pos(3) pos(4)]);
    set(ax(i),'颜色限制', c_fin); %%这里是关键操作
end
```


Chapter 14: Controlling Subplot coloring in MATLAB

As I was struggling with this more than once, and the web isn't really clear on what to do, I decided to take what's out there, adding some of my own in order to explain how to create subplots which have one colorbar and they are scaled according to it.

I have tested this using latest MATLAB but I'm pretty sure it'll work in older versions.

Section 14.1: How it's done

This is a simple code creating 6 3d-subplots and in the end syncing the color displayed in each of them.

```
c_fin = [0,0];
[X,Y] = meshgrid(1:0.1:10,1:0.1:10);

figure; hold on;
for i = 1 : 6
    Z(:,:,i) = i * (sin(X) + cos(Y));

    ax(i) = subplot(3,2,i); hold on; grid on;
    surf(X, Y, Z(:,:,i));
    view(-26,30);
    colormap('jet');
    ca = caxis;
    c_fin = [min(c_fin(1),ca(1)), max(c_fin(2),ca(2))];
end

%%you can stop here to see how it looks before we color-manipulate

c = colorbar('eastoutside');
c.Label.String = 'Units';
set(c, 'Position', [0.9, 0.11, 0.03, 0.815]); %%you may want to play with these values
pause(2); %%need this to allow the last image to resize itself before changing its axes
for i = 1 : 6
    pos=get(ax(i), 'Position');
    axes(ax(i));
    set(ax(i), 'Position', [pos(1) pos(2) 0.85*pos(3) pos(4)]);
    set(ax(i),'Clim', c_fin); %%this is where the magic happens
end
```

第15章：图像处理

第15.1节：基本图像输入输出

```
>> img = imread('football.jpg');
```

使用 imread 将图像文件读入 MATLAB 中的矩阵。
一旦你使用 imread 读取图像，它会以多维数组（ND-array）的形式存储在内存中：

```
>> 大小(图像)
答案 =
 256   320     3
```

图像'football.jpg'有256行320列，且有3个颜色通道：红色、绿色和蓝色。

你现在可以对它进行镜像处理：

```
>> mirrored = img(:, end:-1:1, :);  %// 类似于在MATLAB中镜像任意多维数组
```

最后，使用imwrite将其写回为图像：

```
>> imwrite(mirrored, 'mirrored_football.jpg');
```

第15.2节：从互联网获取图像

只要你有网络连接，就可以从超链接读取图像

```
I=imread('https://cdn.sstatic.net/Sites/stackoverflow/company/img/logos/so/so-logo.png');
```

第15.3节：使用二维快速傅里叶变换进行滤波

像处理一维信号一样，可以通过应用傅里叶变换，在频域中与滤波器相乘，然后再变换回空间域来滤波图像。以下是在 MATLAB 中如何对图像应用高通或低通滤波器的方法：

设image为原始未滤波的图像，下面是计算其二维傅里叶变换（FFT）的方法：

```
ft = fftshift(fft2(image));
```

现在要排除频谱的一部分，需要将其像素值设为0。原始图像中包含的空间频率在使用fftshift后从中心映射到边缘。为了排除低频，我们将把中心的圆形区域设为0。

以下是如何使用内置函数生成半径为D的圆盘形二值掩码：

```
[x y ~] = size(ft);
D = 20;
mask = fspecial('disk', D) == 0;
mask = imresize(padarray(mask, [floor((x/2)-D) floor((y/2)-D)], 1, 'both'), [x y]);
```

通过将傅里叶变换结果与上述获得的二值掩码逐点相乘，可以对频域图像进行掩码处理：

Chapter 15: Image processing

Section 15.1: Basic image I/O

```
>> img = imread('football.jpg');
```

Use imread to read image files into a matrix in MATLAB.
Once you imread an image, it is stored as an ND-array in memory:

```
>> size(img)
ans =
 256   320     3
```

The image 'football.jpg' has 256 rows and 320 columns and it has 3 color channels: Red, Green and Blue.

You can now mirror it:

```
>> mirrored = img(:, end:-1:1, :);  %// like mirroring any ND-array in MATLAB
```

And finally, write it back as an image using imwrite:

```
>> imwrite(mirrored, 'mirrored_football.jpg');
```

Section 15.2: Retrieve Images from the Internet

As long as you have an internet connection, you can read images from an hyperlink

```
I=imread('https://cdn.sstatic.net/Sites/stackoverflow/company/img/logos/so/so-logo.png');
```

Section 15.3: Filtering Using a 2D FFT

Like for 1D signals, it's possible to filter images by applying a Fourier transformation, multiplying with a filter in the frequency domain, and transforming back into the space domain. Here is how you can apply high- or low-pass filters to an image with MATLAB:

Let image be the original, unfiltered image, here's how to compute its 2D FFT:

```
ft = fftshift(fft2(image));
```

Now to exclude a part of the spectrum, one need to set its pixel values to 0. The spatial frequency contained in the original image is mapped from the center to the edges (after using fftshift). To exclude the low frequencies, we will set the central circular area to 0.

Here's how to generate a disc-shaped binary mask with radius D using built-in function:

```
[x y ~] = size(ft);
D = 20;
mask = fspecial('disk', D) == 0;
mask = imresize(padarray(mask, [floor((x/2)-D) floor((y/2)-D)], 1, 'both'), [x y]);
```

Masking the frequency domain image can be done by multiplying the FFT point-wise with the binary mask obtained above:

```
masked_ft = ft .* mask;
```

现在，让我们计算逆FFT：

```
filtered_image = ifft2(ifftshift(masked_ft), 'symmetric');
```

图像中的高频部分是锐利的边缘，因此该高通滤波器可用于锐化图像。

第15.4节：图像滤波

假设你有一张图像 rgbImg，例如，使用 imread 读取的。

```
>> rgbImg = imread('pears.png');
>> figure, imshow(rgbImg), title('原始图像')
```

使用 fspecial 创建二维滤波器：

```
>> h = fspecial('disk', 7);
>> figure, imshow(h, []), title('滤波器')
```



```
masked_ft = ft .* mask;
```

Now, let's compute the inverse FFT:

```
filtered_image = ifft2(ifftshift(masked_ft), 'symmetric');
```

The high frequencies in an image are the sharp edges, so this high-pass filter can be used to sharpen images.

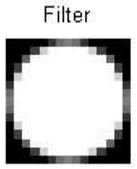
Section 15.4: Image Filtering

Let's say you have an image `rgbImg`, e.g., read in using `imread`.

```
>> rgbImg = imread('pears.png');
>> figure, imshow(rgbImg), title('Original Image')
```

Use `fspecial` to create a 2D filter:

```
>> h = fspecial('disk', 7);
>> figure, imshow(h, []), title('Filter')
```



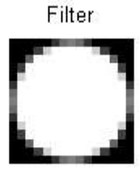
使用 imfilter 对图像应用滤波器：

```
>> filteredRgbImg = imfilter(rgbImg, h);
>> figure, imshow(filteredRgbImg), title('滤波后图像')
```

第15.5节：测量连通区域的属性

从一个二值图像开始，bwImg，图像中包含多个连通物体。

```
>> bwImg = imread('blobs.png');
>> figure, imshow(bwImg), title('二值图像')
```

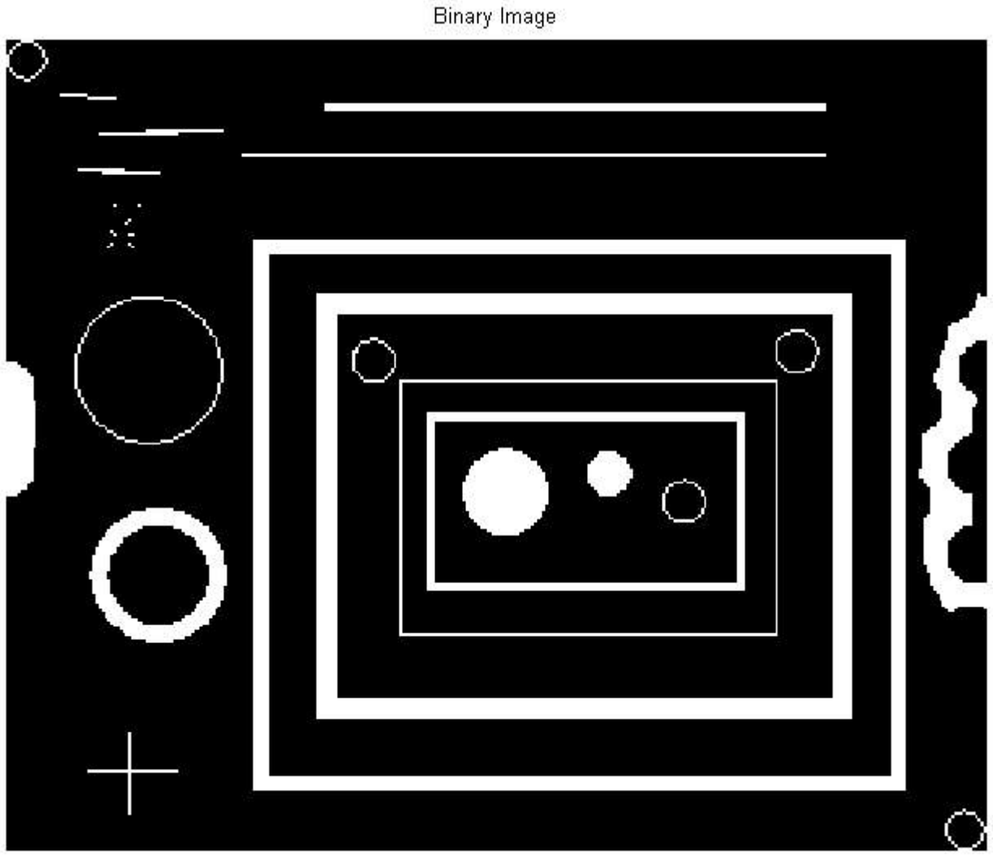
Use imfilter to apply the filter on the image:

```
>> filteredRgbImg = imfilter(rgbImg, h);
>> figure, imshow(filteredRgbImg), title('Filtered Image')
```

Section 15.5: Measuring Properties of Connected Regions

Starting with a binary image, `bwImg`, which contains a number of connected objects.

```
>> bwImg = imread('blobs.png');
>> figure, imshow(bwImg), title('Binary Image')
```



要测量图像中每个物体的属性（例如面积、质心等），使用regionprops：

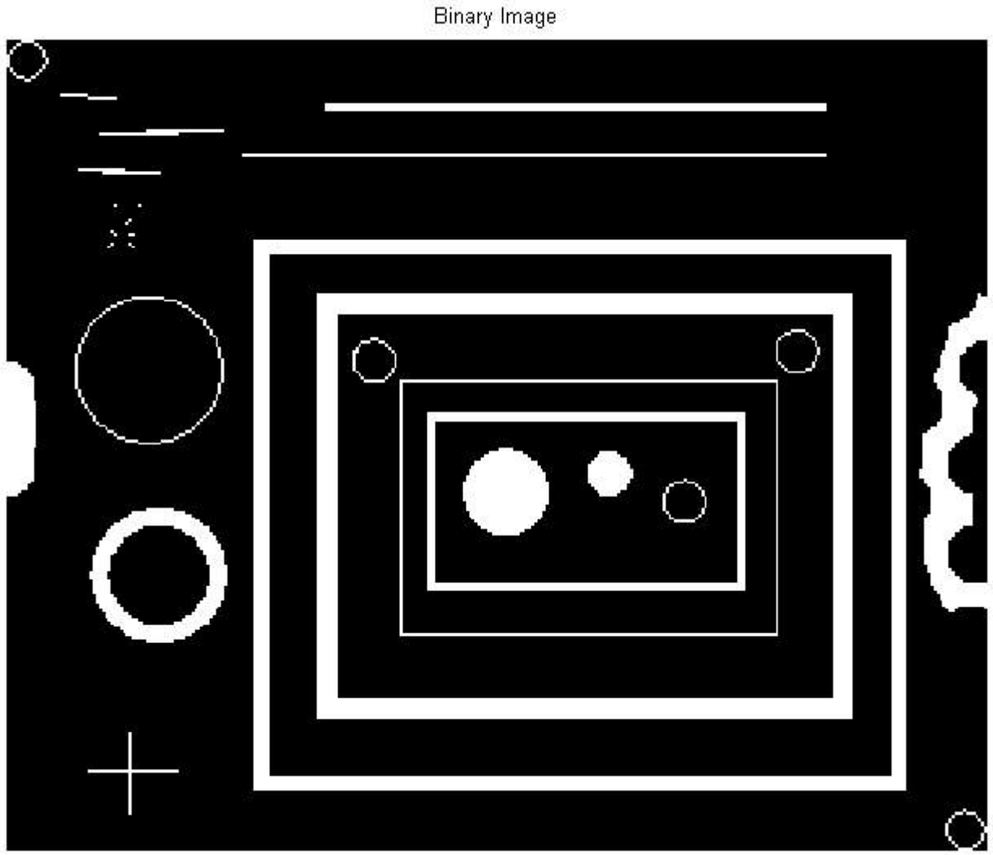
```
>> stats = regionprops(bwImg, 'Area', 'Centroid');
```

stats是一个结构体数组，包含图像中每个物体的结构体。访问某个物体的测量属性很简单。例如，要显示第一个物体的面积，只需：

```
>> stats(1).Area

ans =

    35
```

通过将质心叠加在原始图像上来可视化物体的质心。

```
>> figure, imshow(bwImg), title('叠加质心的二值图像')
>> hold on
>> for i = 1:size(stats)
scatter(stats(i).Centroid(1), stats(i).Centroid(2), 'filled');
end
```

To measure properties (e.g., area, centroid, etc) of every object in the image, use `regionprops`:

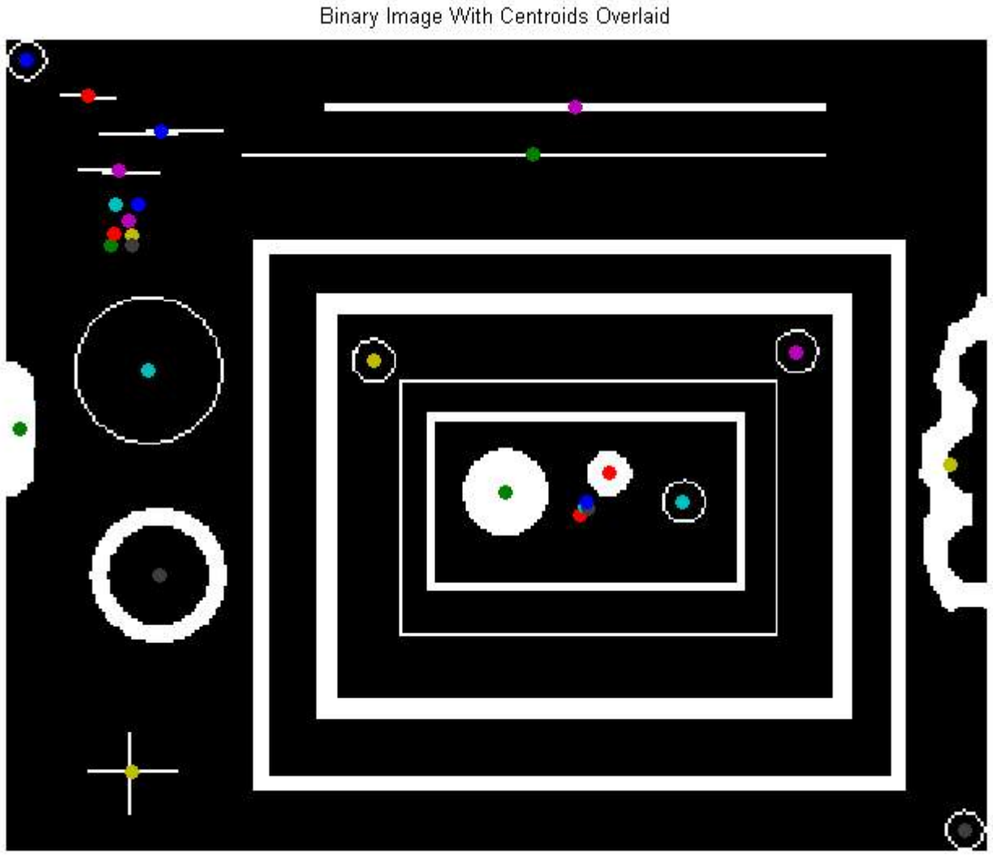
```
>> stats = regionprops(bwImg, 'Area', 'Centroid');
```

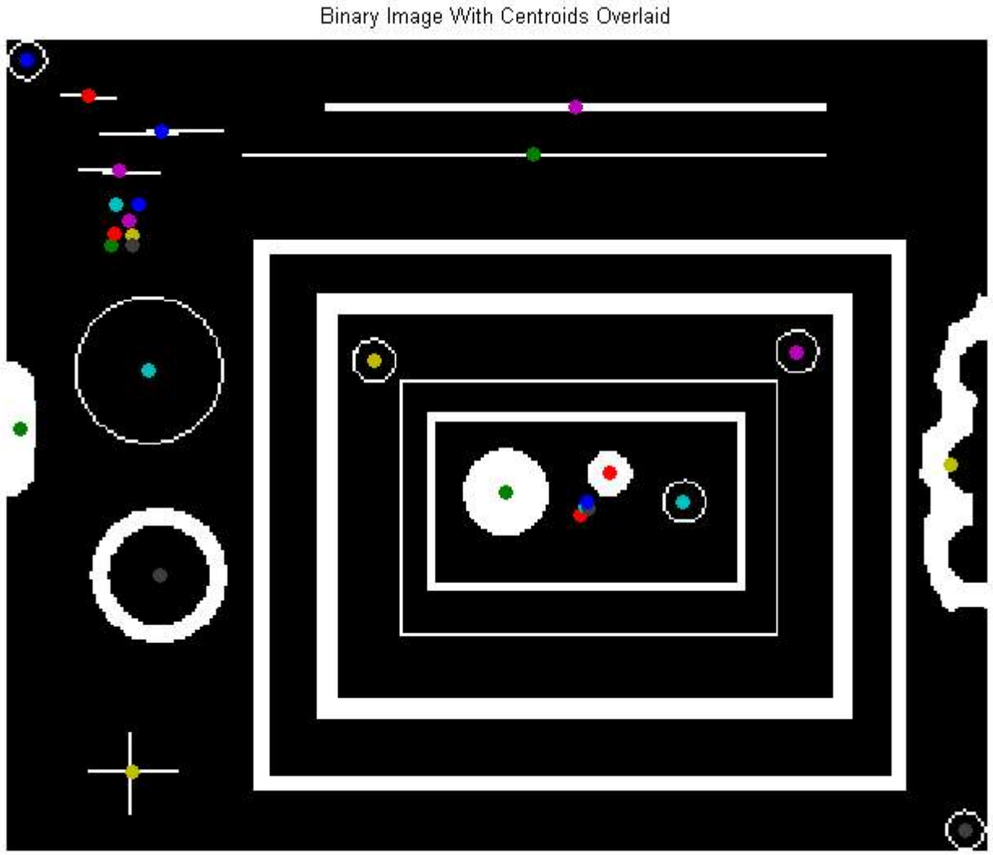
`stats` is a struct array which contains a struct for every object in the image. Accessing a measured property of an object is simple. For example, to display the area of the first object, simply,

```
>> stats(1).Area

ans =

    35
```

Visualize the object centroids by overlaying them on the original image.

```
>> figure, imshow(bwImg), title('Binary Image With Centroids Overlaid')
>> hold on
>> for i = 1:size(stats)
scatter(stats(i).Centroid(1), stats(i).Centroid(2), 'filled');
end
```


Binary Image With Centroids Overlaid


Binary Image With Centroids Overlaid

第16章：绘图

第16.1节：圆形

绘制圆形最简单的选项，显然是rectangle函数。

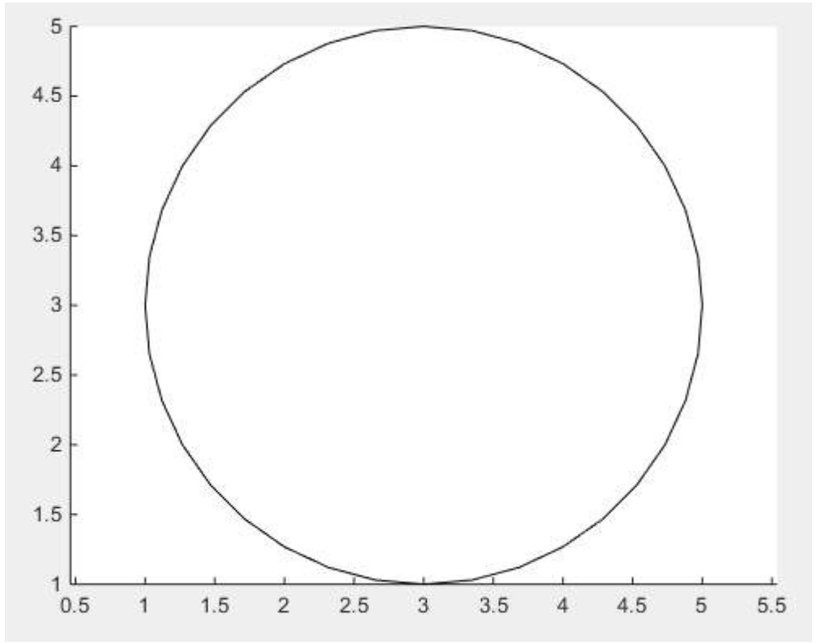
```
%// 半径
r = 2;

%// 中心
c = [3 3];

pos = [c-r 2*r 2*r];
rectangle('位置',pos,'曲率',[1 1])
axis equal
```

但矩形的曲率必须设置为1！

位置向量定义了矩形，前两个值 `x` 和 `y` 是矩形的左下角。
最后两个值定义矩形的宽度和高度。

```
pos = [ [x y] 宽度 高度 ]
```

圆的左下*角*——是的，这个圆有角，虽然是想象中的角——是**中心** `c = [3 3]` **减去半径** `r = 2`，**即[x y] = [1 1]。宽度和高度等于圆的直径，所以** `宽度 = 2*r; 高度 = 宽度;`



如果上述解法的平滑度不够，就只能使用显而易见的方法，通过三角函数绘制一个真正的圆。

```
%// 点的数量
n = 1000;
```


Chapter 16: Drawing

Section 16.1: Circles

The easiest option to draw a circle, is - obviously - the **`rectangle`** function.

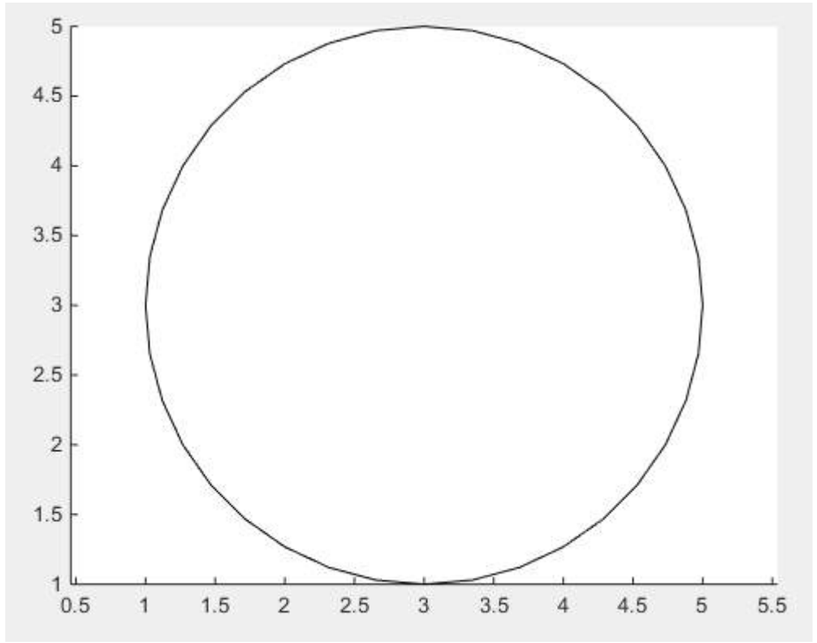
```
%// radius
r = 2;

%// center
c = [3 3];

pos = [c-r 2*r 2*r];
rectangle('Position',pos,'Curvature',[1 1])
axis equal
```

but the curvature of the rectangle has to be set to **1**!

The `position` vector defines the rectangle, the first two values `x` and `y` are the lower left corner of the rectangle. The last two values define width and height of the rectangle.

```
pos = [ [x y] width height ]
```

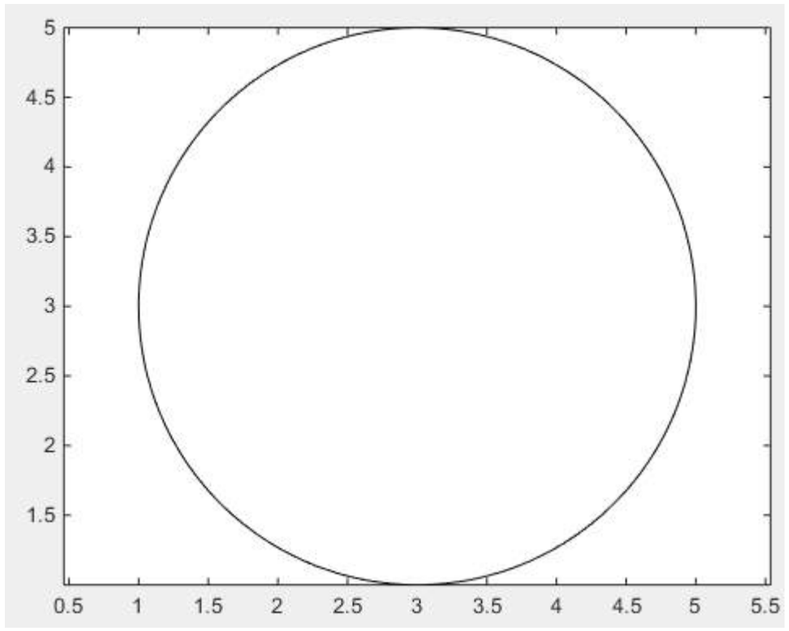
The lower left *corner* of the circle - yes, this circle has corners, imaginary ones though - is the **center** `c = [3 3]` **minus the radius** `r = 2` which is `[x y] = [1 1]`. **Width** and **height** are equal to the **diameter** of the circle, so `width = 2*r; height = width;`



In case the smoothness of the above solution is not sufficient, there is no way around using the obvious way of drawing an actual circle by use of **trigonometric functions**.

```
%// number of points
n = 1000;
```

```
%// 运行变量
t = linspace(0,2*pi,n);

x = c(1) + r*sin(t);
y = c(2) + r*cos(t);

%// 绘制线条
line(x,y)

%// 或者如果你想用颜色填充，可以绘制多边形
%// fill(x,y,[1,1,1])
axis equal
```



第16.2节：箭头

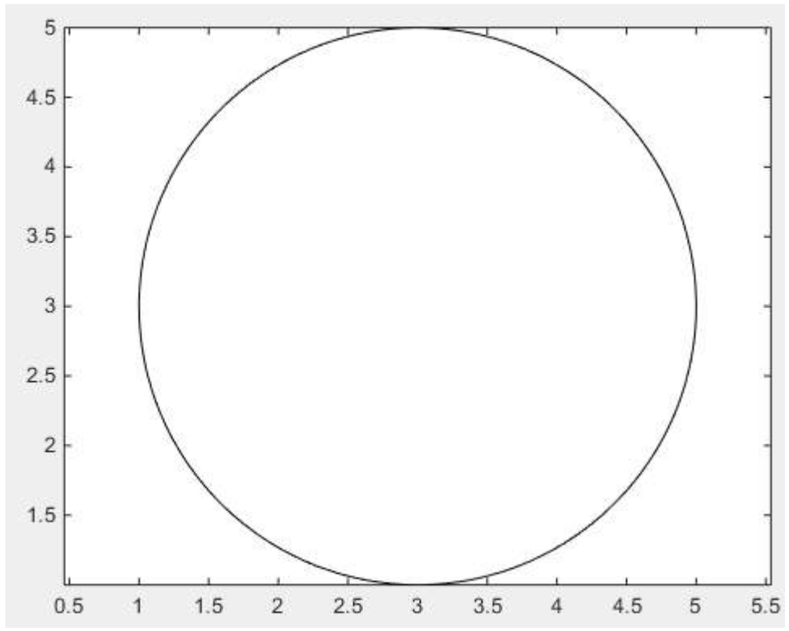
首先，可以使用quiver，这样就不必通过使用

annotation来处理麻烦的归一化图形单位

```
drawArrow = @(x,y) quiver( x(1),y(1),x(2)-x(1),y(2)-y(1),0 )

x1 = [10 30];
y1 = [10 30];

drawArrow(x1,y1); hold on

x2 = [25 15];
y2 = [15 25];

drawArrow(x2,y2)
```



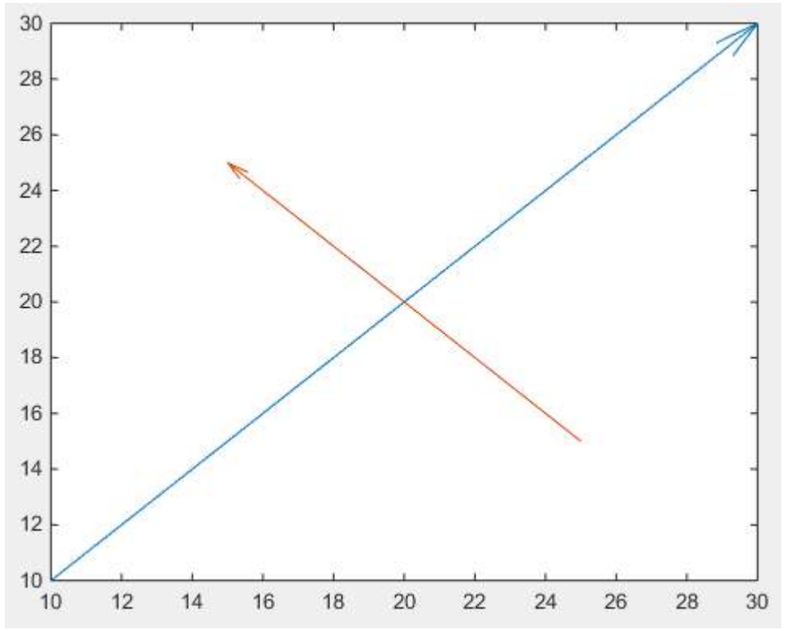
```
%// running variable
t = linspace(0,2*pi,n);

x = c(1) + r*sin(t);
y = c(2) + r*cos(t);

%// draw line
line(x,y)

%// or draw polygon if you want to fill it with color
%// fill(x,y,[1,1,1])
axis equal
```



Section 16.2: Arrows

Firstly, one can use quiver, where one doesn't have to deal with unhandy normalized figure units by use of annotation

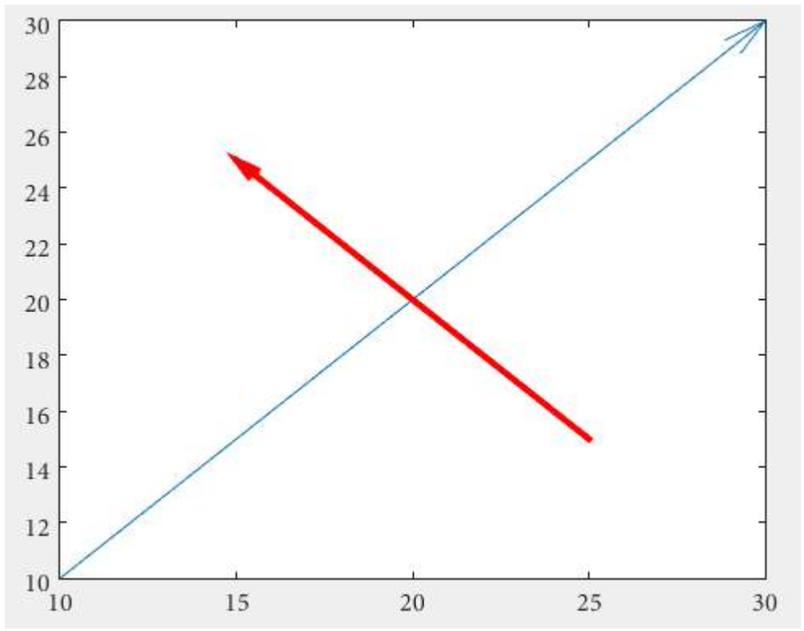
```
drawArrow = @(x,y) quiver( x(1),y(1),x(2)-x(1),y(2)-y(1),0 )

x1 = [10 30];
y1 = [10 30];

drawArrow(x1,y1); hold on

x2 = [25 15];
y2 = [15 25];

drawArrow(x2,y2)
```



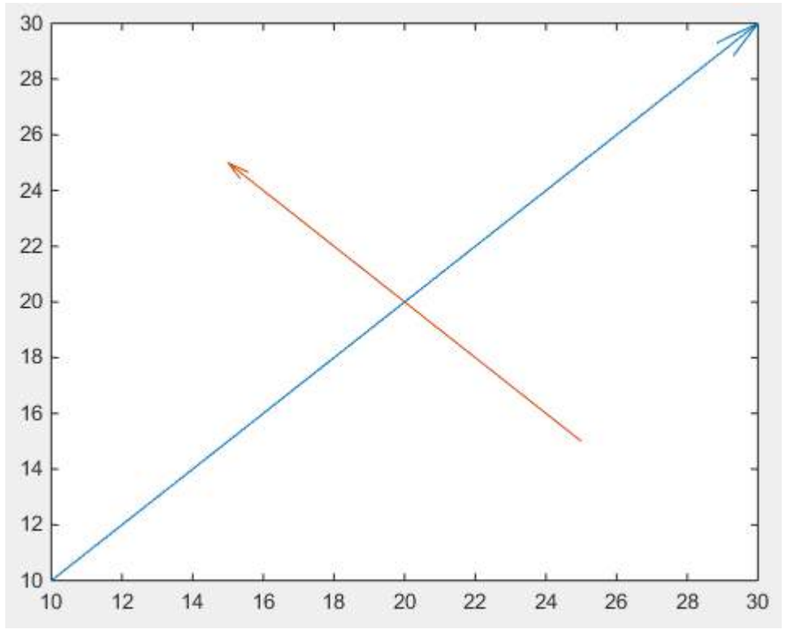
重要的是quiver的第5个参数：0，它禁用了默认的缩放，因为该函数通常用于绘制向量场。（或者使用属性值对'AutoScale','off'）

还可以添加其他功能：

```
drawArrow = @(x,y,varargin) quiver( x(1),y(1),x(2)-x(1),y(2)-y(1),0, varargin{:} )
drawArrow(x1,y1); hold on
drawArrow(x2,y2,'linewidth',3,'color','r')
```



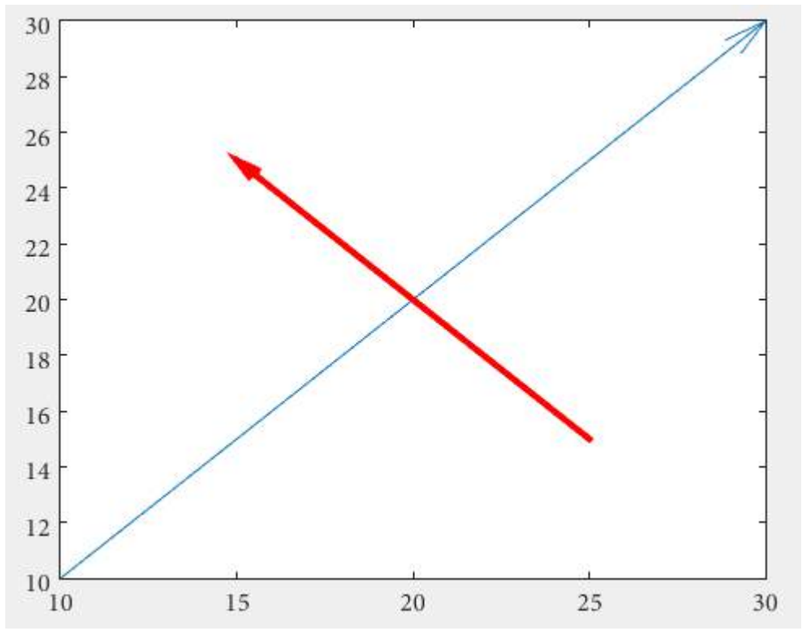
如果需要不同的箭头样式，则需要使用注释（这个答案可能有帮助如何更改quiver图中的箭头样式？）。



Important is the 5th argument of `quiver`: 0 which disables an otherwise default scaling, as this function is usually used to plot vector fields. (or use the property value pair `'AutoScale','off'`)

One can also add additional features:

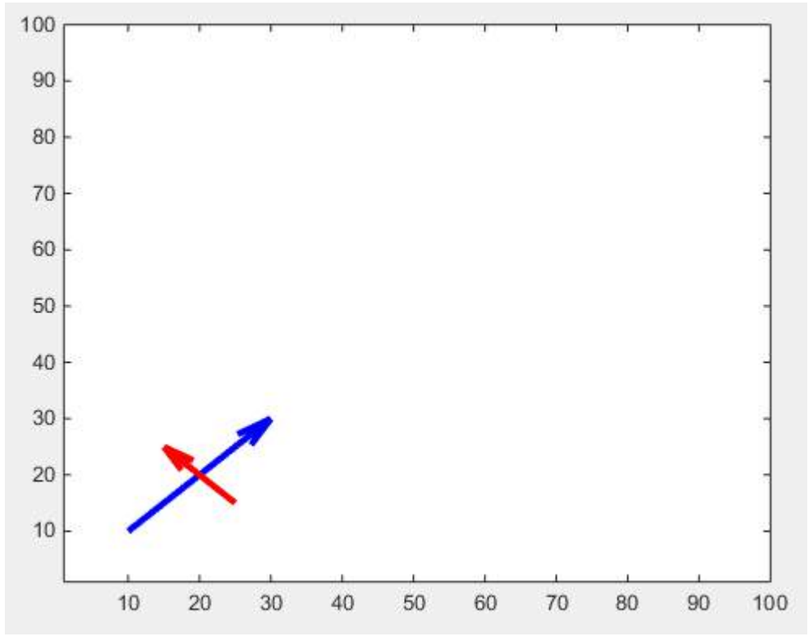
```
drawArrow = @(x,y,varargin) quiver( x(1),y(1),x(2)-x(1),y(2)-y(1),0, varargin{:} )
drawArrow(x1,y1); hold on
drawArrow(x2,y2,'linewidth',3,'color','r')
```



If different arrowheads are desired, one needs to use annotations (this answer is may helpful How do I change the arrow head style in quiver plot?).

箭头头部大小可以通过'MaxHeadSize'属性调整。不幸的是，这个属性不太稳定。坐标轴范围需要随后设置。

```
x1 = [10 30];
y1 = [10 30];
drawArrow(x1,y1,{'MaxHeadSize',0.8,'Color','b','LineWidth',3}); hold on

x2 = [25 15];
y2 = [15 25];
drawArrow(x2,y2,{'MaxHeadSize',10,'Color','r','LineWidth',3}); hold on

xlim([1, 100])
ylim([1, 100])
```



还有一个可调箭头头部的调整方法：

函数 [h] = drawArrow(x,y,xlimits,ylimits,props)

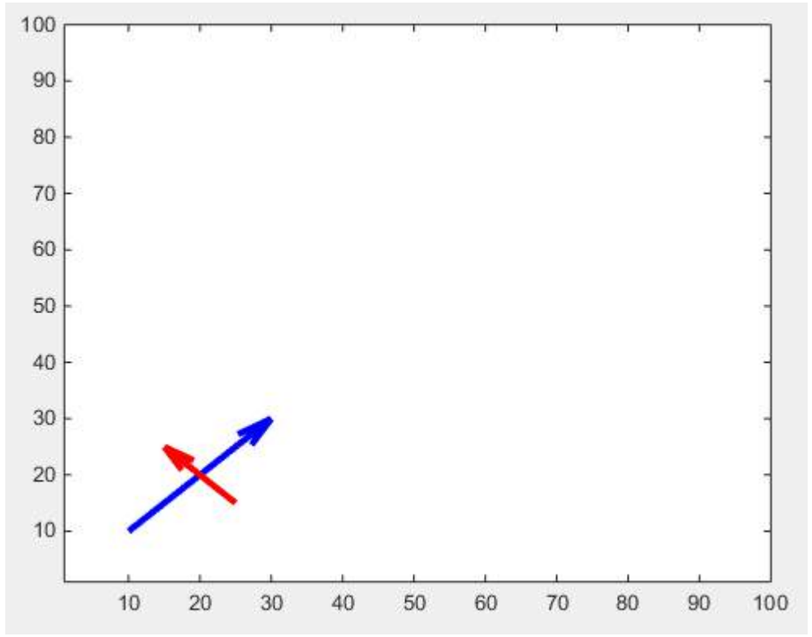
```
xlim(xlimits)
ylim(ylimits)

h = annotation('arrow');
set(h,'parent', gca, ...
    'position', [x(1),y(1),x(2)-x(1),y(2)-y(1)], ...
    'HeadLength', 10, 'HeadWidth', 10, 'HeadStyle', 'cback1', ...
    props{:} );

end
```

你可以在脚本中这样调用它：

```
drawArrow(x1,y1,[1, 100],[1, 100],{'Color','b','LineWidth',3}); hold on
drawArrow(x2,y2,[1, 100],[1, 100],{'Color','r','LineWidth',3}); hold on
```


The arrow head size can be adjust with the 'MaxHeadSize' property. It's not consistent unfortunately. The axes limits need to be set afterwards.

```
x1 = [10 30];
y1 = [10 30];
drawArrow(x1,y1,{'MaxHeadSize',0.8,'Color','b','LineWidth',3}); hold on

x2 = [25 15];
y2 = [15 25];
drawArrow(x2,y2,{'MaxHeadSize',10,'Color','r','LineWidth',3}); hold on

xlim([1, 100])
ylim([1, 100])
```

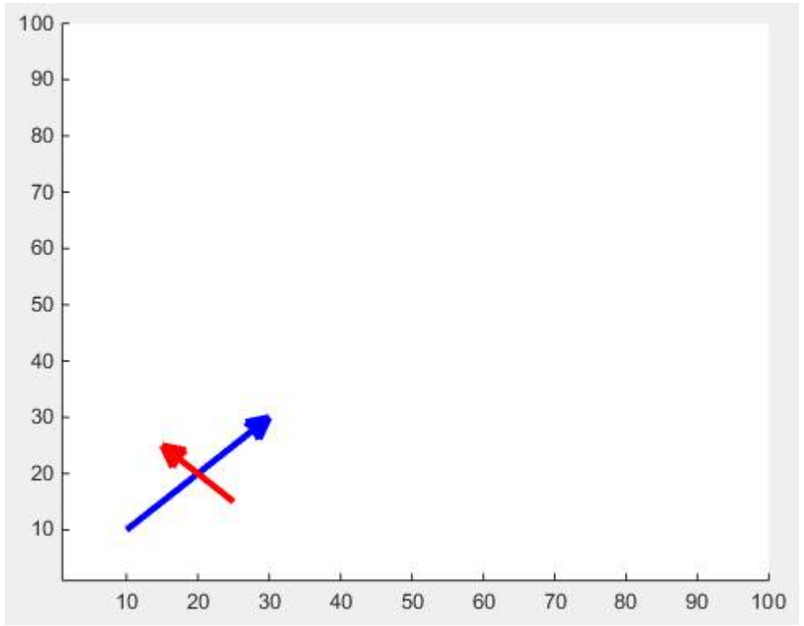


There is another tweak for adjustable arrow heads:

```
function [ h ] = drawArrow( x,y,xlimits,ylimits,props )

xlim(xlimits)
ylim(ylimits)

h = annotation('arrow');
set(h,'parent', gca, ...
    'position', [x(1),y(1),x(2)-x(1),y(2)-y(1)], ...
    'HeadLength', 10, 'HeadWidth', 10, 'HeadStyle', 'cback1', ...
    props{:} );

end
```

which you can call from your script as follows:

```
drawArrow(x1,y1,[1, 100],[1, 100],{'Color','b','LineWidth',3}); hold on
drawArrow(x2,y2,[1, 100],[1, 100],{'Color','r','LineWidth',3}); hold on
```



第16.3节：椭圆

要绘制椭圆，可以使用它的方程。椭圆有长轴和短轴。我们还希望能够在不同的中心点绘制椭圆。因此，我们编写了一个函数，其输入和输出为：

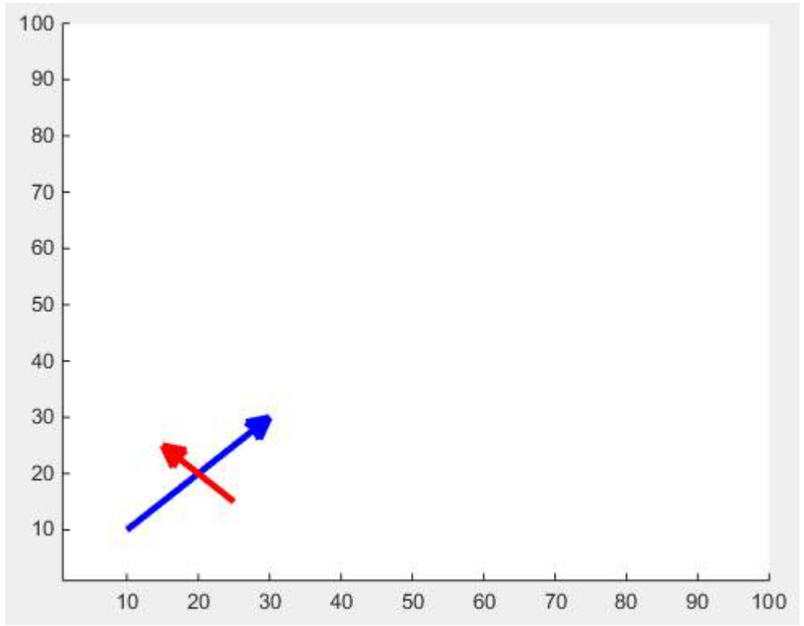
```
输入：
    r1,r2：分别为主轴和次轴轴
    C：椭圆中心(cx,cy)
输出：
     [x,y]: 椭圆周上的点
```

您可以使用以下函数获取椭圆上的点，然后绘制这些点。

```
function [x,y] = getEllipse(r1,r2,C)
beta = linspace(0,2*pi,100);
x = r1*cos(beta) - r2*sin(beta);
y = r1*cos(beta) + r2*sin(beta);
x = x + C(1,1);
y = y + C(1,2);
end
```

示例：

```
[x,y] = getEllipse(1,0.3,[2 3]);
plot(x,y);
```

Section 16.3: Ellipse

To plot an ellipse you can use its equation. An ellipse has a major and a minor axis. Also we want to be able to plot the ellipse on different center points. Therefore we write a function whose inputs and outputs are:

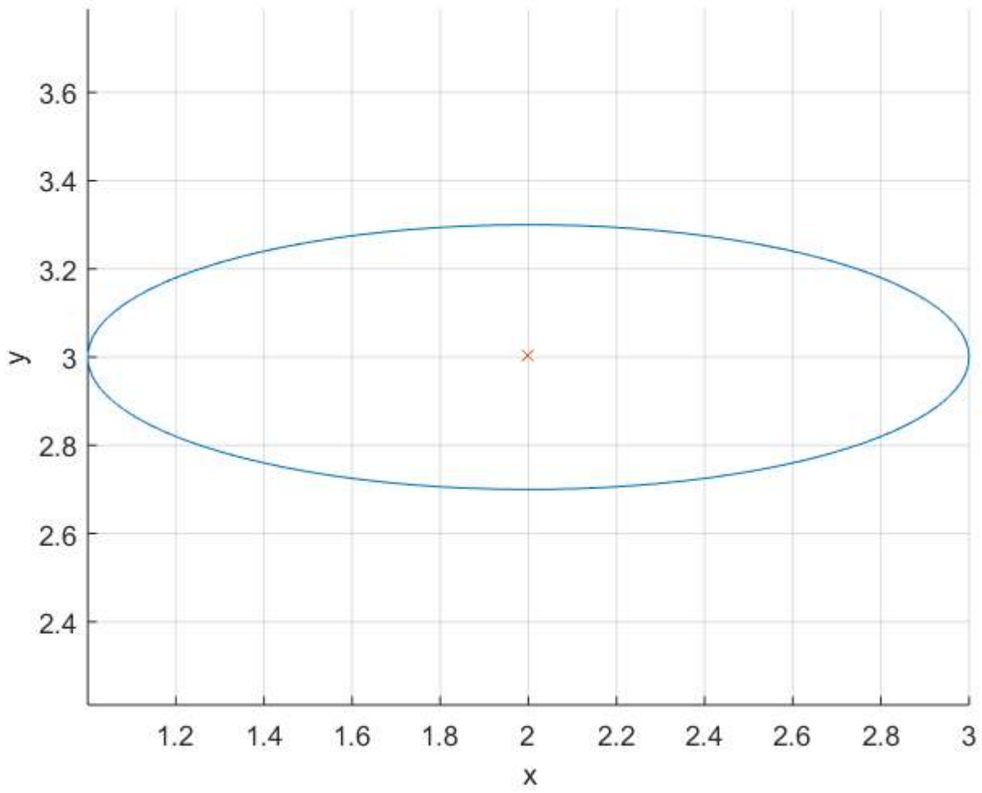
```
Inputs:
    r1,r2: major and minor axis respectively
    C: center of the ellipse (cx,cy)
Output:
    [x,y]: points on the circumference of the ellipse
```

You can use the following function to get the points on an ellipse and then plot those points.

```
function [x,y] = getEllipse(r1,r2,C)
beta = linspace(0,2*pi,100);
x = r1*cos(beta) - r2*sin(beta);
y = r1*cos(beta) + r2*sin(beta);
x = x + C(1,1);
y = y + C(1,2);
end
```

Example:

```
[x,y] = getEllipse(1,0.3,[2 3]);
plot(x,y);
```



第16.4节：伪4D图

一个（m x n）矩阵可以通过使用surf表示为一个曲面；曲面的颜

色会自动根据（m x n）矩阵中的数值设置。如果未指定colormap，则应用默认的颜色映射。

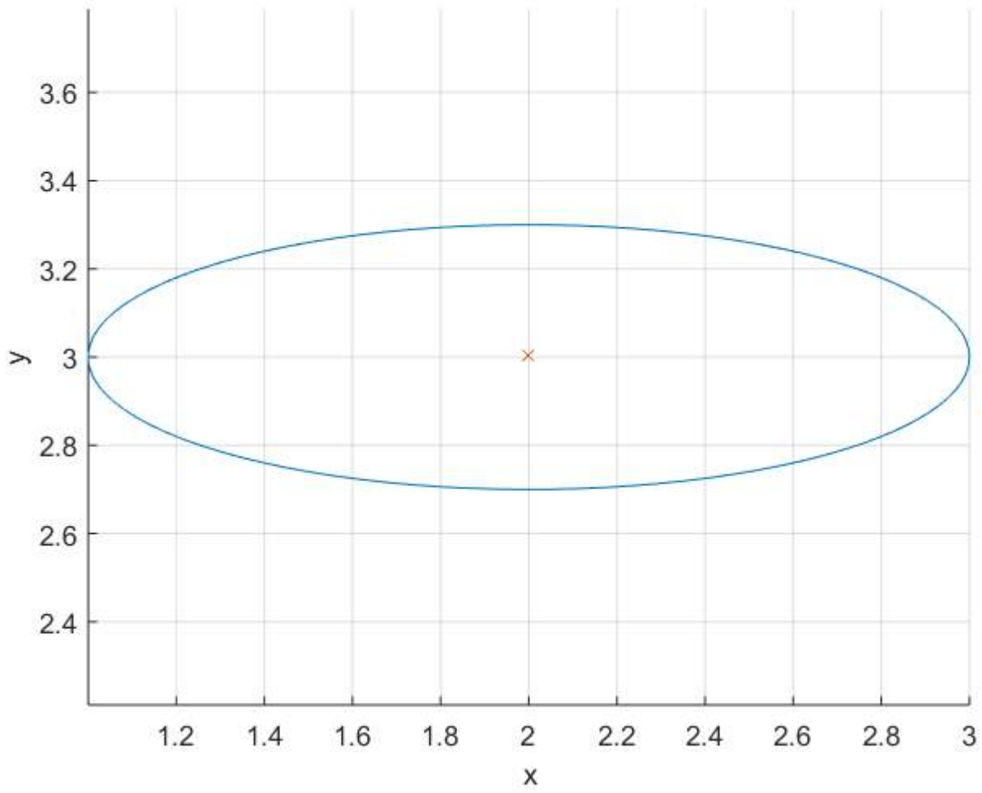
可以添加一个colorbar来显示当前的颜色映射，并指示数据值到颜色映射的映射关系。

在以下示例中，z （m x n）矩阵由以下函数生成：

```
z=x.*y.*sin(x).*cos(y);
```

在区间[-pi,pi]上。可以使用meshgrid函数生成 x 和 y 的值，曲面渲染如下：

```
% 创建一个图形
figure
% 在区间 `[-pi,pi]` 内生成 `x` 和 `y` 值
[x,y] = meshgrid([-pi:.2:pi],[-pi:.2:pi]);
% 在选定区间上计算函数值
z=x.*y.*sin(x).*cos(y);
% 使用 surf 绘制曲面
S=surf(x,y,z);
xlabel('X 轴');
```

Section 16.4: Pseudo 4D plot

A `(m x n)` matrix can be represented by a surface by using surf;

The color of the surface is automatically set as function of the values in the `(m x n)` matrix. If the colormap is not specified, the default one is applied.

A colorbar can be added to display the current colormap and indicate the mapping of data values into the colormap.

In the following example, the `z (m x n)` matrix is generated by the function:

```
z=x.*y.*sin(x).*cos(y);
```

over the interval `[-pi,pi]`. The x and y values can be generated using the meshgrid function and the surface is rendered as follows:

```
% Create a Figure
figure
% Generate the `x` and `y` values in the interval `[-pi,pi]`
[x,y] = meshgrid([-pi:.2:pi],[-pi:.2:pi]);
% Evaluate the function over the selected interval
z=x.*y.*sin(x).*cos(y);
% Use surf to plot the surface
S=surf(x,y,z);
xlabel('X Axis');
```

```
ylabel('Y 轴');
zlabel('Z 轴');
grid minor
colormap('hot')
colorbar
```

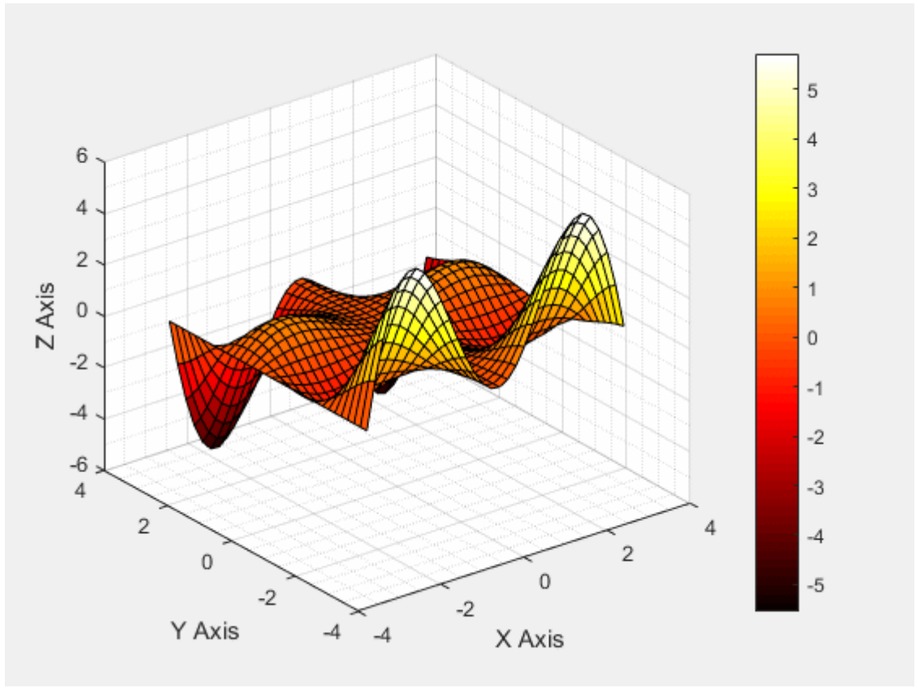


图1

现在可能存在附加信息与z矩阵的值相关联，并且它们存储在另一个(m x n)矩阵中

可以通过修改表面着色的方式，将这些附加信息添加到图中。

这将允许拥有某种4D图：在由第一个(m x n)矩阵生成的表面3D表示的基础上，第四维将由第二个(m x n)矩阵中包含的数据表示。

可以通过调用带有4个输入参数的surf来创建这样的图：

```
surf(x,y,z,C)
```

其中C参数是第二个矩阵（必须与z大小相同），用于定义表面的颜色。

在以下示例中，C矩阵由函数生成：

```
C=10*sin(0.5*(x.^2.+y.^2))*33;
```

在区间`[-pi,pi]`

由C生成的表面是


```
ylabel('Y Axis');
zlabel('Z Axis');
grid minor
colormap('hot')
colorbar
```

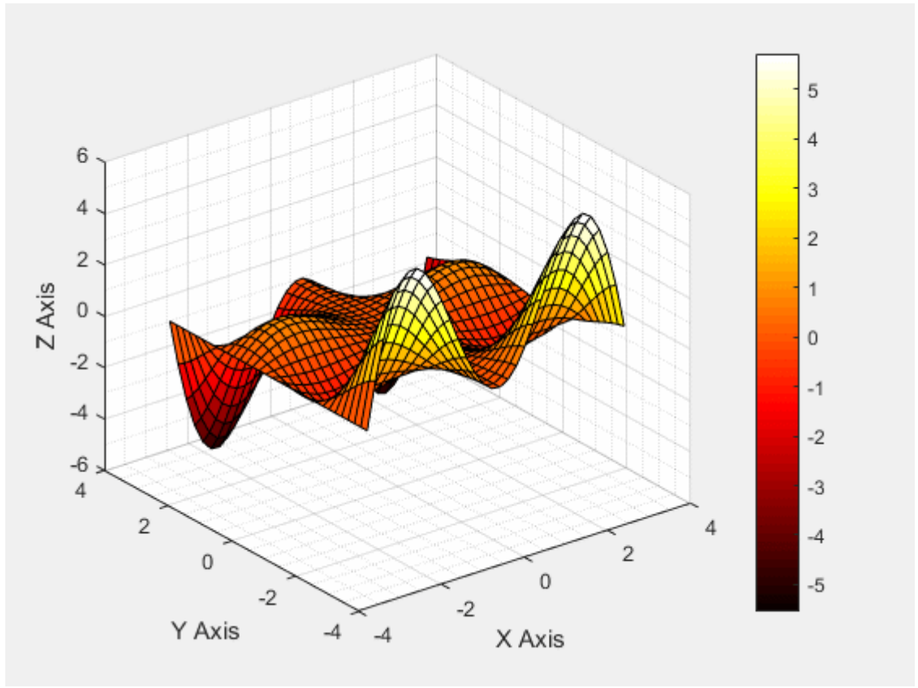


Figure 1

Now it could be the case that additional information are linked to the values of the z matrix and they are store in another `(m x n)` matrix

It is possible to add these additional information on the plot by modifying the way the surface is colored.

This will allows having kinda of 4D plot: to the 3D representation of the surface generated by the first `(m x n)` matrix, the fourth dimension will be represented by the data contained in the second `(m x n)` matrix.

It is possible to create such a plot by calling `surf` with 4 input:

```
surf(x,y,z,C)
```

where the `C` parameter is the second matrix (which has to be of the same size of z) and is used to define the color of the surface.

In the following example, the `C` matrix is generated by the function:

```
C=10*sin(0.5*(x.^2.+y.^2))*33;
```

over the interval `[-pi,pi]`

The surface generated by `C` is

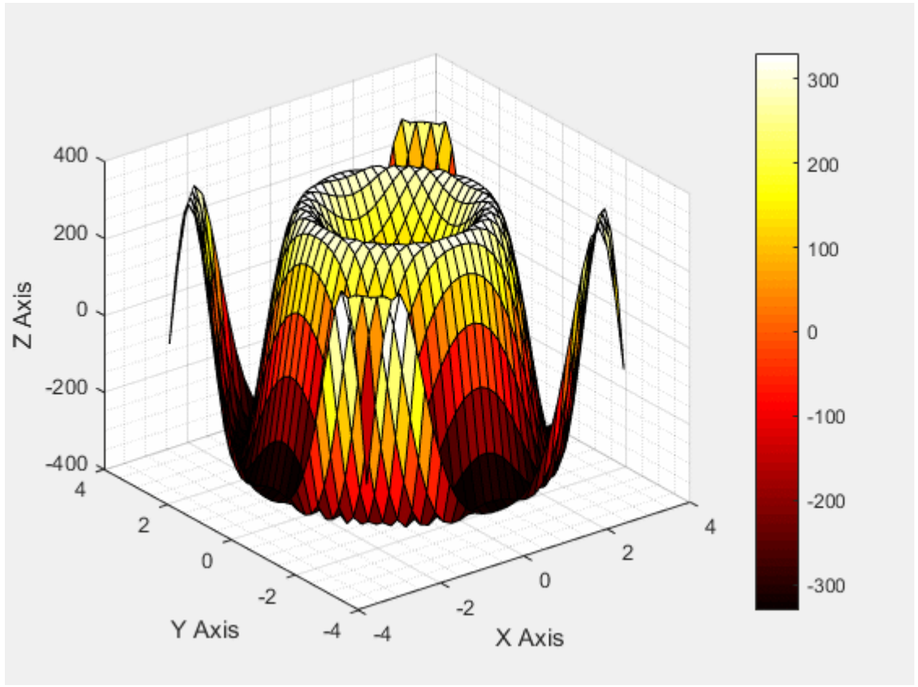


图2

现在我们可以用四个输入调用surf：

```
图
surf(x,y,z,C)
% 阴影插值
xlabel('X轴');
ylabel('Y轴');
zlabel('Z轴');
grid minor
colormap('hot')
colorbar
```

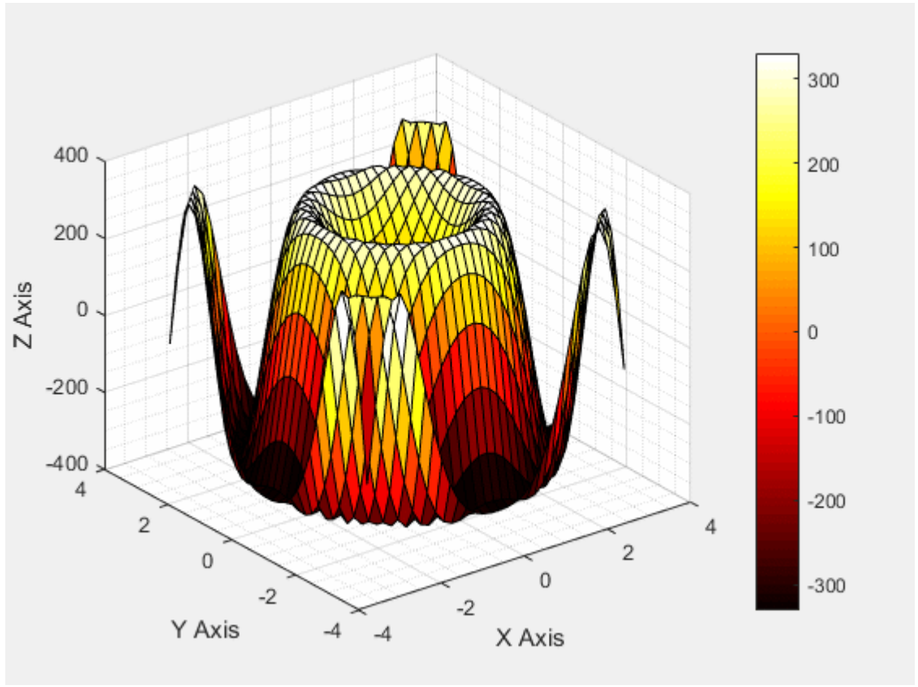



Figure 2

Now we can call `surf` with four input:

```
figure
surf(x,y,z,C)
% shading interp
xlabel('X Axis');
ylabel('Y Axis');
zlabel('Z Axis');
grid minor
colormap('hot')
colorbar
```

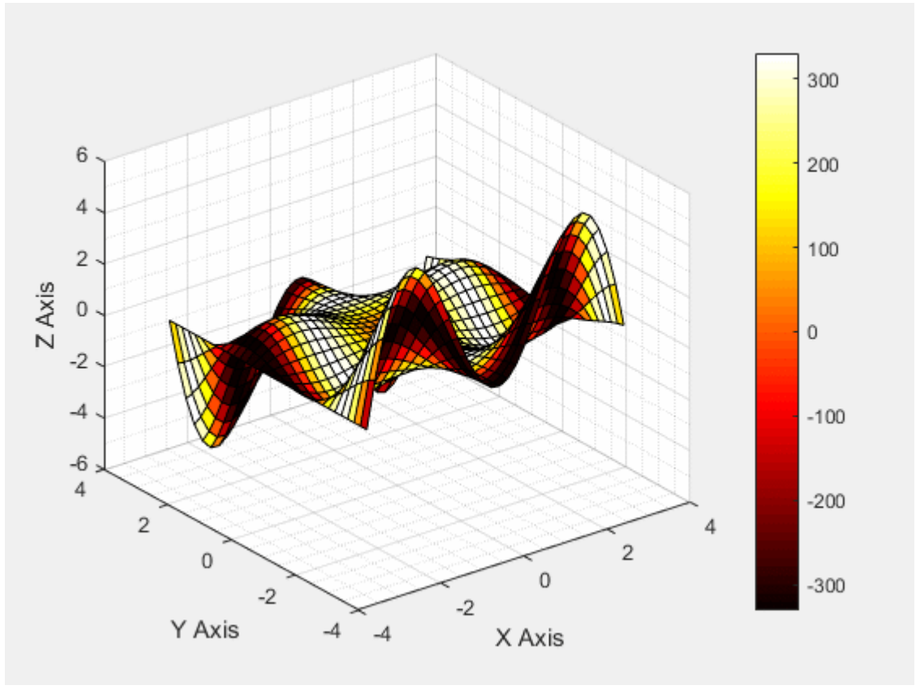
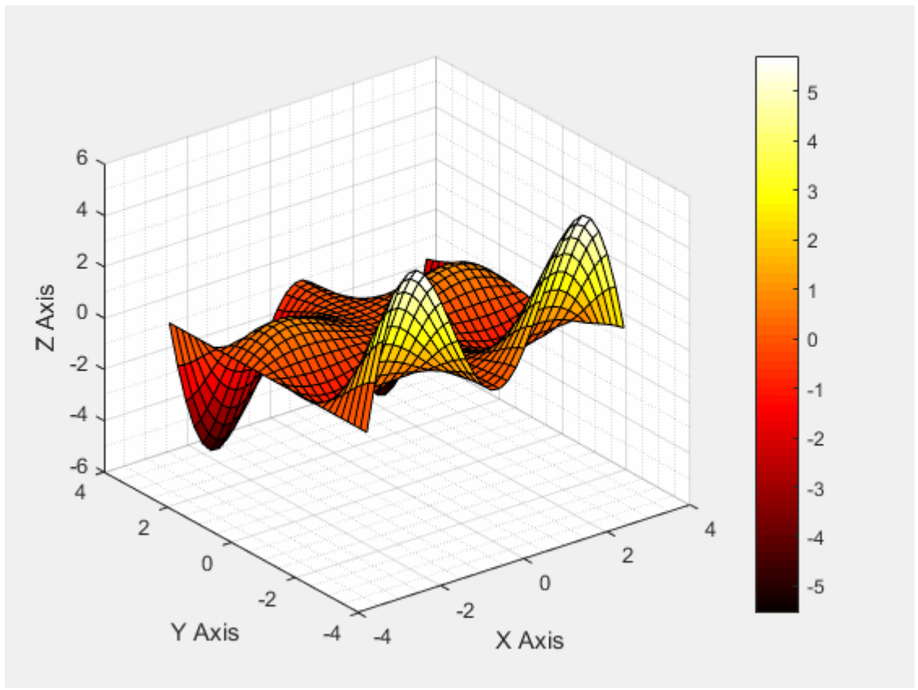


图3

比较图1和图3，我们可以注意到：

- 曲面的形状对应于z值（第一个(m x n)矩阵）曲面的颜色（及其范围，由colorbar
- 给出）对应于C值（第一个(m x n)矩阵）



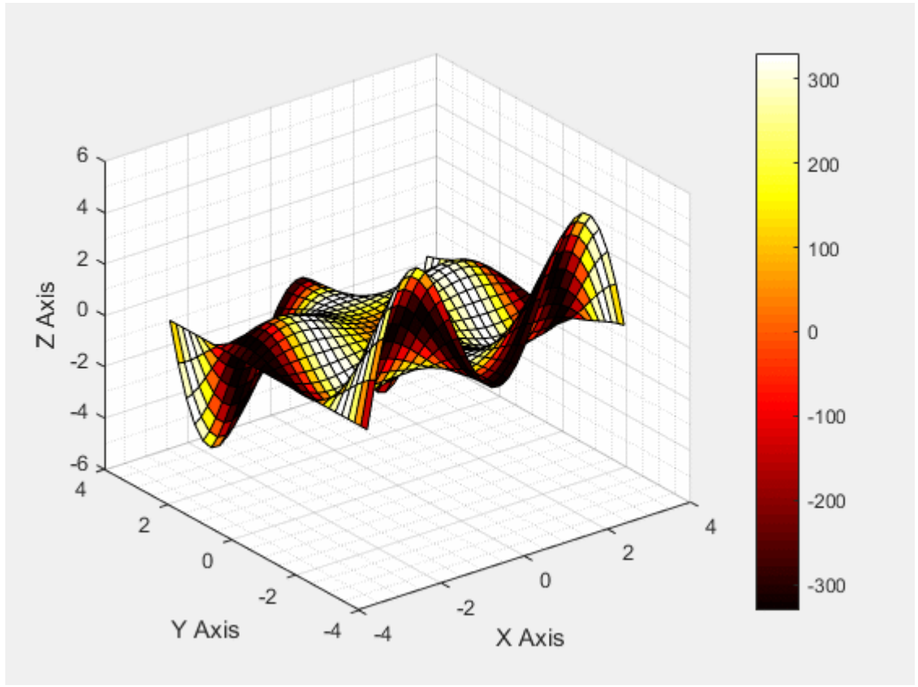


Figure 3

Comparing Figure 1 and Figure 3, we can notice that:

- the shape of the surface corresponds to the `z` values (the first `(m x n)` matrix)
- the colour of the surface (and its range, given by the colorbar) corresponds to the `C` values (the first `(m x n)` matrix)

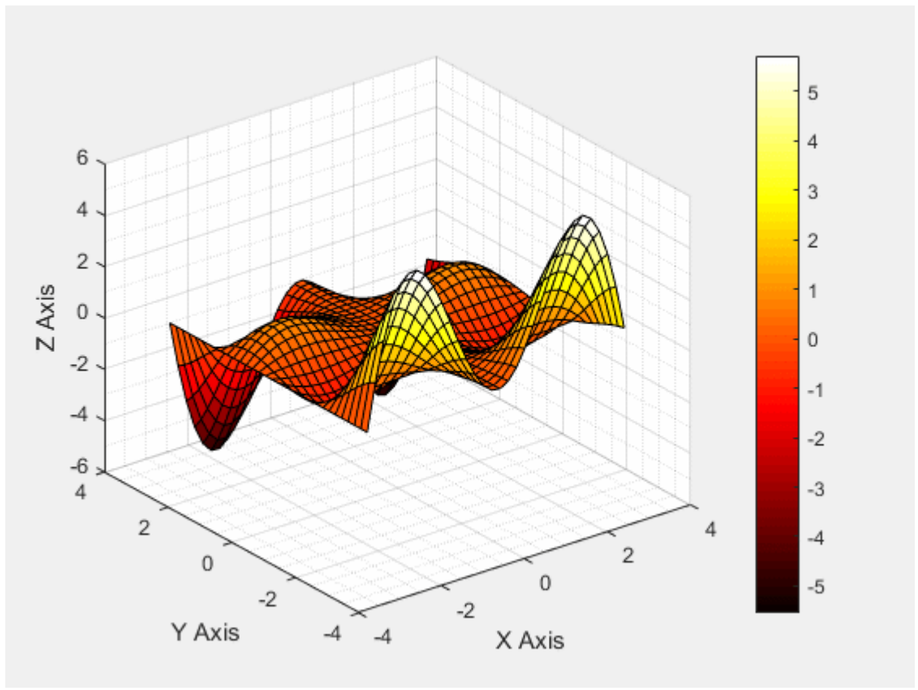
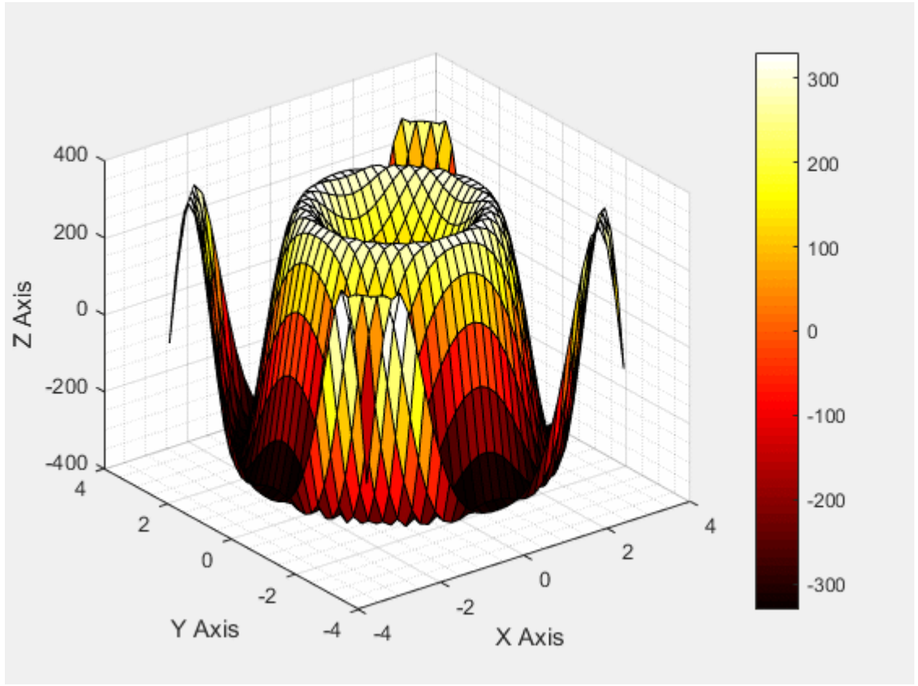


图4

当然，可以在图中交换z和C，以使表面形状由C矩阵给出，颜色由z矩阵给出：

```
图
surf(x,y,C,z)
% 阴影插值
xlabel('X 轴');
ylabel('Y 轴');
zlabel('Z 轴');
grid minor
colormap('hot')
colorbar
```

并且比较图2和图4



第16.5节：快速绘图

有三种主要方法来做顺序绘图或动画：plot(x,y)，set(h，'XData'，y，'YData'，y)和animatedline。如果你想让动画平滑，你需要高效的绘图，这三种方法并不等价。

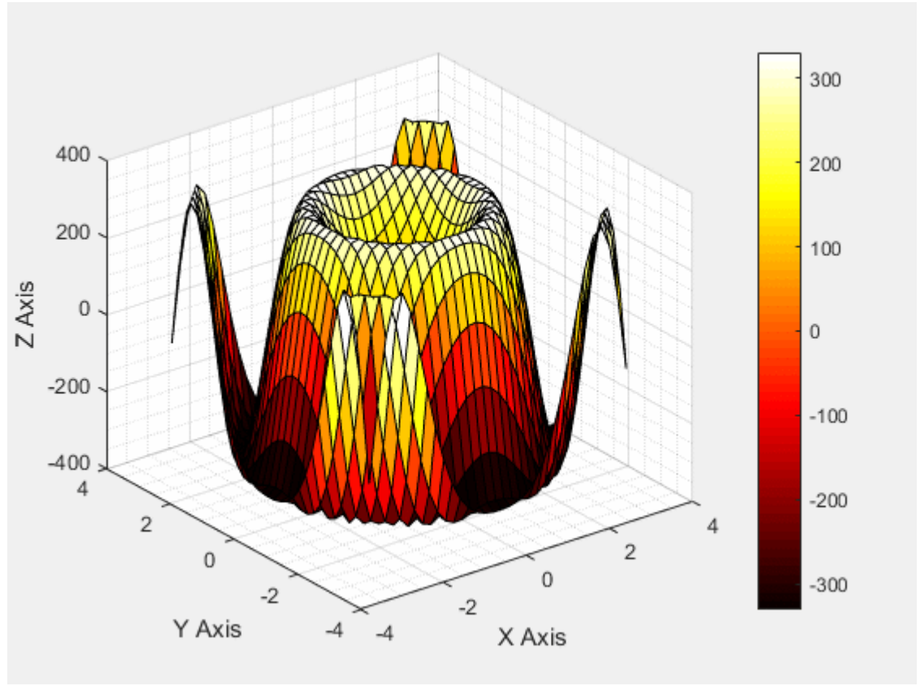
```
% 绘制一个相位逐渐增加的正弦波，共500步
x = linspace(0，2*pi，100);

图
tic
for thetha = linspace(0 , 10*pi , 500)
    y = sin(x + thetha);
    plot(x,y)
    drawnow
end
```


Figure 4

Of course, it is possible to swap `z` and `C` in the plot to have the shape of the surface given by the `C` matrix and the color given by the `z` matrix:

```
figure
surf(x,y,C,z)
% shading interp
xlabel('X Axis');
ylabel('Y Axis');
zlabel('Z Axis');
grid minor
colormap('hot')
colorbar
```

and to compare Figure 2 with Figure 4



Section 16.5: Fast drawing

There are three main ways to do sequential plot or animations: `plot(x,y)`, `set(h , 'XData' , y, 'YData' , y)` and `animatedline`. If you want your animation to be smooth, you need efficient drawing, and the three methods are not equivalent.

```
% Plot a sin with increasing phase shift in 500 steps
x = linspace(0 , 2*pi , 100);

figure
tic
for thetha = linspace(0 , 10*pi , 500)
    y = sin(x + thetha);
    plot(x,y)
    drawnow
end
```

```
toc
```

我得到5.278172秒。plot函数基本上每次都会删除并重新创建线对象。更新绘图的更高效方法是使用Line对象的XData和YData属性。

```
tic
h = [];    % 线对象的句柄
for thetha = linspace(0 , 10*pi , 500)
    y = sin(x + thetha);

    if isempty(h)
        % 如果 Line 仍然不存在，则创建它
       h = plot(x,y);
    否则
        % 如果 Line 存在，则更新它
        set(h , 'YData' , y)
    end
    drawnow
end
toc
```

现在我得到了2.741996秒，效果好多了！

animatedline 是一个相对较新的函数，于 2014b 版本引入。让我们看看它的表现如何：

```
tic
h = animatedline;
for thetha = linspace(0 , 10*pi , 500)
    y = sin(x + thetha);
clearpoints(h)
    addpoints(h , x , y)
    drawnow
end
toc
```

3.360569秒，虽然不如更新现有图形快，但仍比plot(x,y)要好。

当然，如果你只需要绘制一条线，比如这个例子，三种方法几乎等效，并且都能实现流畅的动画。但如果你有更复杂的图形，更新现有的Line对象会带来明显差异。

第16.6节：多边形

创建向量以保存顶点的 x 和 y 位置，将这些传入patch。

单个多边形

```
X=rand(1,4); Y=rand(1,4);
h=patch(X,Y,'红色');
```



```
toc
```

I get 5.278172 seconds. The plot function basically deletes and recreates the line object each time. A more efficient way to update a plot is to use the XData and YData properties of the Line object.

```
tic
h = [];   % Handle of line object
for thetha = linspace(0 , 10*pi , 500)
    y = sin(x + thetha);

    if isempty(h)
        % If Line still does not exist, create it
        h = plot(x,y);
    else
        % If Line exists, update it
        set(h , 'YData' , y)
    end
    drawnow
end
toc
```

Now I get 2.741996 seconds, much better!

animatedline is a relatively new function, introduced in 2014b. Let's see how it fares:

```
tic
h = animatedline;
for thetha = linspace(0 , 10*pi , 500)
    y = sin(x + thetha);
    clearpoints(h)
    addpoints(h , x , y)
    drawnow
end
toc
```

3.360569 seconds, not as good as updating an existing plot, but still better than plot(x,y).

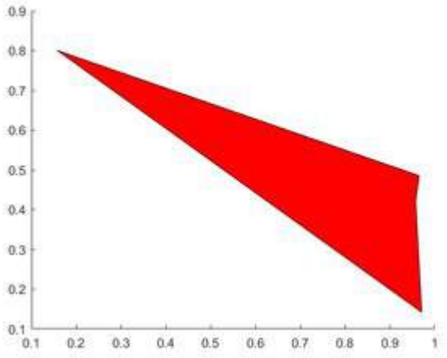
Of course, if you have to plot a single line, like in this example, the three methods are almost equivalent and give smooth animations. But if you have more complex plots, updating existing Line objects will make a difference.

Section 16.6: Polygon(s)

Create vectors to hold the x- and y-locations of vertices, feed these into patch.

Single Polygon

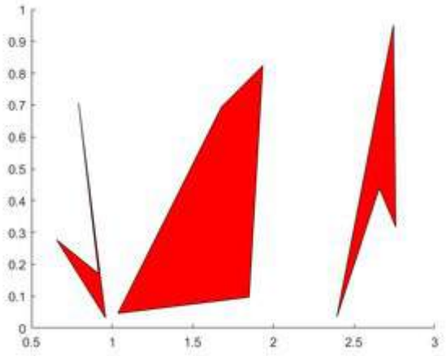
```
X=rand(1,4); Y=rand(1,4);
h=patch(X,Y,'red');
```

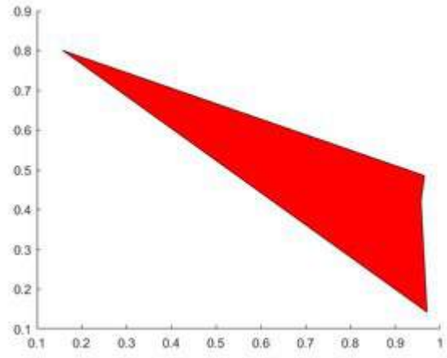


多个多边形

每个多边形的顶点占据X和Y的每一列。

```
X=rand(4,3); Y=rand(4,3);
for i=2:3
X(:,i)=X(:,i)+(i-1); % 创建水平偏移以便可见
end

h=patch(X,Y,'红色');
```

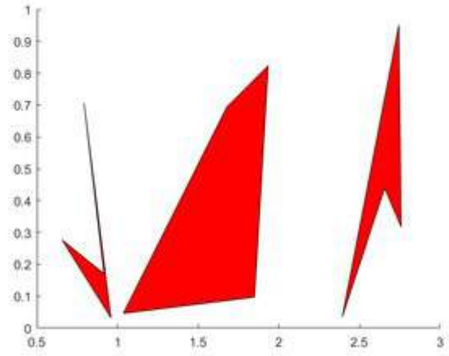




Multiple Polygons

Each polygon's vertices occupy one column of each of `X`, `Y`.

```
X=rand(4,3); Y=rand(4,3);
for i=2:3
  X(:,i)=X(:,i)+(i-1); % create horizontal offsets for visibility
end

h=patch(X,Y,'red');
```



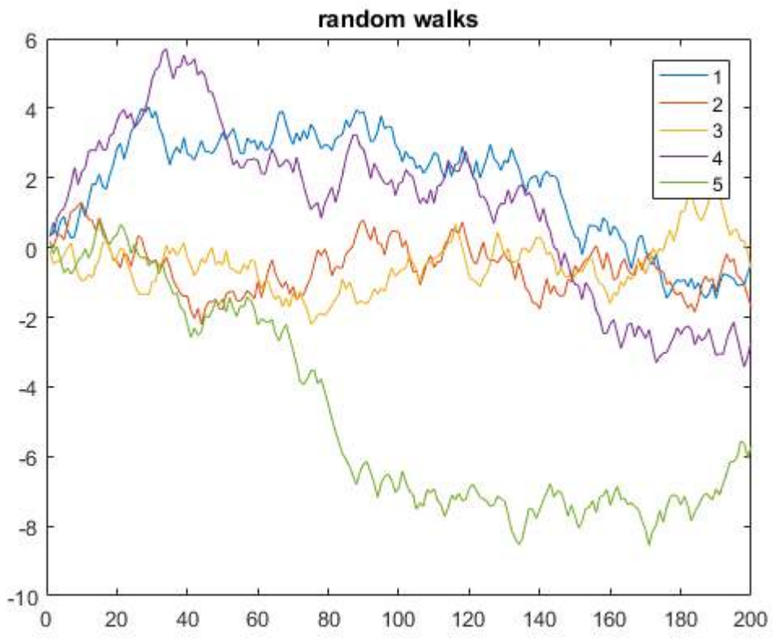
第17章：金融应用

第17.1节：随机游走

以下是一个显示5条一维随机游走，每条200步的示例：

```
y = cumsum(rand(200,5) - 0.5);

plot(y)
legend('1', '2', '3', '4', '5')
title('随机游走')
```

在上述代码中，y 是一个5列的矩阵，每列长度为200。由于省略了 x，默认使用 y 的行号（相当于将 x=1:200 作为x轴）。这样，plot 函数会自动用不同颜色将多个y向量绘制在相同的x向量上。



第17.2节：单变量几何布朗运动

几何布朗运动（GBM）的动态由以下随机微分方程（SDE）描述：

$$dS_t = \mu S_t\,dt + \sigma S_t\,dW_t$$

我可以使用该SDE的exact解

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right)$$

来生成符合GBM路径的轨迹。

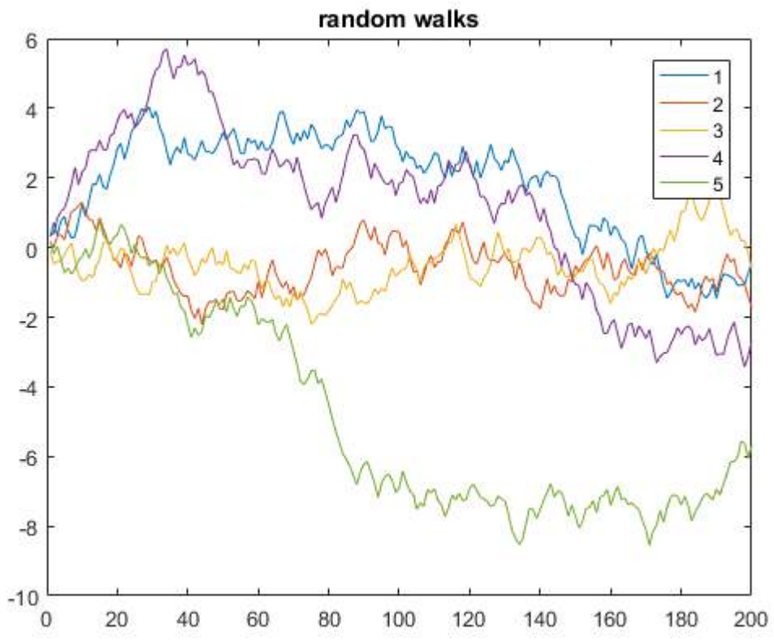
Chapter 17: Financial Applications

Section 17.1: Random Walk

The following is an example that displays 5 one-dimensional random walks of 200 steps:

```
y = cumsum(rand(200,5) - 0.5);

plot(y)
legend('1', '2', '3', '4', '5')
title('random walks')
```

In the above code, y is a matrix of 5 columns, each of length 200. Since x is omitted, it defaults to the row numbers of y (equivalent to using `x=1:200` as the x-axis). This way the `plot` function plots multiple y-vectors against the same x-vector, each using a different color automatically.



Section 17.2: Univariate Geometric Brownian Motion

The dynamics of the Geometric Brownian Motion (GBM) are described by the following stochastic differential equation (SDE):

$$dS_t = \mu S_t\,dt + \sigma S_t\,dW_t$$

I can use the **exact** solution to the SDE

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right)$$

to generate paths that follow a GBM.

给定为期一年的每日参数

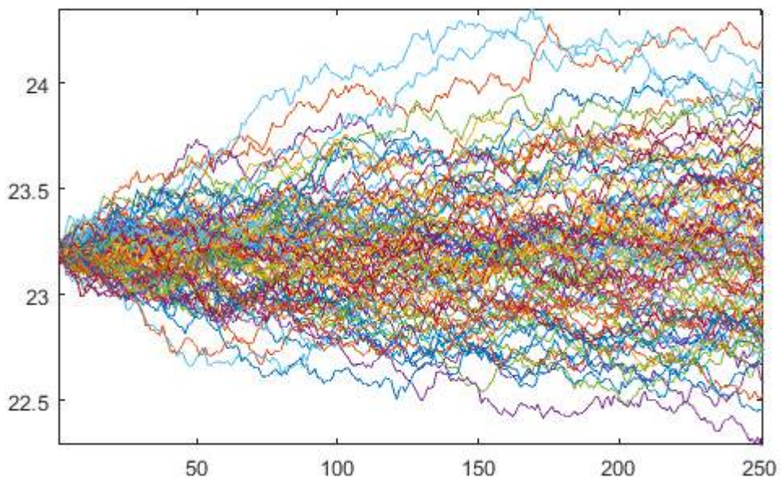
```
mu     = 0.08/250;
sigma  = 0.25/sqrt(250);
dt     = 1/250;
npaths = 100;
nsteps = 250;
S0     = 23.2;
```

我们可以得到从0开始的布朗运动（BM）W，并用它来获得从S0开始的几何布朗运动（GBM）

```
% BM
epsilon = randn(nsteps, npaths);
W       = [zeros(1,npaths); sqrt(dt)*cumsum(epsilon)];

% GBM
t = (0:nsteps)'*dt;
Y = bsxfun(@plus, (mu-0.5*sigma.^2)*t, sigma*W);
Y = S0*exp(Y);
```

这将生成路径

```
plot(Y)
```



Given daily parameters for a year-long simulation

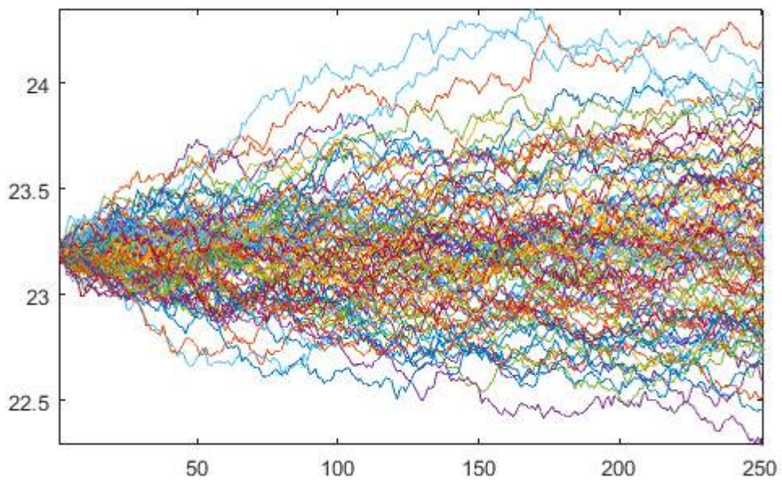
```
mu     = 0.08/250;
sigma  = 0.25/sqrt(250);
dt     = 1/250;
npaths = 100;
nsteps = 250;
S0     = 23.2;
```

we can get the Brownian Motion (BM) `W` starting at 0 and use it to obtain the GBM starting at `S0`

```
% BM
epsilon = randn(nsteps, npaths);
W       = [zeros(1,npaths); sqrt(dt)*cumsum(epsilon)];

% GBM
t = (0:nsteps)'*dt;
Y = bsxfun(@plus, (mu-0.5*sigma.^2)*t, sigma*W);
Y = S0*exp(Y);
```

Which produces the paths

```
plot(Y)
```



第18章：傅里叶变换和逆傅里叶变换

参数	描述
X	这是你的输入时域信号，应该是一个数值向量。
n	这是称为变换长度的NFFT参数，可以将其视为FFT结果的分辨率，它必须是2的幂次方数字（例如64、128、256……2^N）。
dim	这是你想计算FFT的维度，如果想在水平方向计算FFT，使用1；如果想在垂直方向计算FFT，使用2——注意这个参数通常留空，因为函数能够自动检测向量的方向。

第18.1节：在MATLAB中实现简单的傅里叶变换

傅里叶变换可能是数字信号处理中的第一课，它的应用无处不在，是分析数据（各个领域）或信号的强大工具。MATLAB拥有一套强大的傅里叶变换工具箱。在本例中，我们将使用傅里叶变换分析一个基本的正弦波信号，并利用FFT生成有时称为周期图的内容：

```
%信号生成
A1=10;                  % 振幅1
A2=10;                  % 振幅2
w1=2*pi*0.2;            % 角频率1
w2=2*pi*0.225;          % 角频率2
Ts=1;                   % 采样时间
N=64;                   % 生成的处理样本数
K=5;                    % 独立过程实现次数
sgm=1;                  % 噪声的标准差
n=repmat([0:N-1].',1,K);              % 生成分辨率
phi1=repmat(rand(1,K)*2*pi,N,1);      % 随机相位矩阵1
phi2=repmat(rand(1,K)*2*pi,N,1);      % 随机相位矩阵2
x=A1*sin(w1*n*Ts+phi1)+A2*sin(w2*n*Ts+phi2)+sgm*randn(N,K);    % 结果信号

NFFT=256;               % FFT长度
F=fft(x,NFFT);          % 快速傅里叶变换结果
Z=1/N*abs(F).^2;        % 将FFT结果转换为周期图
```

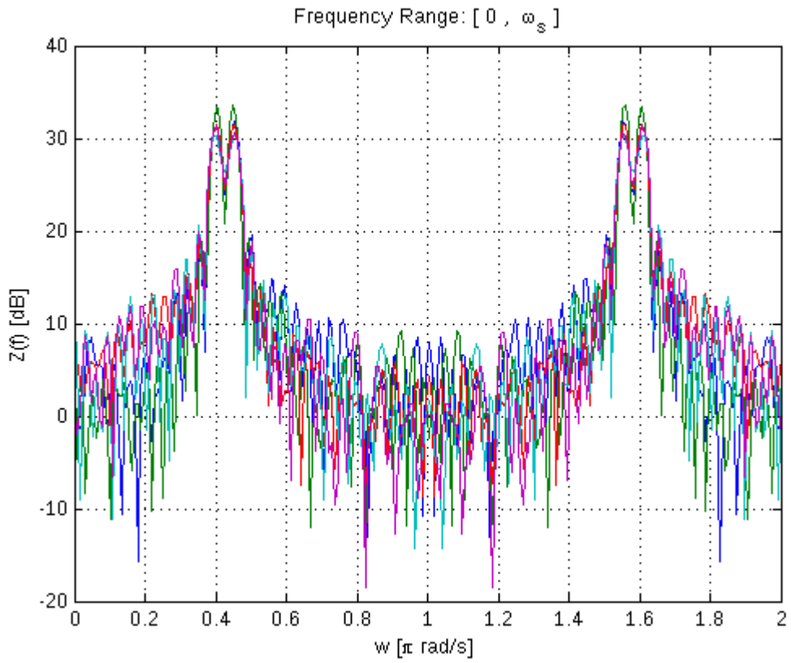

Chapter 18: Fourier Transforms and Inverse Fourier Transforms

Parameter	Description
X	this is your input Time-Domain signal, it should be a vector of numerics.
n	this is the NFFT parameter known as Transform Length, think of it as resolution of your FFT result, it MUST be a number that is a power of 2 (i.e. 64,128,256…2^N)
dim	this is the dimension you want to compute FFT on, use 1 if you want to compute your FFT in the horizontal direction and 2 if you want to compute your FFT in the vertical direction - Note this parameter is usually left blank, as the function is capable of detecting the direction of your vector.

Section 18.1: Implement a simple Fourier Transform in MATLAB

Fourier Transform is probably the first lesson in Digital Signal Processing, it's application is everywhere and it is a powerful tool when it comes to analyze data (in all sectors) or signals. MATLAB has a set of powerful toolboxes for Fourier Transform. In this example, we will use Fourier Transform to analyze a basic sine-wave signal and generate what is sometimes known as a Periodogram using FFT:

```
%Signal Generation
A1=10;                  % Amplitude 1
A2=10;                  % Amplitude 2
w1=2*pi*0.2;            % Angular frequency 1
w2=2*pi*0.225;          % Angular frequency 2
Ts=1;                   % Sampling time
N=64;                   % Number of process samples to be generated
K=5;                    % Number of independent process realizations
sgm=1;                  % Standard deviation of the noise
n=repmat([0:N-1].',1,K);              % Generate resolution
phi1=repmat(rand(1,K)*2*pi,N,1);      % Random phase matrix 1
phi2=repmat(rand(1,K)*2*pi,N,1);      % Random phase matrix 2
x=A1*sin(w1*n*Ts+phi1)+A2*sin(w2*n*Ts+phi2)+sgm*randn(N,K);    % Resulting Signal

NFFT=256;               % FFT length
F=fft(x,NFFT);          % Fast Fourier Transform Result
Z=1/N*abs(F).^2;        % Convert FFT result into a Periodogram
```



请注意，离散傅里叶变换在MATLAB中是通过快速傅里叶变换（fft）实现的，两者结果相同，但FFT是DFT的快速实现。

```
图
w=linspace(0,2,NFFT);
plot(w,10*log10(Z)),grid;
xlabel('w [\pi rad/s]')
ylabel('Z(f) [dB]')
title('频率范围: [ 0 ,  \omega_s ]')
```

第18.2节：图像与多维傅里叶变换

在医学成像、光谱学、图像处理、密码学及其他科学与工程领域，常常需要计算图像的多维傅里叶变换。这在MATLAB中相当简单：（多维）图像本质上就是n维矩阵，而傅里叶变换是线性算子：只需沿其他维度迭代进行傅里叶变换。MATLAB提供了

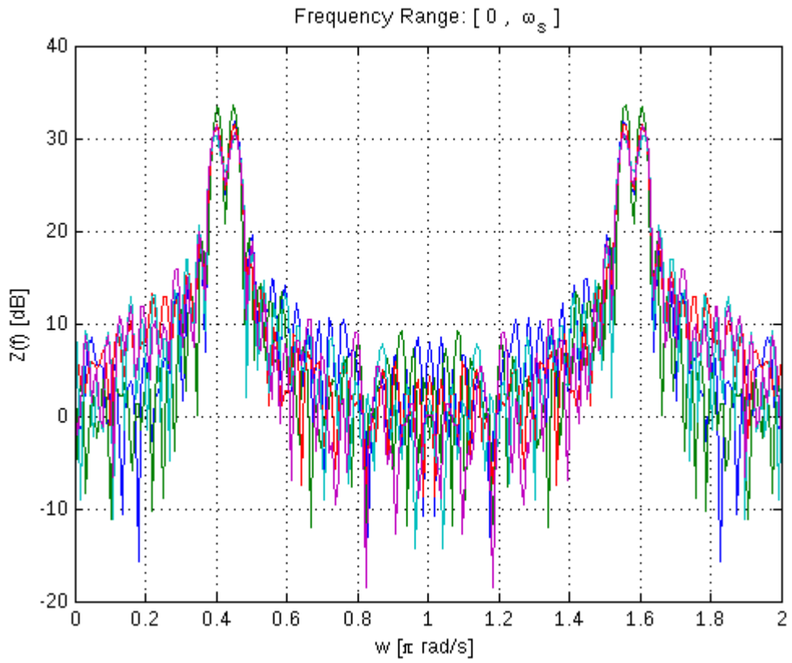
fft2和ifft2用于二维变换，或fftn用于n维变换。

一个潜在的陷阱是，图像的傅里叶变换通常以“中心排序”方式显示，即k空间的原点位于图像中心。MATLAB提供了fftshift命令来适当交换傅里叶变换中直流分量的位置。这种排序方式大大简化了常见图像处理技术的实现，下面将举例说明其中一种。

零填充

一种“快速且简便”的方法将小图像插值放大，是对其进行傅里叶变换，在傅里叶变换结果中填充零，然后再进行逆变换。这实际上是用sinc形状的基函数在每个像素之间进行插值，常用于放大低分辨率的医学影像数据。我们先从加载一个内置的图像示例开始

```
%加载示例图像
```

Note that the Discrete Fourier Transform is implemented by Fast Fourier Transform (fft) in MATLAB, both will yield the same result, but FFT is a fast implementation of DFT.

```
figure
w=linspace(0,2,NFFT);
plot(w,10*log10(Z)),grid;
xlabel('w [\pi rad/s]')
ylabel('Z(f) [dB]')
title('Frequency Range: [ 0 ,  \omega_s ]')
```

Section 18.2: Images and multidimensional FTs

In medical imaging, spectroscopy, image processing, cryptography and other areas of science and engineering it is often the case that one wishes to compute multidimensional Fourier transforms of images. This is quite straightforward in MATLAB: (multidimensional) images are just n-dimensional matrices, after all, and Fourier transforms are linear operators: one just iteratively Fourier transforms along other dimensions. MATLAB provides `fft2` and `ifft2` to do this in 2-d, or `fftn` in n-dimensions.

One potential pitfall is that the Fourier transform of images are usually shown "centric ordered", i.e. with the origin of k-space in the middle of the picture. MATLAB provides the `fftshift` command to swap the location of the DC components of the Fourier transform appropriately. This ordering notation makes it substantially easier to perform common image processing techniques, one of which is illustrated below.

Zero filling

One "quick and dirty" way to interpolate a small image to a larger size is to Fourier transform it, pad the Fourier transform with zeros, and then take the inverse transform. This effectively interpolates between each pixel with a sinc shaped basis function, and is commonly used to up-scale low resolution medical imaging data. Let's start by loading a built-in image example

```
%Load example image
```

```
I=imread('coins.png'); %加载示例数据——coins.png是MATLAB内置图像
I=double(I); %转换为双精度——imread返回整数类型
imageSize = size(I); % I是一个246 x 300的二维图像

%显示图像
imagesc(I); colormap gray; axis equal;
%imagesc显示按最大强度缩放的图像
```

现在我们可以获得I的傅里叶变换。为了说明fftshift的作用，我们比较两种方法：

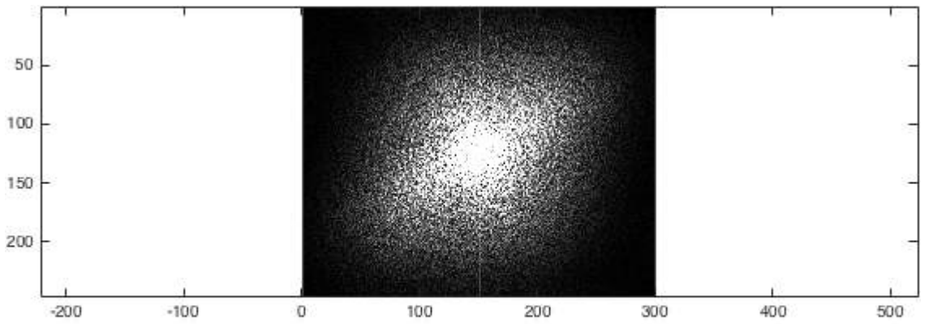
```
% 傅里叶变换
%获得I的中心化和非中心化顺序的傅里叶变换
k=fftshift(fft2(fftshift(I)));
kwrong=fft2(I);

%仅作比较，显示两个变换的幅度：
figure; subplot(2,1,1);
imagesc(abs(k),[0 1e4]); colormap gray; axis equal;
subplot(2,1,2);
imagesc(abs(kwrong),[0 1e4]); colormap gray; axis equal;
%(imagesc 的第二个参数设置颜色轴以使差异更明显)。
```

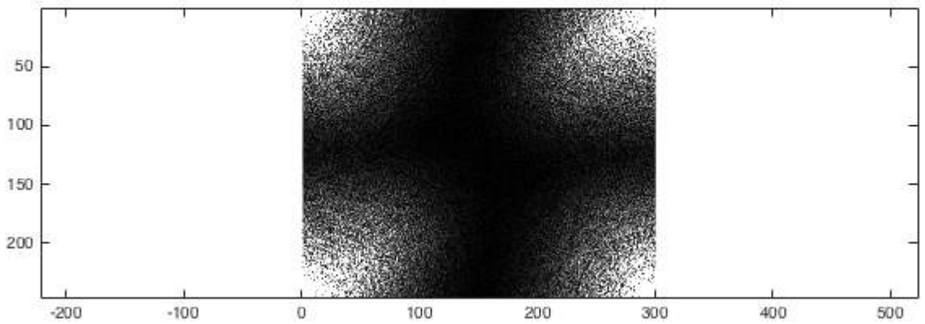


```
I=imread('coins.png'); %Load example data -- coins.png is builtin to MATLAB
I=double(I); %Convert to double precision -- imread returns integers
imageSize = size(I); % I is a 246 x 300 2D image

%Display it
imagesc(I); colormap gray; axis equal;
%imagesc displays images scaled to maximum intensity
```

We can now obtain the Fourier transform of I. To illustrate what `fftshift` does, let's compare the two methods:

```
% Fourier transform
%Obtain the centric- and non-centric ordered Fourier transform of I
k=fftshift(fft2(fftshift(I)));
kwrong=fft2(I);

%Just for the sake of comparison, show the magnitude of both transforms:
figure; subplot(2,1,1);
imagesc(abs(k),[0 1e4]); colormap gray; axis equal;
subplot(2,1,2);
imagesc(abs(kwrong),[0 1e4]); colormap gray; axis equal;
%(The second argument to imagesc sets the colour axis to make the difference clear).
```

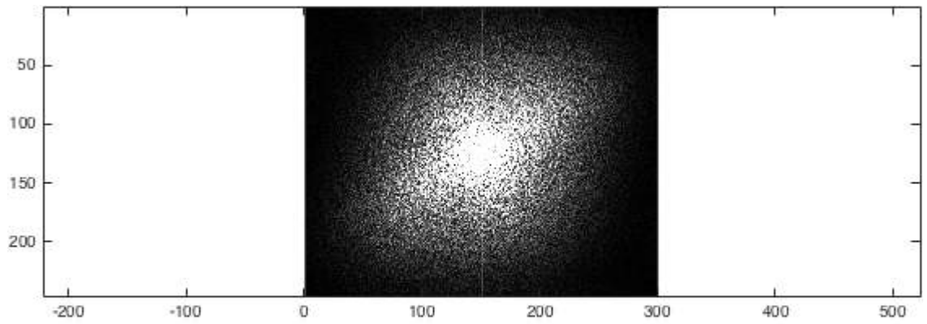


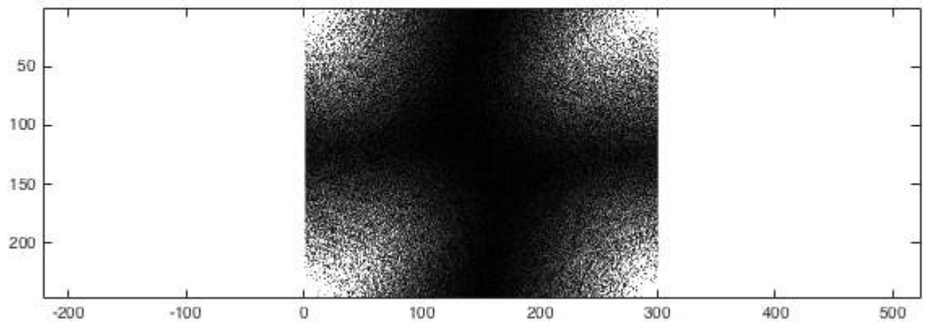


我们现在已经获得了一个示例图像的二维傅里叶变换。为了进行零填充，我们需要对每个 k 空间，在边缘填充零，然后进行反变换：

```
%零填充
kzf = zeros(imageSize .* 2); %生成一个 492x600 的空数组用于存放结果
kzf(end/4:3*end/4-1,end/4:3*end/4-1) = k; %将 k 放在中间
kzfwrong = zeros(imageSize .* 2); %生成一个 492x600 的空数组用于存放结果
kzfwrong(end/4:3*end/4-1,end/4:3*end/4-1) = kwrong; %将 k 放在中间

%再次显示差异
%仅作比较，显示两个变换的幅度：
figure; subplot(2,1,1);
imagesc(abs(kzf),[0 1e4]); colormap gray; axis equal;
subplot(2,1,2);
imagesc(abs(kzfwrong),[0 1e4]); colormap gray; axis equal;
%(imagesc的第二个参数设置颜色轴以使差异更明显)。
```

此时，结果相当平淡：

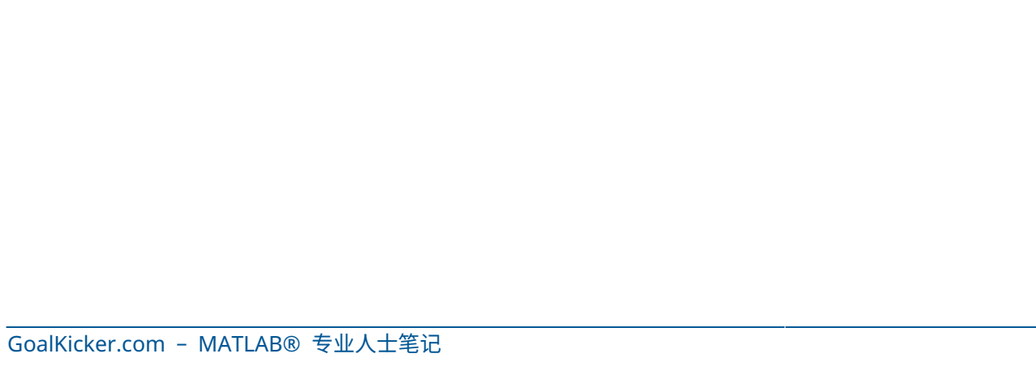




We now have obtained the 2D FT of an example image. To zero-fill it, we want to take each k-space, pad the edges with zeros, and then take the back transform:

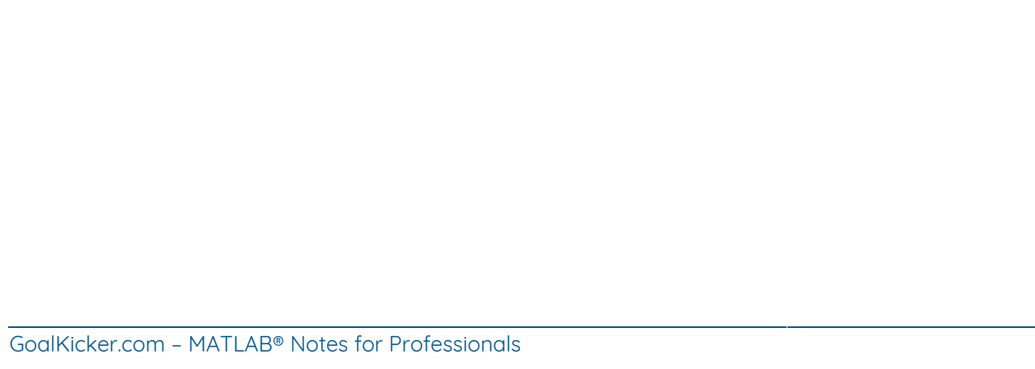
```
%Zero fill
kzf = zeros(imageSize .* 2); %Generate a  492x600 empty array to put the result in
kzf(end/4:3*end/4-1,end/4:3*end/4-1) = k; %Put k in the middle
kzfwrong = zeros(imageSize .* 2); %Generate a  492x600 empty array to put the result in
kzfwrong(end/4:3*end/4-1,end/4:3*end/4-1) = kwrong; %Put k in the middle

%Show the differences again
%Just for the sake of comparison, show the magnitude of both transforms:
figure; subplot(2,1,1);
imagesc(abs(kzf),[0 1e4]); colormap gray; axis equal;
subplot(2,1,2);
imagesc(abs(kzfwrong),[0 1e4]); colormap gray; axis equal;
%(The second argument to imagesc sets the colour axis to make the difference clear).
```

At this point, the result fairly unremarkable:



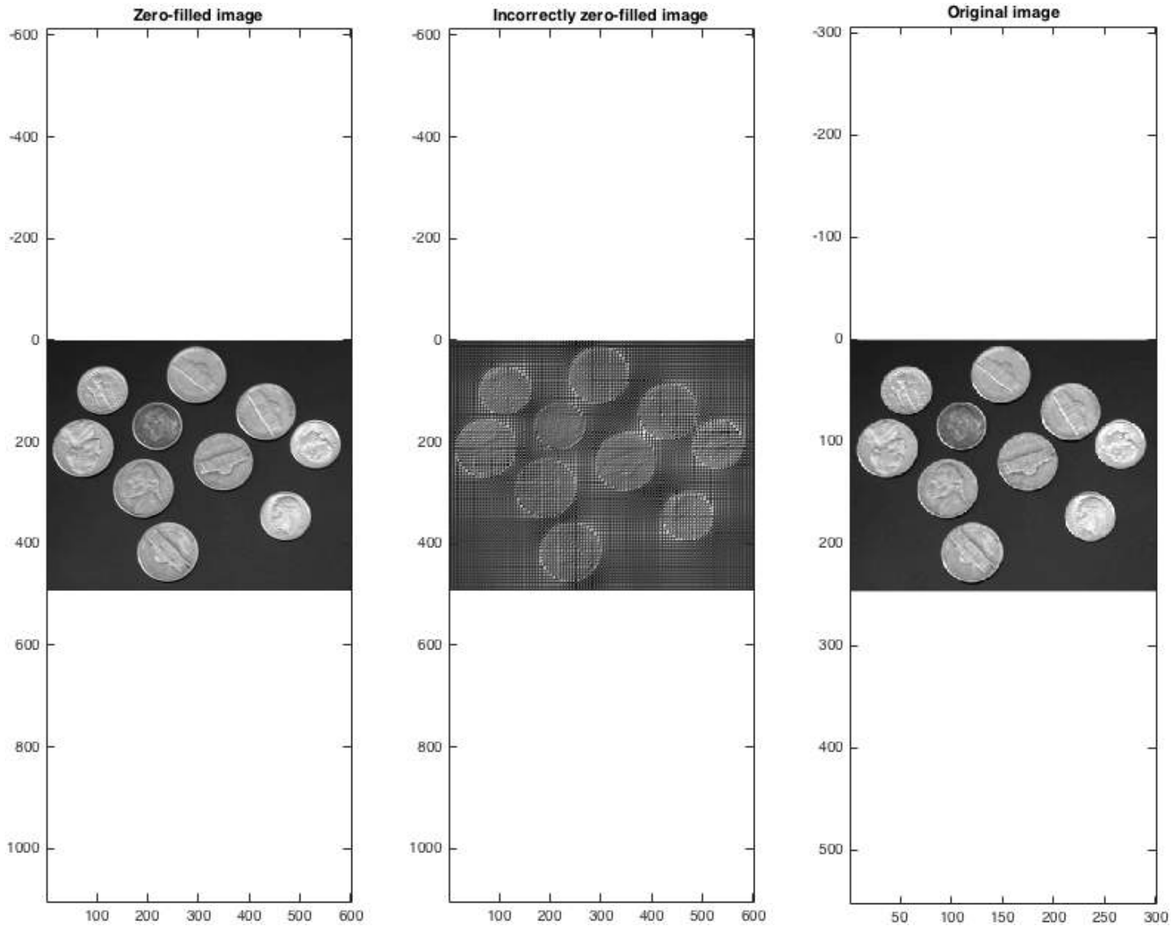
一旦我们进行反变换，就可以看到（正确地！）零填充数据提供了一种合理的插值方法：

```
% 进行反变换并查看
Izf = fftshift(ifft2(ifftshift(kzf)));
Izfwrong = ifft2(kzfwrong);

figure; subplot(1,3,1);
imagesc(abs(Izf)); colormap gray; axis equal;
title('零填充图像');
subplot(1,3,2);
imagesc(abs(Izfwrong)); colormap gray; axis equal;
title('错误零填充图像');
subplot(1,3,3);
imagesc(I); colormap gray; axis equal;
title('原始图像');
set(gcf,'color','w');
```

Once we then take the back-transforms, we can see that (correctly!) zero-filling data provides a sensible method for interpolation:

```
% Take the back transform and view
Izf = fftshift(ifft2(ifftshift(kzf)));
Izfwrong = ifft2(kzfwrong);

figure; subplot(1,3,1);
imagesc(abs(Izf)); colormap gray; axis equal;
title('Zero-filled image');
subplot(1,3,2);
imagesc(abs(Izfwrong)); colormap gray; axis equal;
title('Incorrectly zero-filled image');
subplot(1,3,3);
imagesc(I); colormap gray; axis equal;
title('Original image');
set(gcf,'color','w');
```



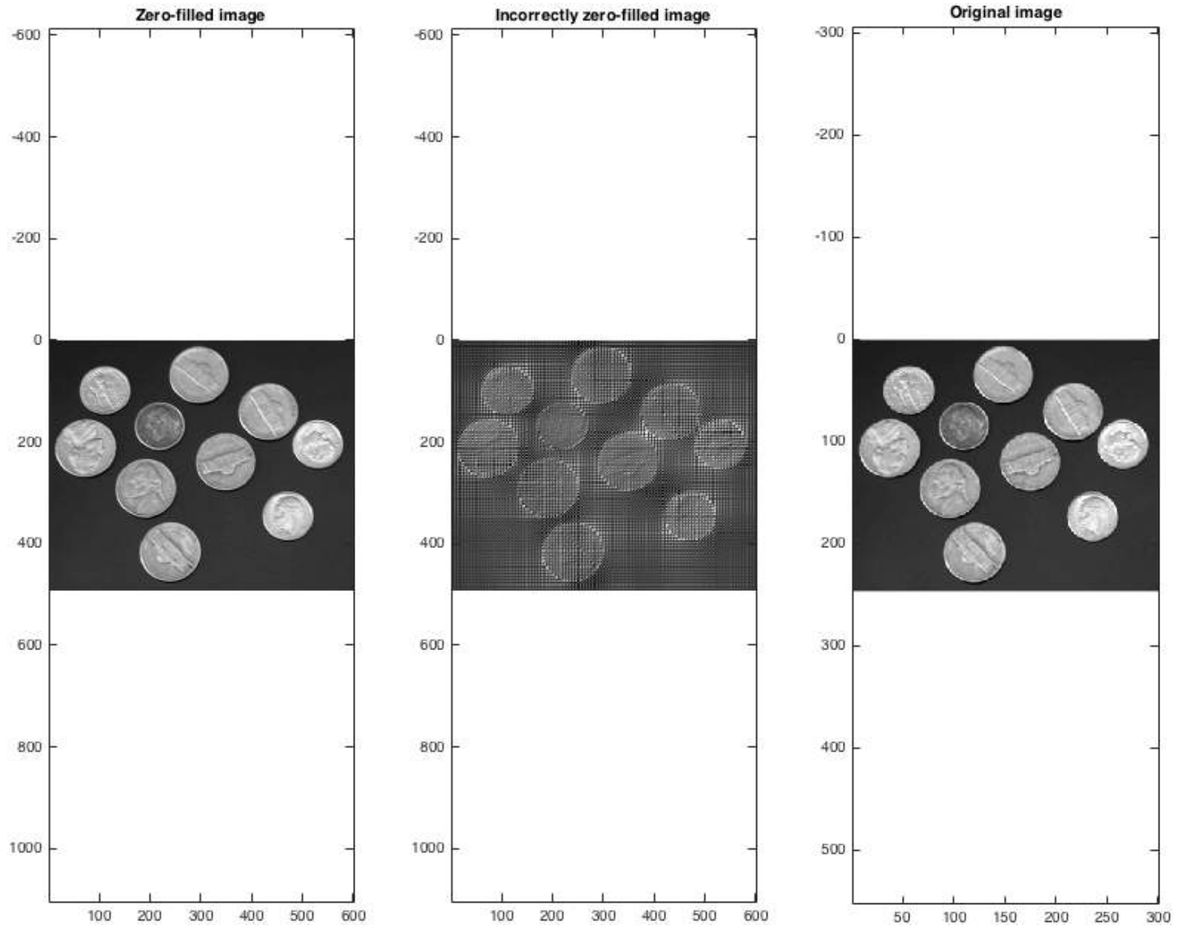
请注意，零填充后的图像尺寸是原始图像的两倍。可以在每个维度上进行超过两倍的零填充，尽管显然这样做并不会任意增加图像的大小。

提示、技巧、三维及更高维度

上述示例适用于三维图像（例如，通常由医学成像技术或共聚焦显微镜生成的图像），但需要将fft2替换为fftn（例如I，3）。由于多次编写fftshift（fft（fftshift（...））的过程较为繁琐，通常会在局部定义诸如fft2c的函数以提供更简便的语法——例如：

```
function y = fft2c(x)

y = fftshift(fft2(fftshift(x)));
```

请注意，FFT运算速度很快，但大型多维傅里叶变换在现代计算机上仍然需要一定时间。此外，傅里叶变换本质上是复数的：上文展示了k空间的幅度，但相位绝对关键；图像域中的平移等价于傅里叶域中的相位斜坡。傅里叶域中还可以进行许多更复杂的操作，例如通过乘以滤波器来滤除高频或低频空间频率，或屏蔽对应噪声的离散点。相应地，MATLAB主社区资源库——File Exchange上有大量社区生成的代码，用于处理常见的傅里叶操作。



Note that the zero-filled image size is double that of the original. One can zero fill by more than a factor of two in each dimension, although obviously doing so does not arbitrarily increase the size of an image.

Hints, tips, 3D and beyond

The above example holds for 3D images (as are often generated by medical imaging techniques or confocal microscopy, for example), but require `fft2` to be replaced by `fftn(I, 3)`, for example. Due to the somewhat cumbersome nature of writing `fftshift(fft(fftshift(...` several times, it is quite common to define functions such as `fft2c` locally to provide easier syntax locally -- such as:

```
function y = fft2c(x)

y = fftshift(fft2(fftshift(x)));
```

Note that the FFT is fast, but large, multidimensional Fourier transforms will still take time on a modern computer. It is additionally inherently complex: the magnitude of k-space was shown above, but the phase is absolutely vital; translations in the image domain are equivalent to a phase ramp in the Fourier domain. There are several far more complex operations that one may wish to do in the Fourier domain, such as filtering high or low spatial frequencies (by multiplying it with a filter), or masking out discrete points corresponding to noise. There is correspondingly a large quantity of community generated code for handling common Fourier operations available on MATLAB's main community repository site, the File Exchange.

第18.3节：逆傅里叶变换

傅里叶变换的一个主要优点是能够无损地逆变换回时域。让我们考虑前面示例中使用的相同信号：

```
A1=10;                % 振幅1
A2=10;                % 振幅2
w1=2*pi*0.2;          % 角频率1
w2=2*pi*0.225;        % 角频率2
Ts=1;                 % 采样时间
N=64;                 % 生成的处理样本数量
K=1;                  % 独立处理实现的数量
sgm=1;                % 噪声的标准差
n=repmat([0:N-1].',1,K);              % 生成分辨率
phi1=repmat(rand(1,K)*2*pi,N,1);      % 随机相位矩阵1
phi2=repmat(rand(1,K)*2*pi,N,1);      % 随机相位矩阵2
x=A1*sin(w1*n*Ts+phi1)+A2*sin(w2*n*Ts+phi2)+sgm*randn(N,K);    % 结果信号

NFFT=256;             % FFT长度
F=fft(x,NFFT);       % 时域信号的FFT结果
```

如果我们在MATLAB中打开F，会发现它是一个复数矩阵，包含实部和虚部。根据定义，为了恢复原始的时域信号，我们需要实部（表示幅度变化）和虚部（表示相位变化），因此要返回时域，可以简单地：

```
TD = ifft(F,NFFT);    % 返回F的时域逆变换
```

这里注意，返回的TD长度为256，因为我们将NFFT设置为256，但x的长度只有64，所以MATLAB会在TD变换的末尾补零。例如，如果NFFT是1024且长度是64，那么返回的TD将是64加上960个零。还要注意，由于浮点数舍入，你可能会得到类似3.1 * 10e-20的数值，但一般来说：对于任意X，ifft(fft(X))在舍入误差范围内等于X。

假设在变换后，我们做了一些操作，只剩下FFT的实部：

```
R = real(F);          % 获取FFT的实部
TDR = ifft(R,NFFT);   % 获取FFT实部的时域信号
```

这意味着我们丢失了FFT的虚部，因此在逆过程丢失了信息。为了不丢失信息地保留原始信号，应该始终保留FFT的虚部，使用imag，并将你的函数应用于实部和虚部或仅实部。

```
图
subplot(3,1,1)
plot(x);xlabel('时间采样点');ylabel('幅度');title('原始时域信号')
subplot(3,1,2)
plot(TD(1:64));xlabel('时间采样点');ylabel('幅度');title('逆傅里叶变换后的时域信号')

subplot(3,1,3)
plot(TDR(1:64));xlabel('时间采样');ylabel('幅度');title('IFFT变换的实部
时域信号')
```


Section 18.3: Inverse Fourier Transforms

One of the major benefit of Fourier Transform is its ability to inverse back in to the Time Domain without losing information. Let us consider the same Signal we used in the previous example:

```
A1=10;                % Amplitude 1
A2=10;                % Amplitude 2
w1=2*pi*0.2;          % Angular frequency 1
w2=2*pi*0.225;        % Angular frequency 2
Ts=1;                 % Sampling time
N=64;                 % Number of process samples to be generated
K=1;                  % Number of independent process realizations
sgm=1;                % Standard deviation of the noise
n=repmat([0:N-1].',1,K);              % Generate resolution
phi1=repmat(rand(1,K)*2*pi,N,1);      % Random phase matrix 1
phi2=repmat(rand(1,K)*2*pi,N,1);      % Random phase matrix 2
x=A1*sin(w1*n*Ts+phi1)+A2*sin(w2*n*Ts+phi2)+sgm*randn(N,K);    % Resulting Signal

NFFT=256;             % FFT length
F=fft(x,NFFT);        % FFT result of time domain signal
```

If we open F in MATLAB, we will find that it is a matrix of complex numbers, a real part and an imaginary part. By definition, in order to recover the original Time Domain signal, we need both the Real (which represents Magnitude variation) and the Imaginary (which represents Phase variation), so to return to the Time Domain, one may simply want to:

```
TD = ifft(F,NFFT);    %Returns the Inverse of F in Time Domain
```

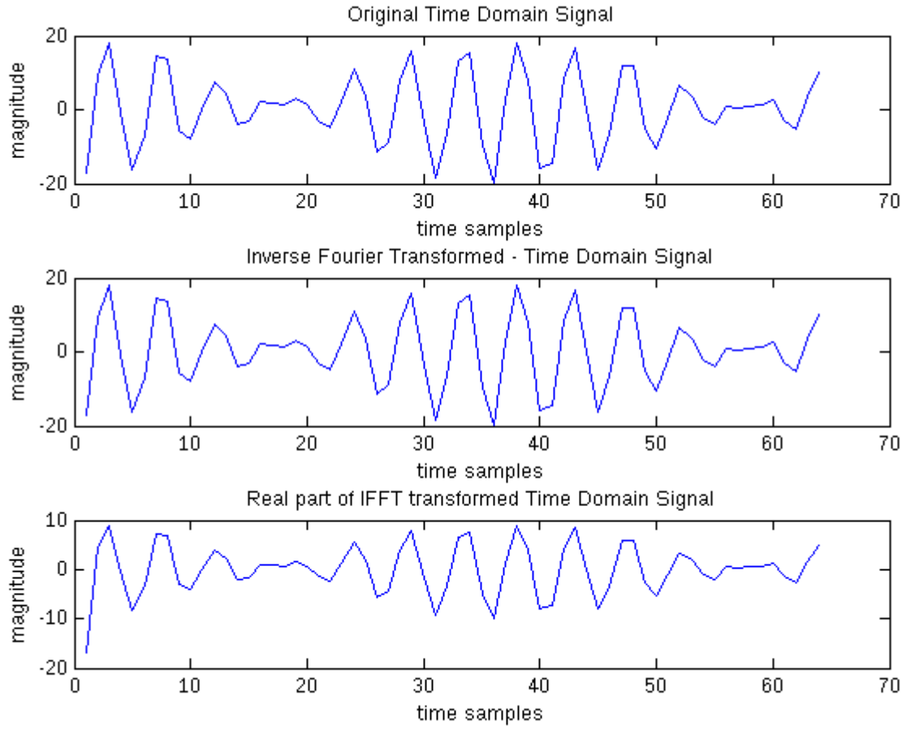
Note here that TD returned would be length 256 because we set NFFT to 256, however, the length of x is only 64, so MATLAB will pad zeros to the end of the TD transform. So for example, if NFFT was 1024 and the length was 64, then TD returned will be 64 + 960 zeros. Also note that due to floating point rounding, you might get something like 3.1 * 10e-20 but for general purposed: For any X, ifft(fft(X)) equals X to within roundoff error.

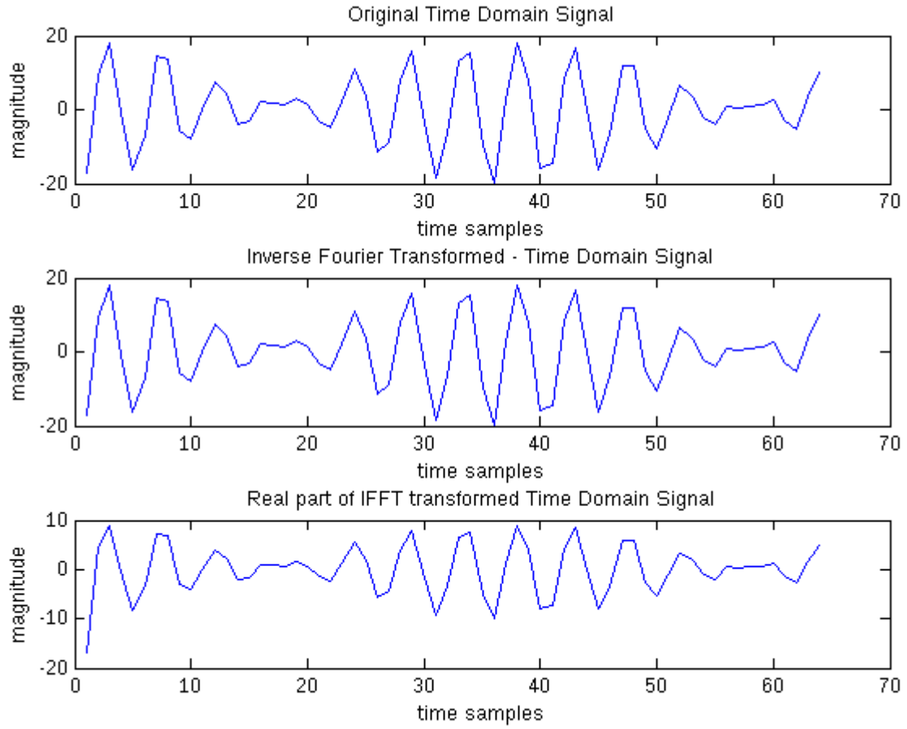
Let us say for a moment that after the transformation, we did something and are only left with the REAL part of the FFT:

```
R = real(F);          %Give the Real Part of the FFT
TDR = ifft(R,NFFT);   %Give the Time Domain of the Real Part of the FFT
```

This means that we are losing the imaginary part of our FFT, and therefore, we are losing information in this reverse process. To preserve the original without losing information, you should always keep the imaginary part of the FFT using `imag` and apply your functions to either both or the real part.

```
figure
subplot(3,1,1)
plot(x);xlabel('time samples');ylabel('magnitude');title('Original Time Domain Signal')
subplot(3,1,2)
plot(TD(1:64));xlabel('time samples');ylabel('magnitude');title('Inverse Fourier Transformed - Time
Domain Signal')
subplot(3,1,3)
plot(TDR(1:64));xlabel('time samples');ylabel('magnitude');title('Real part of IFFT transformed
Time Domain Signal')
```


Original Time Domain Signal
magnitude
time samples
Inverse Fourier Transformed - Time Domain Signal
magnitude
time samples
Real part of IFFT transformed Time Domain Signal
magnitude
time samples


Original Time Domain Signal
magnitude
time samples
Inverse Fourier Transformed - Time Domain Signal
magnitude
time samples
Real part of IFFT transformed Time Domain Signal
magnitude
time samples

第19章：常微分方程（ODE）求解器

第19.1节：odeset示例

首先我们初始化要解决的初值问题。

```
odefun = @(t,y) cos(y).^2*sin(t);
tspan = [0 16*pi];
y0=1;
```

然后我们使用ode45函数，在没有指定任何选项的情况下求解该问题。为了后续比较，我们绘制轨迹。

```
[t,y] = ode45(odefun, tspan, y0);
plot(t,y,'-o');
```

现在我们为问题设置较严格的相对和绝对容差限制。

```
options = odeset('RelTol',1e-2,'AbsTol',1e-2);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

我们设置了严格的相对误差和较窄的绝对误差容限。

```
options = odeset('RelTol',1e-7,'AbsTol',1e-2);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

我们设置了较窄的相对误差和严格的绝对误差容限。正如前面示例中所见，在误差容限较窄的情况下，轨迹与未使用任何特定选项的第一个图完全不同。

```
options = odeset('RelTol',1e-2,'AbsTol',1e-7);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

我们设置了严格的相对误差和严格的绝对误差容限。将结果与其他图进行比较，将突出显示在使用较窄误差容限计算时产生的误差。

```
options = odeset('RelTol',1e-7,'AbsTol',1e-7);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

以下内容应展示精度与运行时间之间的权衡。

```
tic;
options = odeset('RelTol',1e-7,'AbsTol',1e-7);
[t,y] = ode45(odefun, tspan, y0, options);
time1 = toc;
plot(t,y,'-o');
```

为了比较，我们收紧了绝对误差和相对误差的容限限制。现在我们可以看到，如果没有大幅提高精度，解决我们的初值问题将花费更长时间。

Chapter 19: Ordinary Differential Equations (ODE) Solvers

Section 19.1: Example for odeset

First we initialize our initial value problem we want to solve.

```
odefun = @(t,y) cos(y).^2*sin(t);
tspan = [0 16*pi];
y0=1;
```

We then use the ode45 function without any specified options to solve this problem. To compare it later we plot the trajectory.

```
[t,y] = ode45(odefun, tspan, y0);
plot(t,y,'-o');
```

We now set a narrow relative and a narrow absolute limit of tolerance for our problem.

```
options = odeset('RelTol',1e-2,'AbsTol',1e-2);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

We set tight relative and narrow absolute limit of tolerance.

```
options = odeset('RelTol',1e-7,'AbsTol',1e-2);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

We set narrow relative and tight absolute limit of tolerance. As in the previous examples with narrow limits of tolerance one sees the trajectory being completely different from the first plot without any specific options.

```
options = odeset('RelTol',1e-2,'AbsTol',1e-7);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

We set tight relative and tight absolute limit of tolerance. Comparing the result with the other plot will underline the errors made calculating with narrow tolerance limits.

```
options = odeset('RelTol',1e-7,'AbsTol',1e-7);
[t,y] = ode45(odefun, tspan, y0, options);
plot(t,y,'-o');
```

The following should demonstrate the trade-off between precision and run-time.

```
tic;
options = odeset('RelTol',1e-7,'AbsTol',1e-7);
[t,y] = ode45(odefun, tspan, y0, options);
time1 = toc;
plot(t,y,'-o');
```

For comparison we tighten the limit of tolerance for absolute and relative error. We now can see that without large gain in precision it will take considerably longer to solve our initial value problem.

```
tic;
options = odeset('RelTol',1e-13,'AbsTol',1e-13);
[t,y] = ode45(odefun, tspan, y0, options);
time2 = toc;
plot(t,y,'-o');
```



```
tic;
options = odeset('RelTol',1e-13,'AbsTol',1e-13);
[t,y] = ode45(odefun, tspan, y0, options);
time2 = toc;
plot(t,y,'-o');
```

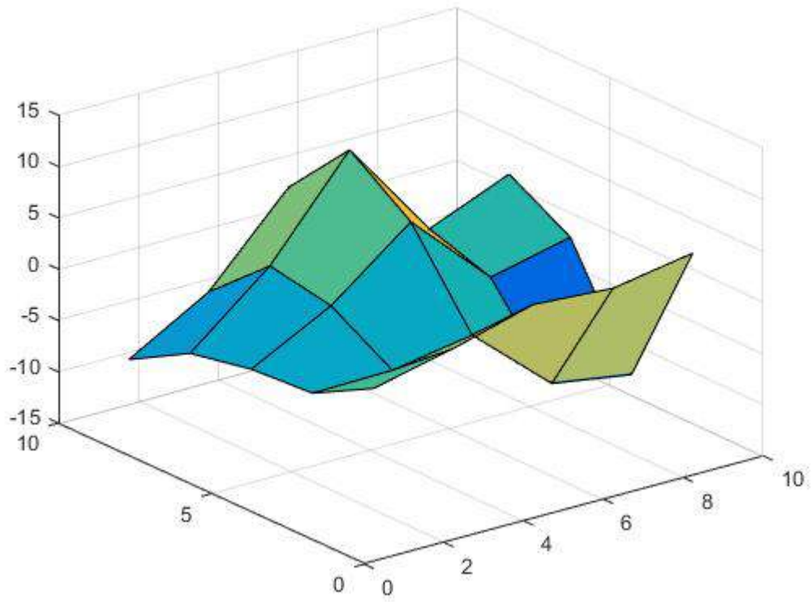
第20章：使用MATLAB进行插值

第20.1节：二维分段插值

我们初始化数据：

```
[X,Y] = meshgrid(1:2:10);
Z = X.*cos(Y) - Y.*sin(X);
```

该曲面如下所示。



现在我们设置想要插值的点：

```
[Vx,Vy] = meshgrid(1:0.25:10);
```

我们现在可以执行最近邻插值，

```
Vz = interp2(X,Y,Z,Vx,Vy,'nearest');
```

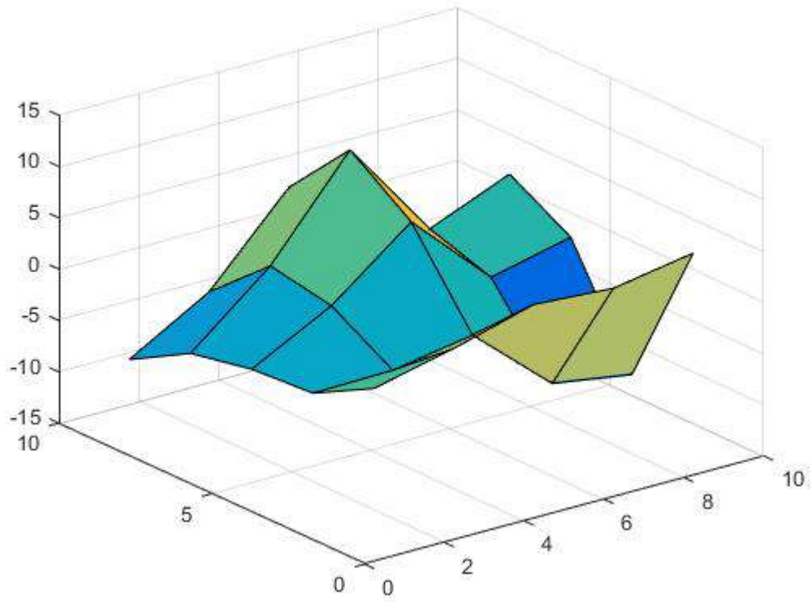

Chapter 20: Interpolation with MATLAB

Section 20.1: Piecewise interpolation 2 dimensional

We initialize the data:

```
[X,Y] = meshgrid(1:2:10);
Z = X.*cos(Y) - Y.*sin(X);
```

The surface looks like the following.

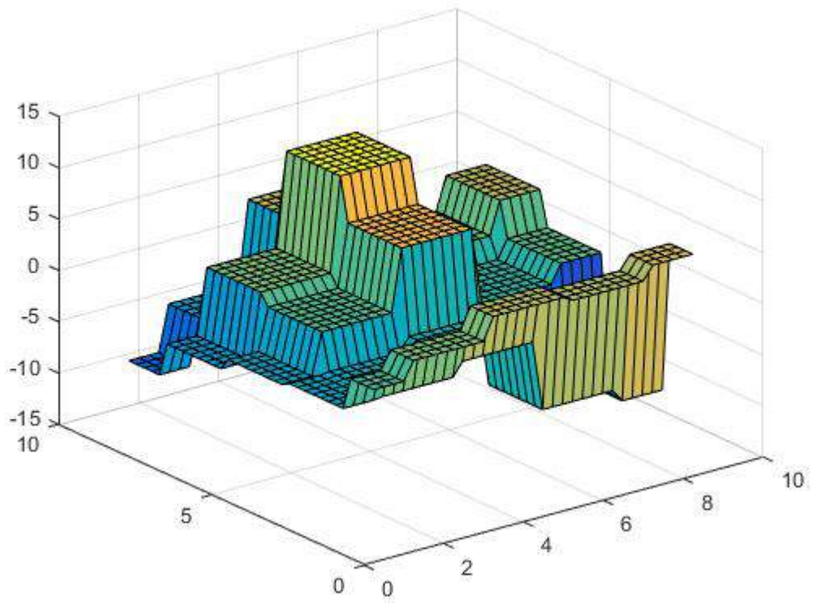


Now we set the points where we want to interpolate:

```
[Vx,Vy] = meshgrid(1:0.25:10);
```

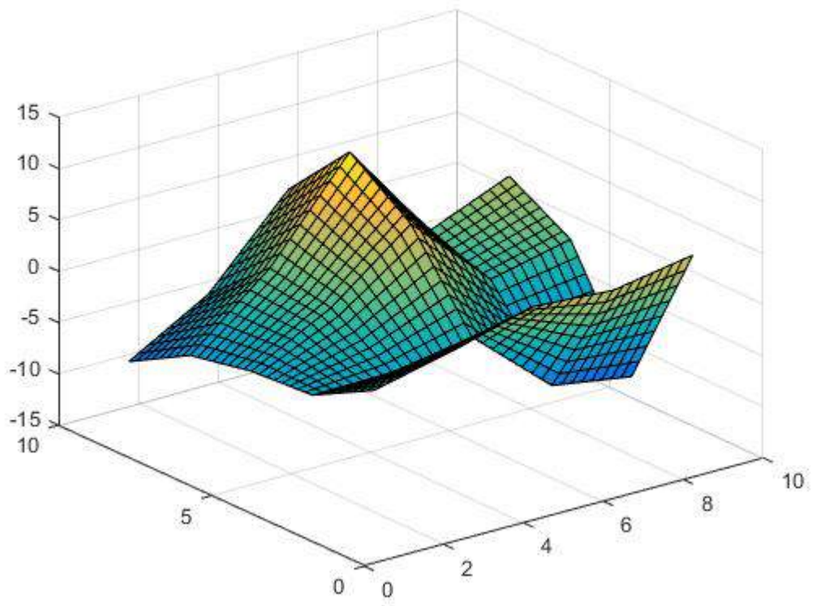
We now can perform nearest interpolation,

```
Vz = interp2(X,Y,Z,Vx,Vy,'nearest');
```



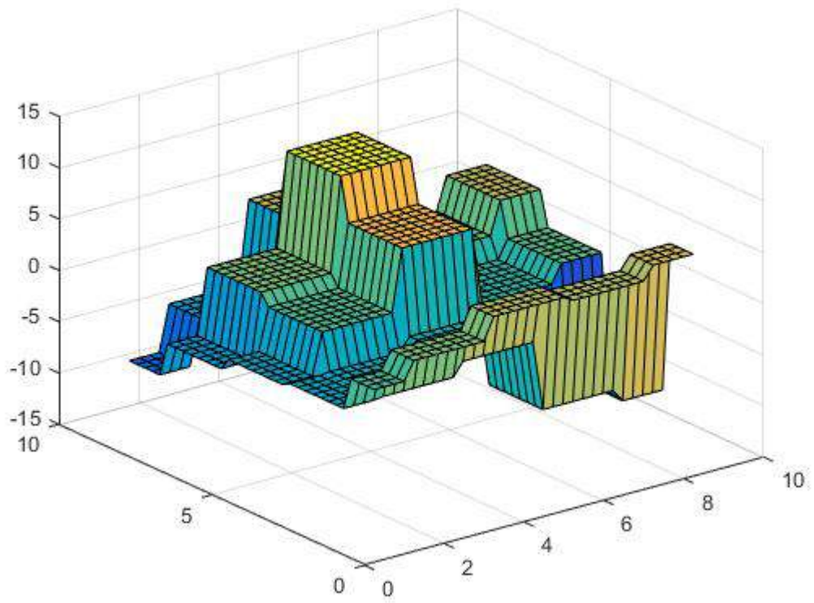
线性插值，

```
Vz = interp2(X,Y,Z,Vx,Vy,'linear');
```



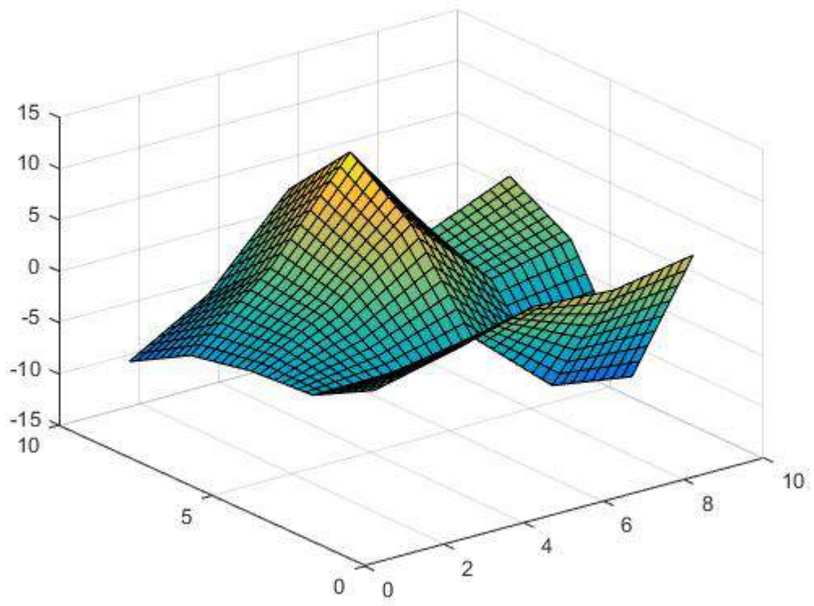
三次插值

```
Vz = interp2(X,Y,Z,Vx,Vy,'cubic');
```

linear interpolation,

```
Vz = interp2(X,Y,Z,Vx,Vy,'linear');
```

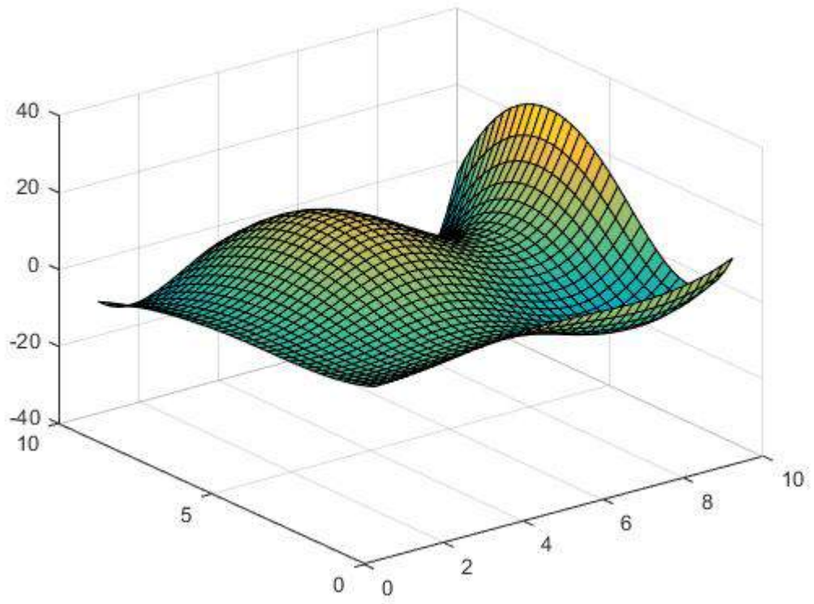


cubic interpolation

```
Vz = interp2(X,Y,Z,Vx,Vy,'cubic');
```

或样条插值：

```
Vz = interp2(X,Y,Z,Vx,Vy,'spline');
```



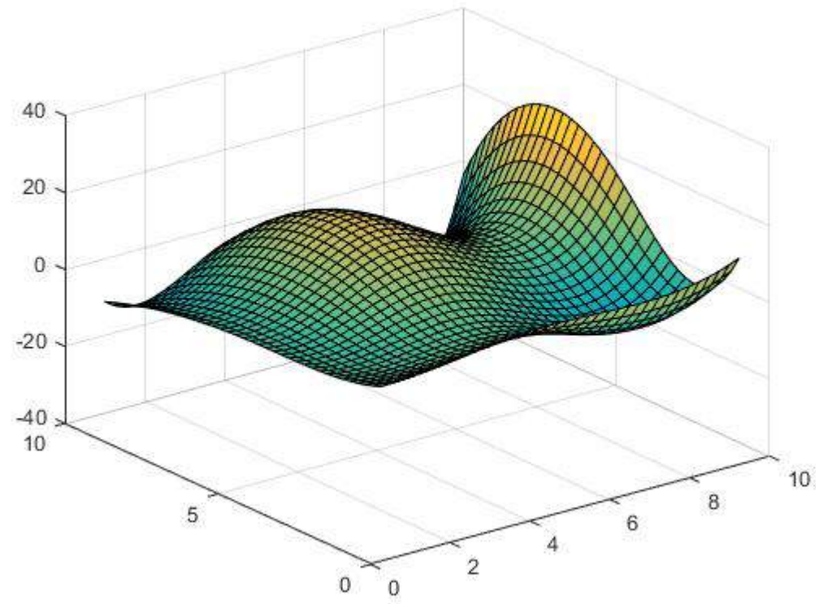
第20.2节：一维分段插值

我们将使用以下数据：

```
x = 1:5:50;
y = randi([-10 10],1,10);
```


or spline interpolation:

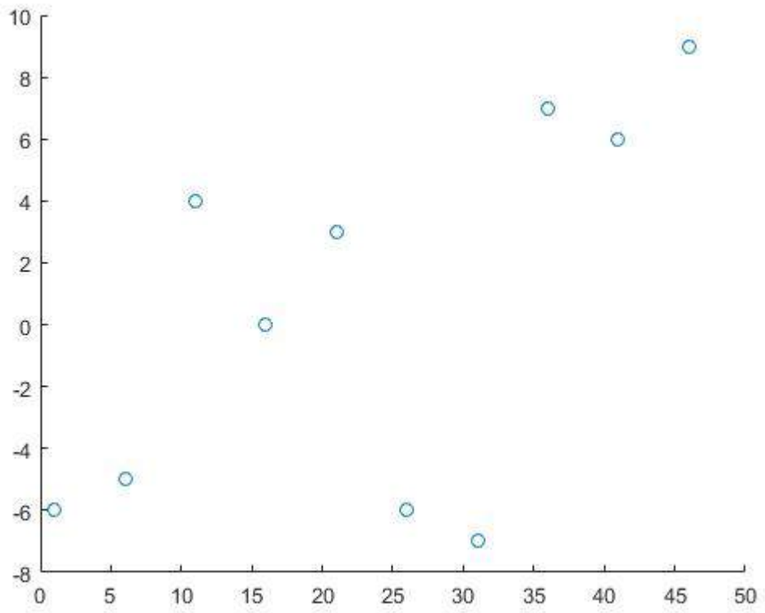
```
Vz = interp2(X,Y,Z,Vx,Vy,'spline');
```



Section 20.2: Piecewise interpolation 1 dimensional

We will use the following data:

```
x = 1:5:50;
y = randi([-10 10],1,10);
```

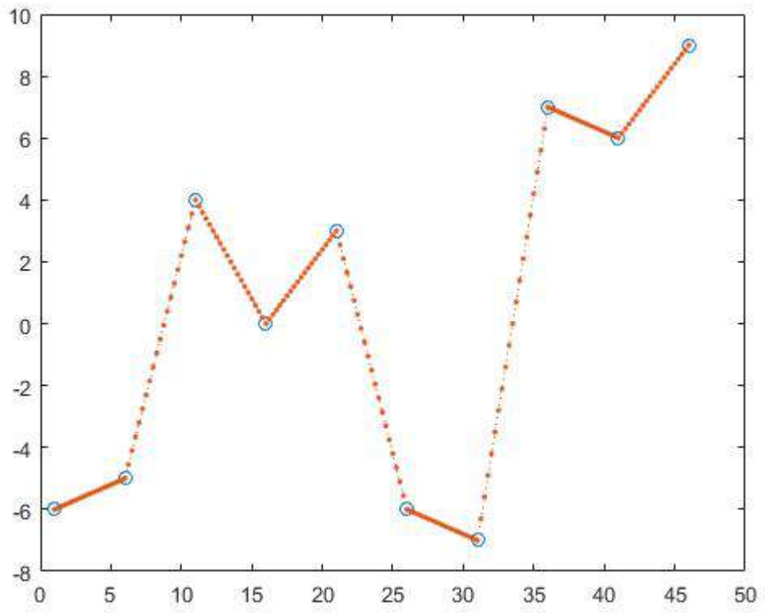


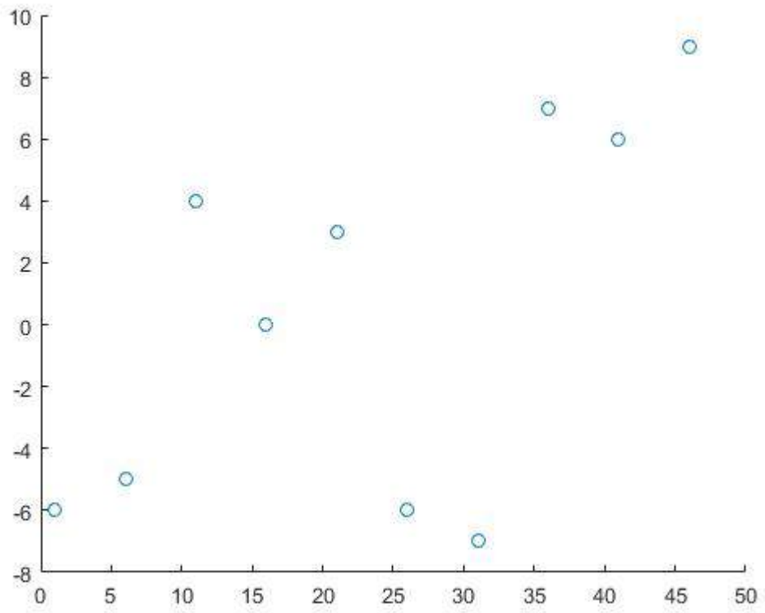
这里，x 和 y 是数据点的坐标，z 是我们需要获取信息的点。

```
z = 0:0.25:50;
```

一种找到 z 对应 y 值的方法是分段线性插值。

```
z_y = interp1(x,y,z,'linear');
```



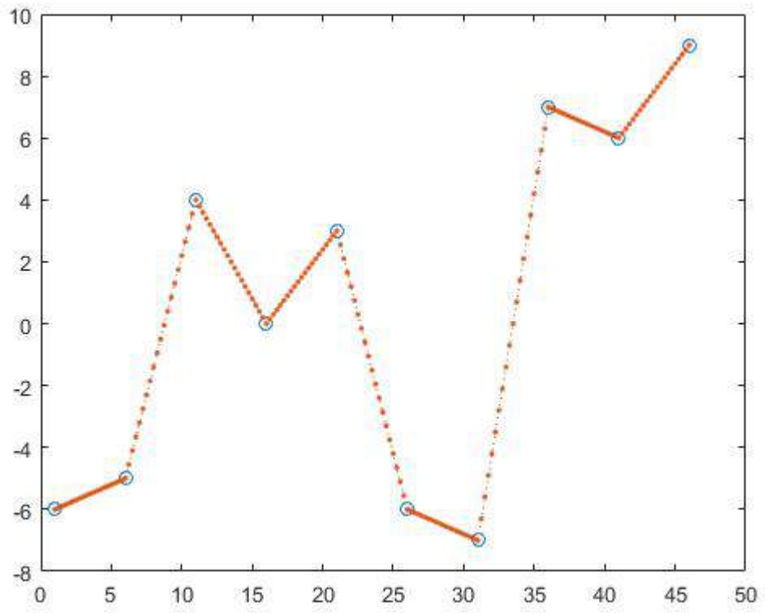


Hereby x and y are the coordinates of the data points and z are the points we need information about.

```
z = 0:0.25:50;
```

One way to find the y-values of z is piecewise linear interpolation.

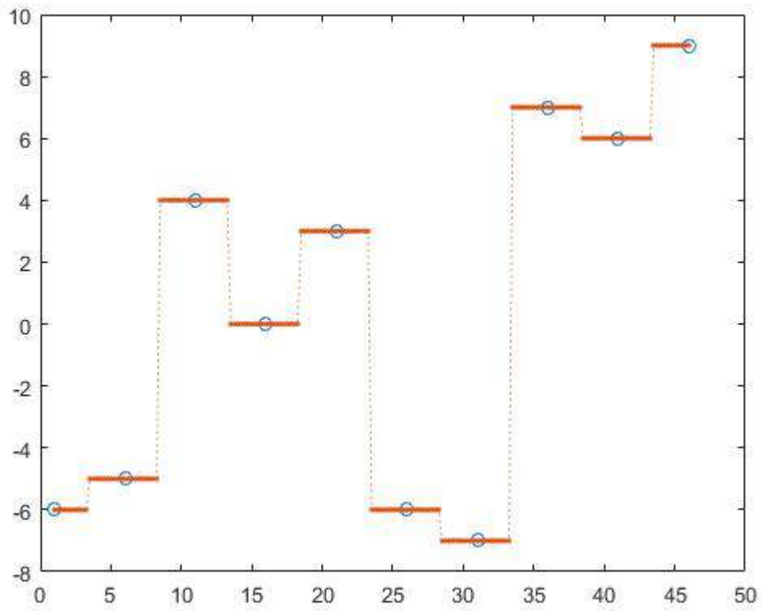
```
z_y = interp1(x,y,z,'linear');
```



这里通过计算两个相邻点之间的直线，并假设该点属于这些直线之一，从而得到 z_y。

interp1 还提供其他选项，如最近邻插值，

```
z_y = interp1(x,y,z, 'nearest');
```



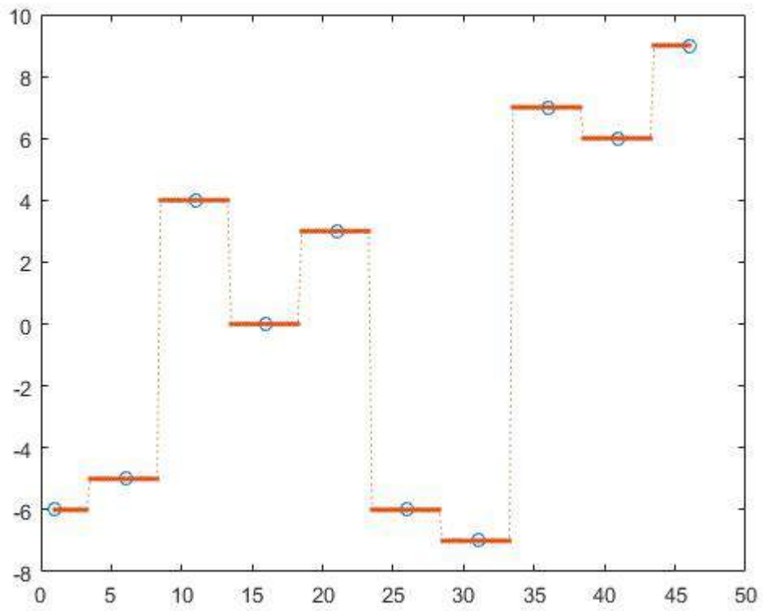
下一个插值，

```
z_y = interp1(x,y,z, 'next');
```


Hereby one calculates the line between two adjacent points and gets z_y by assuming that the point would be an element of those lines.

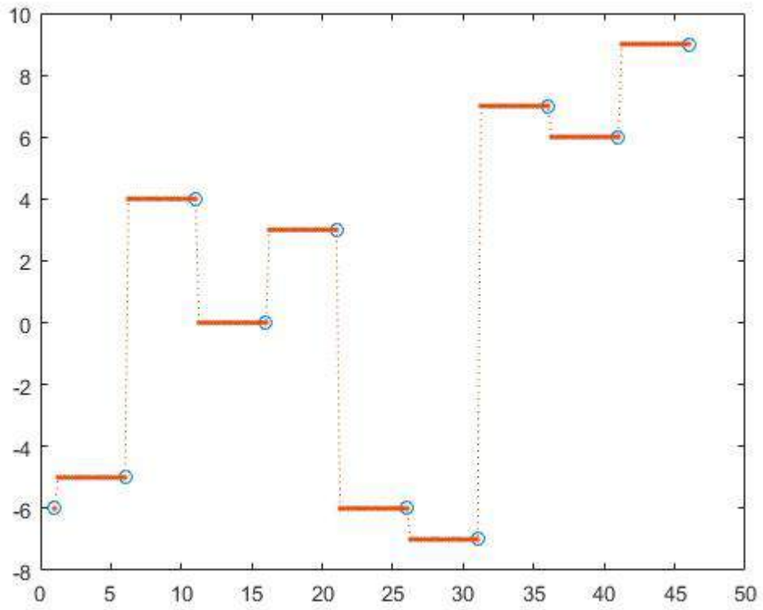
`interp1` provides other options too like nearest interpolation,

```
z_y = interp1(x,y,z, 'nearest');
```



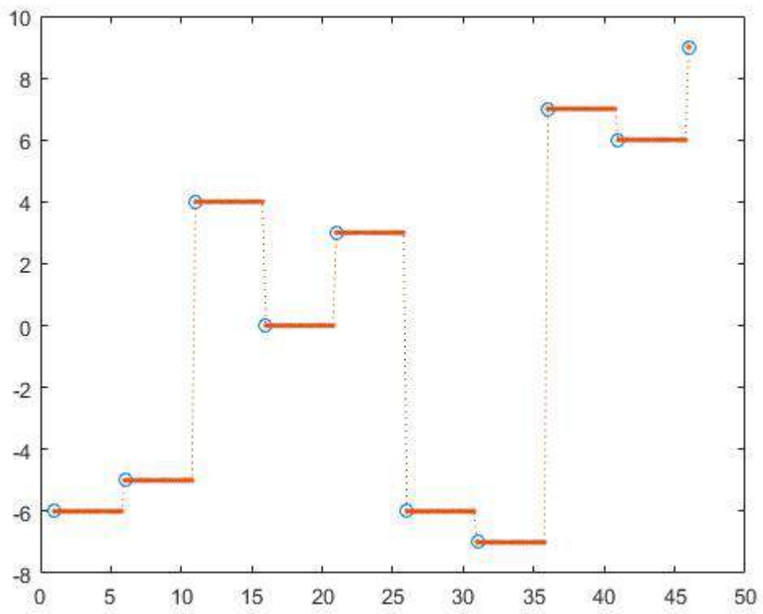
next interpolation,

```
z_y = interp1(x,y,z, 'next');
```



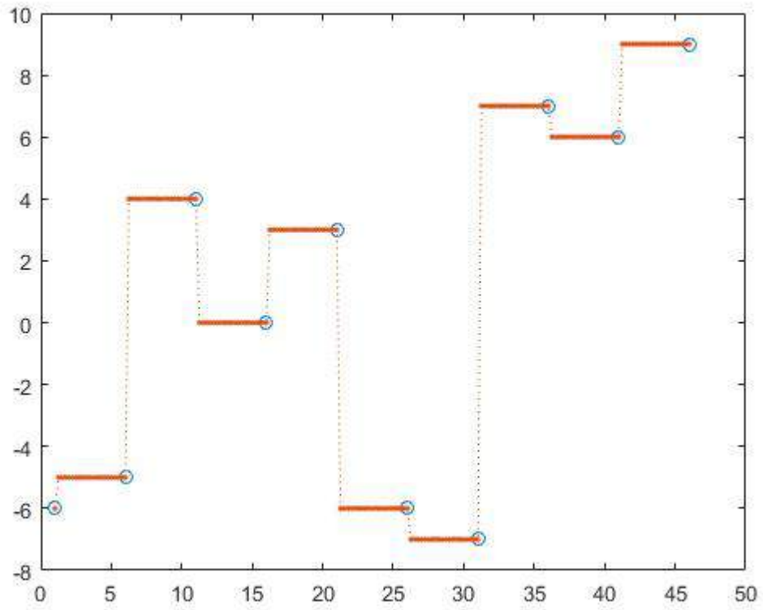
上一个插值，

```
z_y = interp1(x,y,z, 'previous');
```



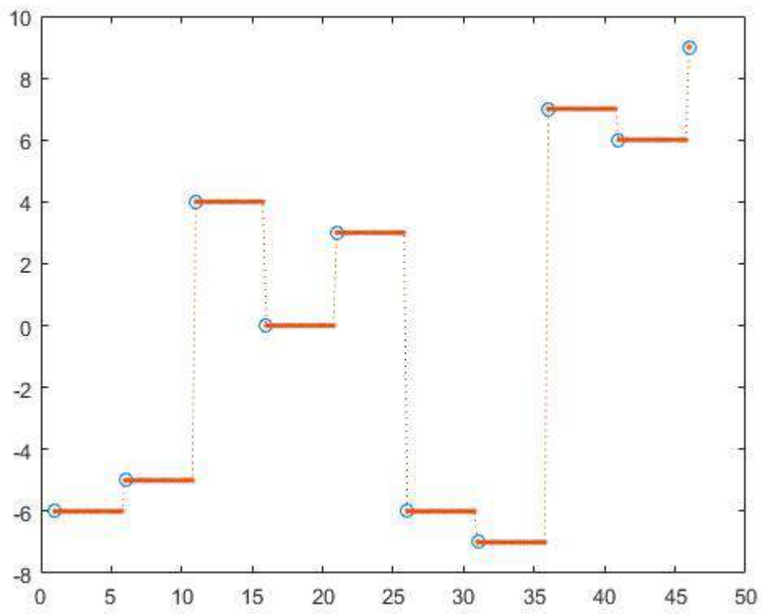
保持形状的分段三次插值，

```
z_y = interp1(x,y,z, 'pchip');
```

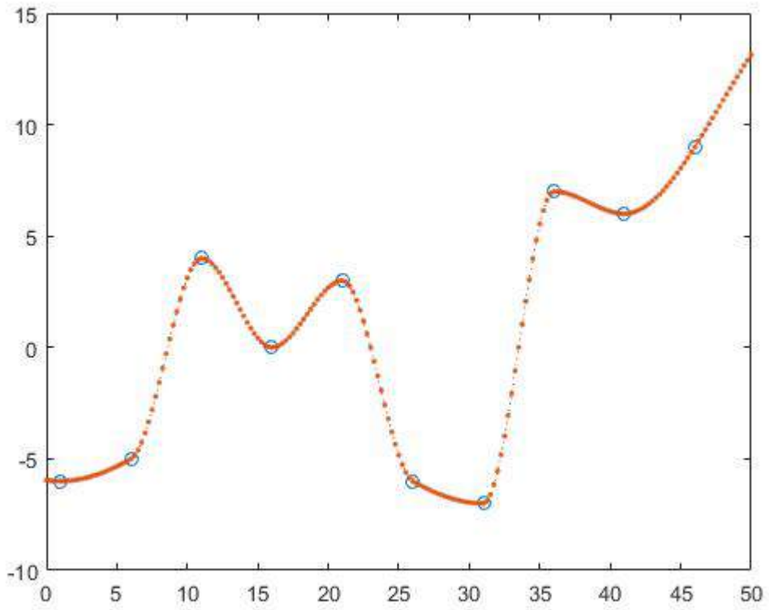
previous interpolation,

```
z_y = interp1(x,y,z, 'previous');
```

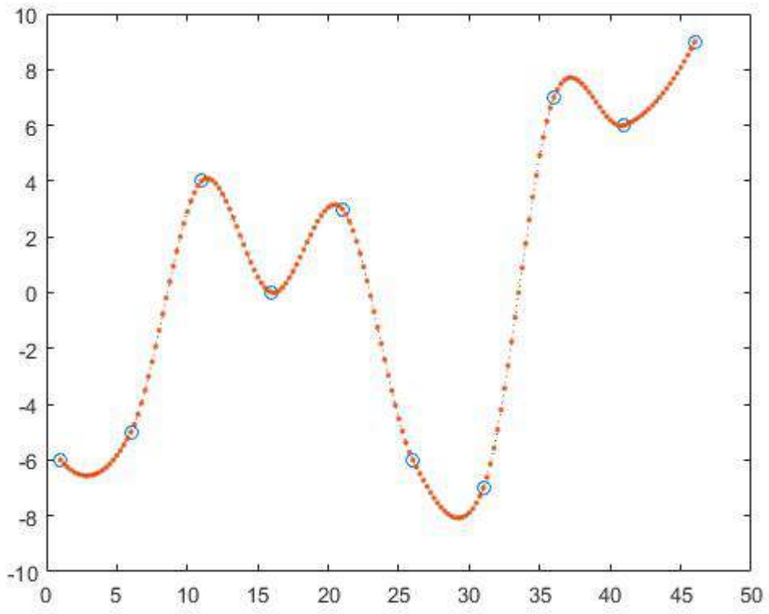


Shape-preserving piecewise cubic interpolation,

```
z_y = interp1(x,y,z, 'pchip');
```

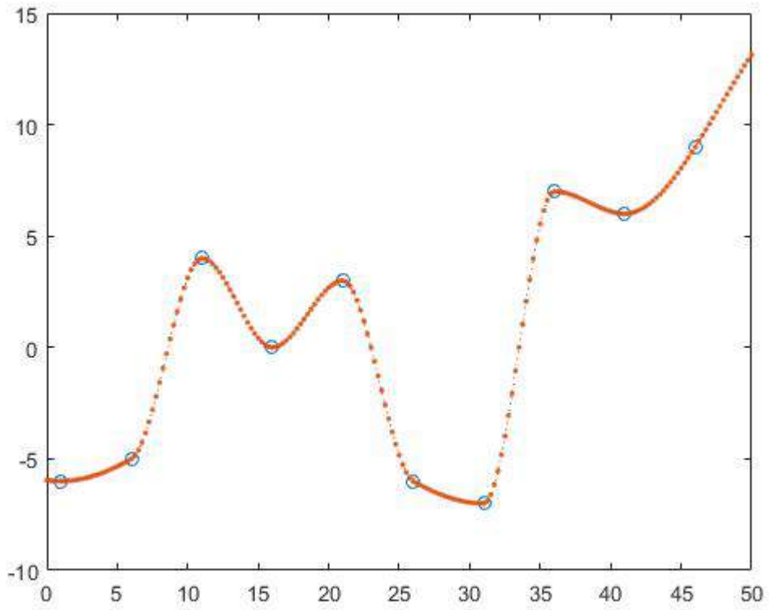


三次卷积，z_y = interp1(x,y,z, 'v5cubic');

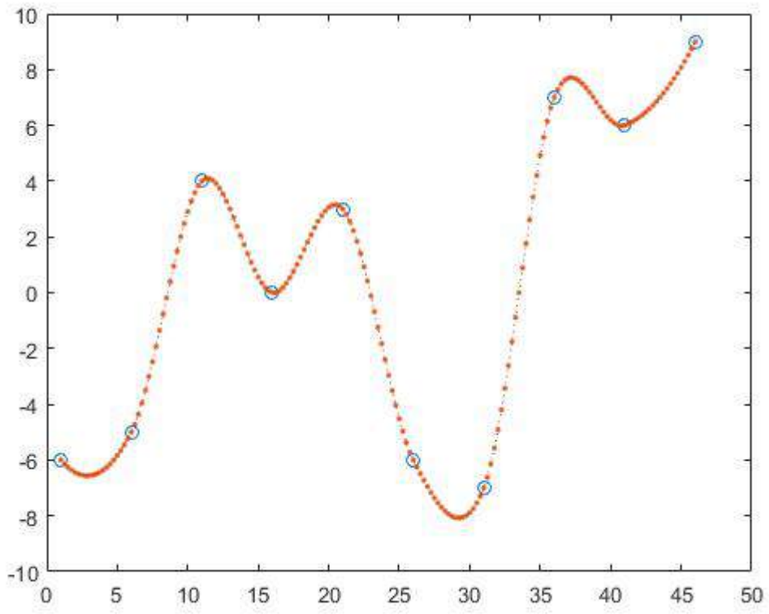


和样条插值

```
z_y = interp1(x,y,z, 'spline');
```

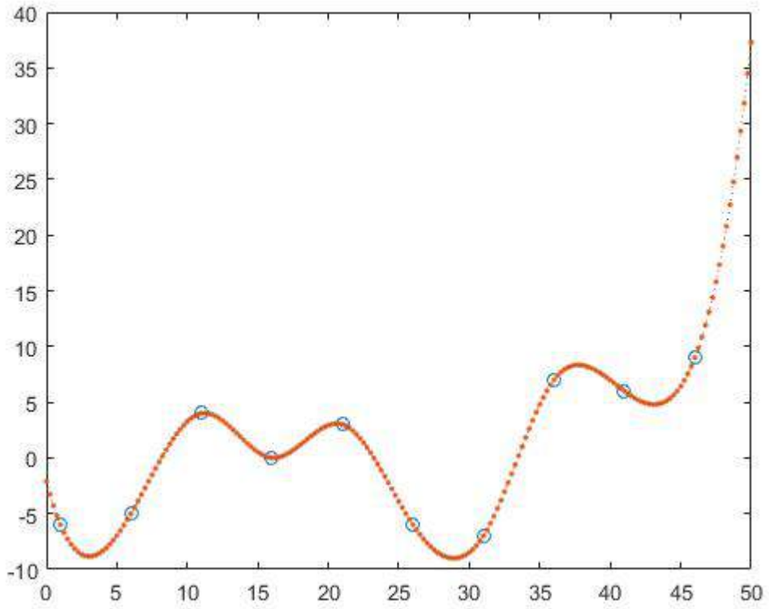



cubic convolution, z_y = interp1(x,y,z, 'v5cubic');



and spline interpolation

```
z_y = interp1(x,y,z, 'spline');
```



这里是最近、下一个和上一个分段常数插值。

第20.3节：多项式插值

我们初始化要插值的数据：

```
x = 0:0.5:10;
y = sin(x/2);
```

这意味着区间[0,10]内数据的基础函数是正弦函数。现在计算近似多项式的系数：

```
p1 = polyfit(x,y,1);
p2 = polyfit(x,y,2);
p3 = polyfit(x,y,3);
p5 = polyfit(x,y,5);
p10 = polyfit(x,y,10);
```

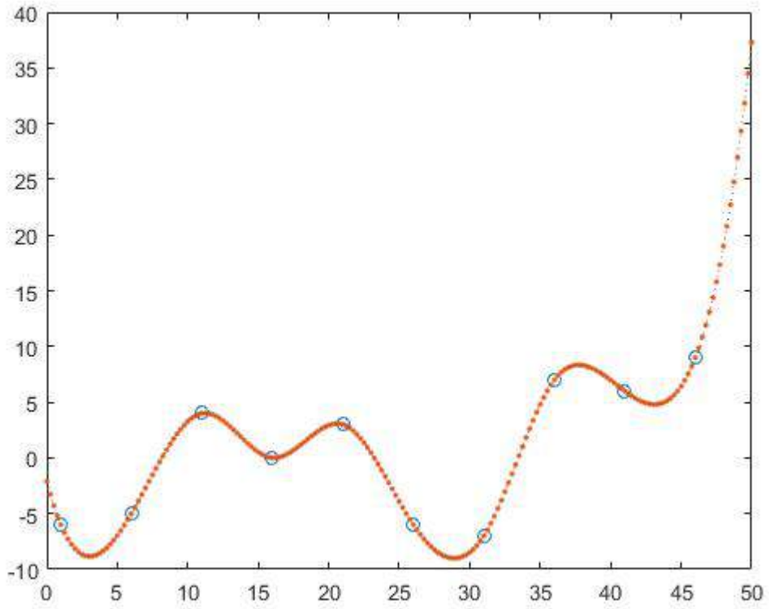
这里，x 是数据点的x值，y 是y值，第三个数字是多项式的阶数。现在我们设置计算插值函数的网格：

```
zx = 0:0.1:10;
```

并计算y值：

```
zy1 = polyval(p1,zx);
zy2 = polyval(p2,zx);
zy3 = polyval(p3,zx);
zy5 = polyval(p5,zx);
zy10 = polyval(p10,zx);
```

可以看出，随着多项式次数的增加，样本的近似误差变小。



Hereby are nearst, next and previous interpolation piecewise constant interpolations.

Section 20.3: Polynomial interpolation

We initialize the data we want to interpolate:

```
x = 0:0.5:10;
y = sin(x/2);
```

This means the underlying function for the data in the interval [0,10] is sinusoidal. Now the coefficients of the approximating polynomials are being calculated:

```
p1 = polyfit(x,y,1);
p2 = polyfit(x,y,2);
p3 = polyfit(x,y,3);
p5 = polyfit(x,y,5);
p10 = polyfit(x,y,10);
```

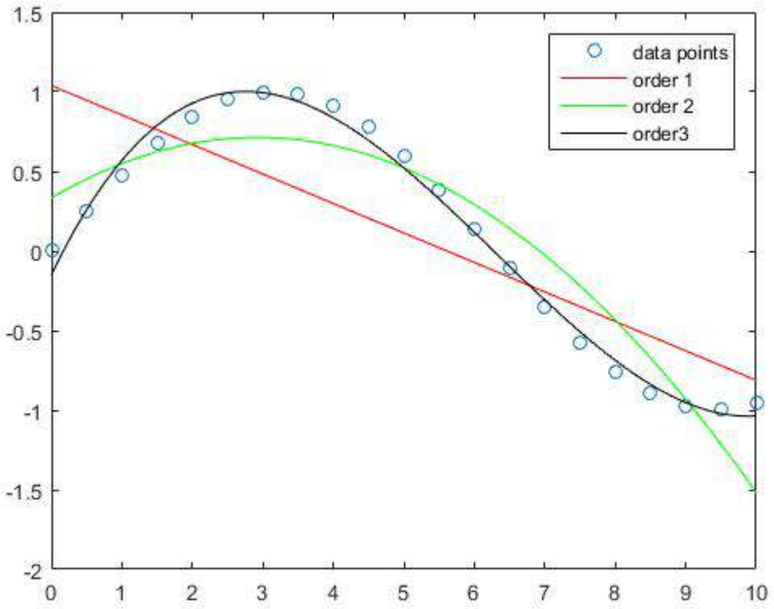
Hereby is x the x-value and y the y-value of our data points and the third number is the order/degree of the polynomial. We now set the grid we want to compute our interpolating function on:

```
zx = 0:0.1:10;
```

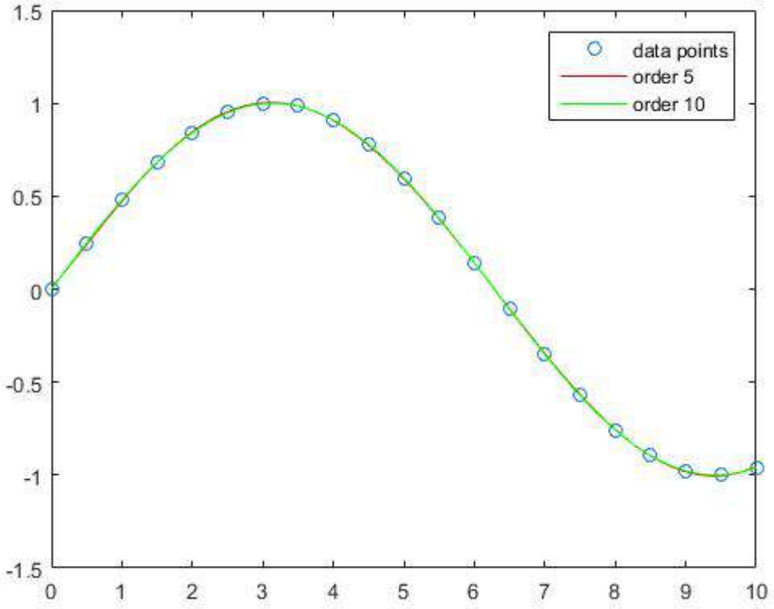
and calculate the y-values:

```
zy1 = polyval(p1,zx);
zy2 = polyval(p2,zx);
zy3 = polyval(p3,zx);
zy5 = polyval(p5,zx);
zy10 = polyval(p10,zx);
```

One can see that the approximation error for the sample gets smaller when the degree of the polynomial increases.

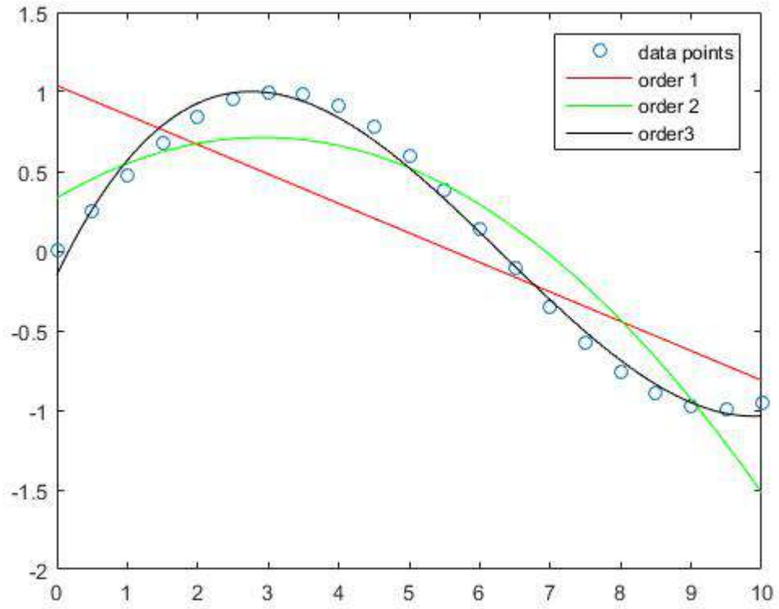


虽然本例中直线的近似误差较大，但三阶多项式在该区间内对正弦函数的近似相对较好。

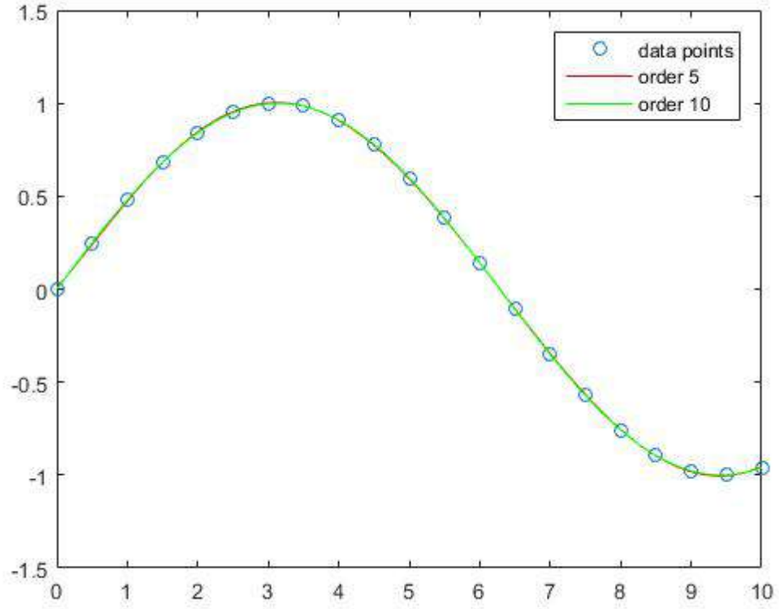


五阶和十阶多项式的插值几乎没有近似误差。

然而，如果考虑样本外的表现，可以看到次数过高往往会过拟合，因此在样本外表现较差。我们设定



While the approximation of the straight line in this example has larger errors the order 3 polynomial approximates the sinus function in this interval relatively good.



The interpolation with order 5 and order 10 polynomials has almost no approximation error.

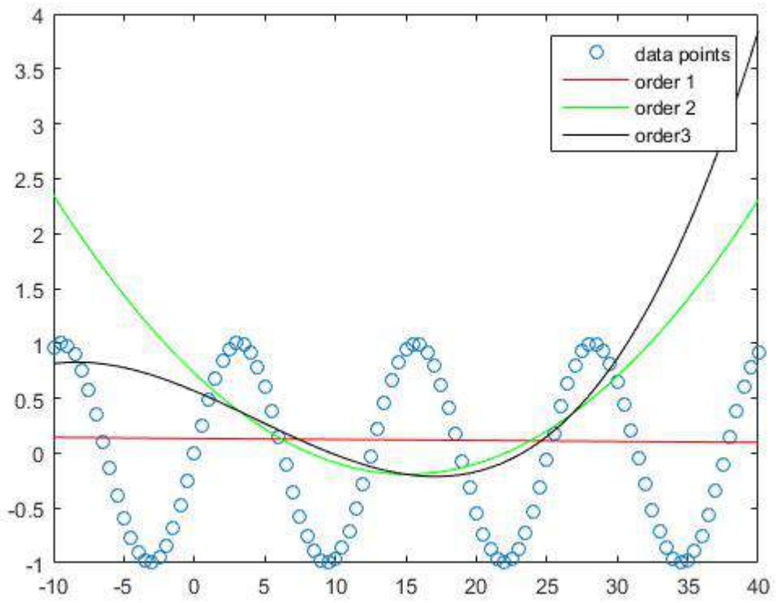
However if we consider the out of sample performance one sees that too high orders tend to overfit and therefore perform badly out of sample. We set

```
zx = -10:0.1:40;
p10 = polyfit(X,Y,10);
p20 = polyfit(X,Y,20);
```

和

```
zy10 = polyval(p10,zx);
zy20 = polyval(p20,zx);
```

如果我们看一下图表，可以看到样本外表现对于阶数1是最好的



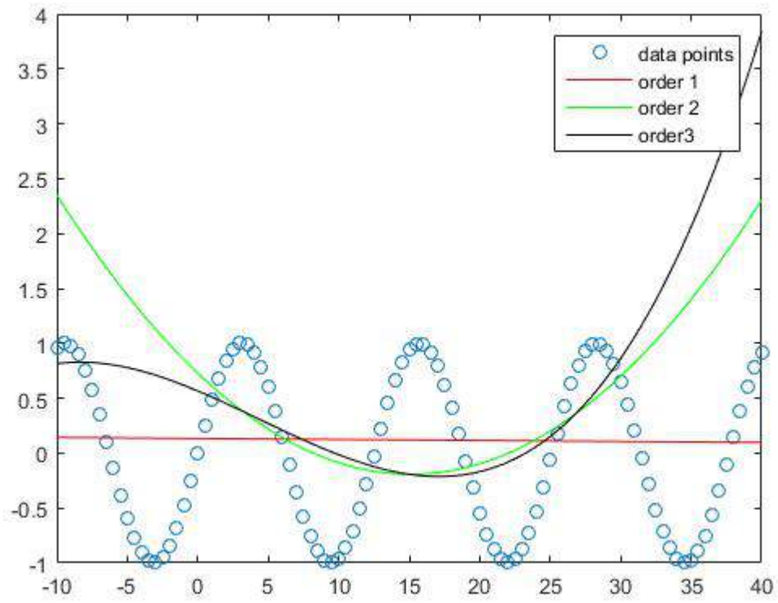
并且随着次数的增加情况越来越糟。


```
zx = -10:0.1:40;
p10 = polyfit(X,Y,10);
p20 = polyfit(X,Y,20);
```

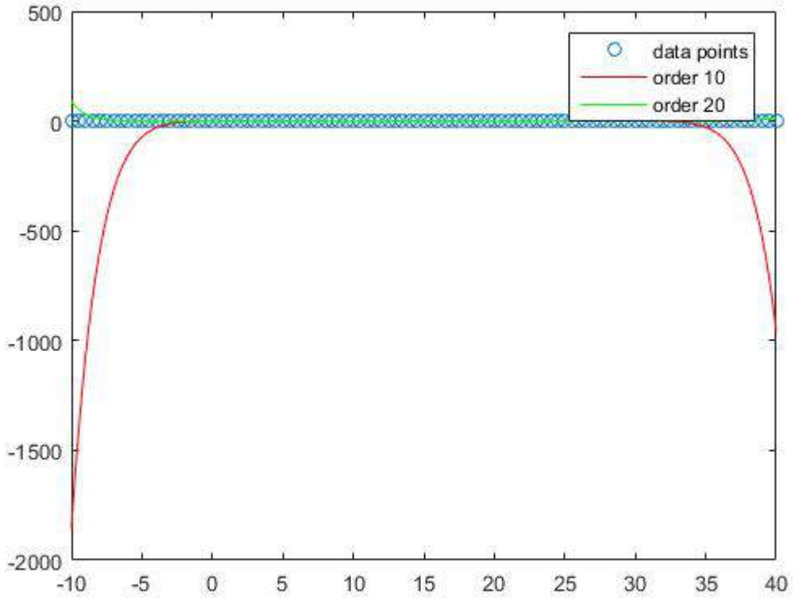
and

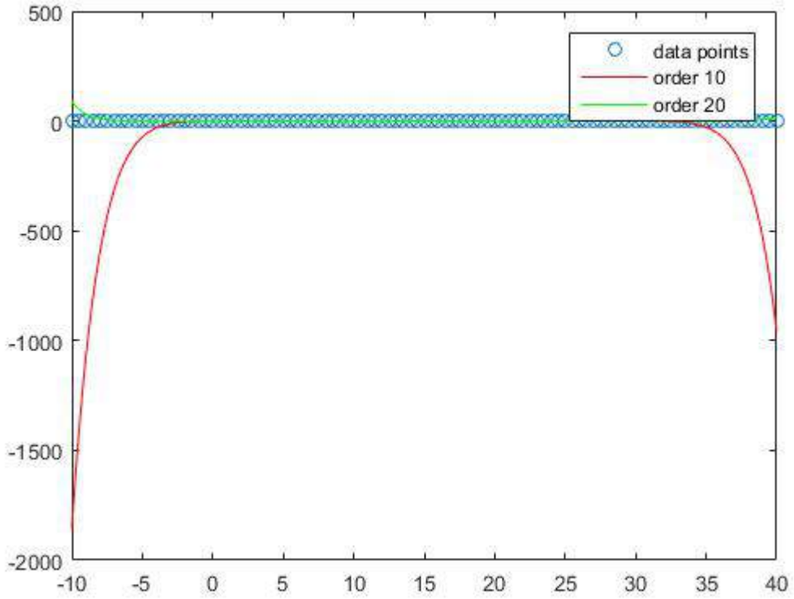
```
zy10 = polyval(p10,zx);
zy20 = polyval(p20,zx);
```

If we take a look at the plot we see that the out of sample performance is best for the order 1



and keeps getting worse with increasing degree.


data points
order 10
order 20
500
0
-500
-1000
-1500
-2000
-10
-5
0
5
10
15
20
25
30
35
40


data points
order 10
order 20
500
0
-500
-1000
-1500
-2000
-10
-5
0
5
10
15
20
25
30
35
40

第21章：积分

第21.1节：integral，integral2，integral3

一维

对一维函数进行积分

```
f = @(x) sin(x).^3 + 1;
```

在区间内

```
xmin = 2;
xmax = 8;
```

可以调用该函数

```
q = integral(f,xmin,xmax);
```

也可以设置相对误差和绝对误差的边界

```
q = integral(f,xmin,xmax, 'RelTol',10e-6, 'AbsTol',10-4);
```

二维

如果想对二维函数进行积分

```
f = @(x,y) sin(x).^y ;
```

在区间内

```
xmin = 2;
xmax = 8;
ymin = 1;
ymax = 4;
```

调用函数

```
q = integral2(f,xmin,xmax,ymin,ymax);
```

和另一种情况一样，也可以限制容差

```
q = integral2(f,xmin,xmax,ymin,ymax, 'RelTol',10e-6, 'AbsTol',10-4);
```

三维

对三维函数进行积分

```
f = @(x,y,z) sin(x).^y - cos(z) ;
```

在区间内

```
xmin = 2;
xmax = 8;
```


Chapter 21: Integration

Section 21.1: Integral, integral2, integral3

1 dimensional

To integrate a one dimensional function

```
f = @(x) sin(x).^3 + 1;
```

within the range

```
xmin = 2;
xmax = 8;
```

one can call the function

```
q = integral(f,xmin,xmax);
```

it's also possible to set boundaries for relative and absolute errors

```
q = integral(f,xmin,xmax, 'RelTol',10e-6, 'AbsTol',10-4);
```

2 dimensional

If one wants to integrate a two dimensional function

```
f = @(x,y) sin(x).^y ;
```

within the range

```
xmin = 2;
xmax = 8;
ymin = 1;
ymax = 4;
```

one calls the function

```
q = integral2(f,xmin,xmax,ymin,ymax);
```

Like in the other case it's possible to limit the tolerances

```
q = integral2(f,xmin,xmax,ymin,ymax, 'RelTol',10e-6, 'AbsTol',10-4);
```

3 dimensional

Integrating a three dimensional function

```
f = @(x,y,z) sin(x).^y - cos(z) ;
```

within the range

```
xmin = 2;
xmax = 8;
```

```
ymin = 1;
ymax = 4;
zmin = 6;
zmax = 13;
```

通过调用执行

```
q = integral3(f,xmin,xmax,ymin,ymax, zmin, zmax);
```

同样可以限制容差

```
q = integral3(f,xmin,xmax,ymin,ymax, zmin, zmax, 'RelTol',10e-6, 'AbsTol',10-4);
```



```
ymin = 1;
ymax = 4;
zmin = 6;
zmax = 13;
```

is performed by calling

```
q = integral3(f,xmin,xmax,ymin,ymax, zmin, zmax);
```

Again it's possible to limit the tolerances

```
q = integral3(f,xmin,xmax,ymin,ymax, zmin, zmax, 'RelTol',10e-6, 'AbsTol',10-4);
```

第22章：读取大文件

第22.1节：textscan

假设你有一个格式化的数据存储在一个大型文本文件或字符串中，例如：

```
Data,2015-09-16,15:41:52;781,780.000000,0.0034,2.2345
Data,2015-09-16,15:41:52;791,790.000000,0.1255,96.5948
Data,2015-09-16,15:41:52;801,800.000000,1.5123,0.0043
```

可以使用textscan来快速读取这些数据。为此，先用fopen获取文本文件的文件标识符：

```
fid = fopen('path/to/myfile');
```

假设对于本例中的数据，我们想忽略第一列“Data”，将日期和时间读取为字符串，
其余列读取为双精度数，即：

```
Data , 2015-09-16 , 15:41:52;801 , 800.000000 , 1.5123 , 0.0043
忽略     字符串         字符串          双精度       双精度     双精度
```

为此，调用：

```
data = textscan(fid,'%*s %s %s %f %f %f','Delimiter',',');
```

格式说明符中%*s表示“忽略此列”，%s表示“按字符串解析”，%f表示“按双精度浮点数解析”。最后，'Delimiter',','表示所有逗号都作为列之间的分隔符。

总结如下：

```
fid = fopen('path/to/myfile');
data = textscan(fid,'%*s %s %s %f %f %f','Delimiter',',');
```

data 现在包含一个单元数组，每列在一个单元中。

第22.2节：将日期和时间字符串快速转换为数值数组

将日期和时间字符串转换为数值数组可以使用 datenum，尽管这可能花费读取大型数据文件时间的一半。

考虑示例 **Textscan** 中的数据。通过再次使用 textscan 并将日期和时间解释为整数，它们可以
快速转换为数值数组。

即示例数据中的一行将被解释为：

```
数据 , 2015 - 09 - 16 , 15 : 41 : 52 ; 801 , 800.000000 , 1.5123 , 0.0043
ignore double double double double double double double double double double
```

将被读取为：

```
fid = fopen('path/to/myfile');
data = textscan(fid,'%*s %f %f %f %f %f %f %f %f %f %f','Delimiter',',-:;');
fclose(fid);
```


Chapter 22: Reading large files

Section 22.1: textscan

Assume you have formatted data in a large text file or string, e.g.

```
Data,2015-09-16,15:41:52;781,780.000000,0.0034,2.2345
Data,2015-09-16,15:41:52;791,790.000000,0.1255,96.5948
Data,2015-09-16,15:41:52;801,800.000000,1.5123,0.0043
```

one may use `textscan` to read this quite fast. To do so, get a file identifier of the text file with `fopen`:

```
fid = fopen('path/to/myfile');
```

Assume for the data in this example, we want to ignore the first column "Data", read the date and time as strings, and read the rest of the columns as doubles, i.e.

```
 Data  ,  2015-09-16  , 15:41:52;801  , 800.000000  , 1.5123  ,  0.0043
ignore     string         string         double      double     double
```

To do this, call:

```
data = textscan(fid,'%*s %s %s %f %f %f','Delimiter',',');
```

The asterisk in `%*s` means "ignore this column". `%s` means "interpret as a string". `%f` means "interpret as doubles (floats)". Finally, `'Delimiter',','` states that all commas should be interpreted as the delimiter between each column.

To sum up:

```
fid = fopen('path/to/myfile');
data = textscan(fid,'%*s %s %s %f %f %f','Delimiter',',');
```

`data` now contains a cell array with each column in a cell.

Section 22.2: Date and time strings to numeric array fast

Converting date and time strings to numeric arrays can be done with `datenum`, though it may take as much as half the time of reading a large data file.

Consider the data in example **Textscan**. By, again, using textscan and interpret date and time as integers, they can rapidly be converted into a numeric array.

I.e. a line in the example data would be interpreted as:

```
 Data , 2015 - 09 - 16 , 15 : 41 : 52 ; 801 , 800.000000 , 1.5123 , 0.0043
ignore double double double double double double double double double double
```

which will be read as:

```
fid = fopen('path/to/myfile');
data = textscan(fid,'%*s %f %f %f %f %f %f %f %f %f %f','Delimiter',',-:;');
fclose(fid);
```

现在：

```
y = data{1};          % 年
m = data{2};          % 月
d = data{3};          % 日
H = data{4};          % 小时
M = data{5};          % 分
S = data{6};          % 秒
F = data{7};          % 毫秒

% 从月份转换为天数
ms = [0,31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334];

n = length(y);        % 元素数量
Time = zeros(n,1);    % 声明数值时间数组

% 计算数值时间数组的算法
for k = 1:n
Time(k) = y(k)*365 + ms(m(k)) + d(k) + floor(y(k)/4)...
- floor(y(k)/100) + floor(y(k)/400) + (mod(y(k),4)~=0)...
- (mod(y(k),100)~=0) + (mod(y(k),400)~=0)...
+ (H(k)*3600 + M(k)*60 + S(k) + F(k)/1000)/86400 + 1;
end
```

使用 datenun处理566,678个元素耗时6.626570秒，而上述方法仅需0.048334秒，即 datenun时间的0.73%，约快137倍。

Now:

```
y = data{1};          % year
m = data{2};          % month
d = data{3};          % day
H = data{4};          % hours
M = data{5};          % minutes
S = data{6};          % seconds
F = data{7};          % milliseconds

% Translation from month to days
ms = [0,31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334];

n = length(y);        % Number of elements
Time = zeros(n,1);    % Declare numeric time array

% Algorithm for calculating numeric time array
for k = 1:n
    Time(k) = y(k)*365 + ms(m(k)) + d(k) + floor(y(k)/4)...
              - floor(y(k)/100) + floor(y(k)/400) + (mod(y(k),4)~=0)...
              - (mod(y(k),100)~=0) + (mod(y(k),400)~=0)...
              + (H(k)*3600 + M(k)*60 + S(k) + F(k)/1000)/86400 + 1;
end
```

Using datenum on 566,678 elements required 6.626570 seconds, whilst the method above required 0.048334 seconds, i.e. 0.73% of the time for datenum or ~137 times faster.

第23章：`accumarray()`函数的用法

参数	详细信息
subscriptArray	下标矩阵，指定为索引向量、索引矩阵或索引向量的单元数组。
`valuesArray`	数据，指定为向量或标量。
`sizeOfOutput`	输出数组的大小，指定为正整数向量。
`funcHandle`	在聚合过程中应用于每组项目的函数，指定为函数句柄或[]。
`fillVal`	填充值，当subs未引用输出中的每个元素时使用。
`isSparse`	输出应为稀疏数组吗？

accumarray 允许以多种方式聚合数组的元素，可能在此过程中对元素应用某些函数。accumarray 可以被视为一个轻量级的reducer（参见：MapReduce简介）。

本主题将包含accumarray 特别有用的常见场景。

第23.1节：对图像块应用滤波并将每个像素设置为每个块结果的平均值

许多现代图像处理算法使用图像块作为其基本处理单元。
例如，可以对图像块进行去噪（参见BM3D算法）。

然而，在从处理后的图像块构建图像时，同一像素会有多个结果。
处理它的一种方法是取同一像素所有值的平均值（经验均值）。

以下代码展示了如何将图像分割成小块，并使用平均值从小块重建图像，方法是使用[accumarray()][1]：

```
numRows = 5;
numCols = 5;

numRowsPatch = 3;
numColsPatch = 3;

% 图像
mI = rand([numRows, numCols]);

% 分解成小块 - 每个像素属于多个小块（忽略边界时，每个像素属于 (numRowsPatch * numCols
Patch) 个小块）。
mY = ImageToColumnsSliding(mI, [numRowsPatch, numColsPatch]);

% 这里可以对小块应用某些操作

% 创建每个像素索引的图像
mPxIdx = reshape(1:(numRows * numCols), [numRows, numCols]);

% 创建相同索引的小块
mSubsAccu = ImageToColumnsSliding(mPxIdx, [numRowsPatch, numColsPatch]);

% 重建图像 - 选项A
mO = accumarray(mSubsAccu(:), mY(:)) ./ accumarray(mSubsAccu(:), 1);

% 重建图像 - 选项B
mO = accumarray(mSubsAccu, mY(:), [(numRows * numCols), 1], @(x) mean(x));
```


Chapter 23: Usage of `accumarray()` Function

Parameter	Details
`subscriptArray`	Subscript matrix, specified as a vector of indices, matrix of indices, or cell array of index vectors.
`valuesArray`	Data, specified as a vector or a scalar.
`sizeOfOutput`	Size of output array, specified as a vector of positive integers.
`funcHandle`	Function to be applied to each set of items during aggregation, specified as a function handle or [ ].
`fillVal`	Fill value, for when `subs` does not reference each element in the output.
`isSparse`	Should the output be a sparse array?

`accumarray` allows to aggregate items of an array in various ways, potentially applying some function to the items in the process. `accumarray` can be thought of as a lightweight reducer (see also: Introduction to MapReduce).

This topic will contain common scenarios where `accumarray` is especially useful.

Section 23.1: Apply Filter to Image Patches and Set Each Pixel as the Mean of the Result of Each Patch

Many modern Image Processing algorithms use patches are their basic element to work on.
For instance one could denoise patches (See BM3D Algorithm).

Yet when building the image form the processed patches we have many results for the same pixel.
One way to deal with it is taking the average (Empirical Mean) of all values of the same pixel.

The following code shows how to break an image into patches and them reconstruct the image from patches using the average by using `[accumarray()][1]`:

```
numRows = 5;
numCols = 5;

numRowsPatch = 3;
numColsPatch = 3;

% The Image
mI = rand([numRows, numCols]);

% Decomposing into Patches - Each pixel is part of many patches (Neglecting
% boundaries, each pixel is part of (numRowsPatch * numColsPatch) patches).
mY = ImageToColumnsSliding(mI, [numRowsPatch, numColsPatch]);

% Here one would apply some operation which work on patches

% Creating image of the index of each pixel
mPxIdx = reshape(1:(numRows * numCols), [numRows, numCols]);

% Creating patches of the same indices
mSubsAccu = ImageToColumnsSliding(mPxIdx, [numRowsPatch, numColsPatch]);

% Reconstruct the image - Option A
mO = accumarray(mSubsAccu(:), mY(:)) ./ accumarray(mSubsAccu(:), 1);

% Reconstruct the image - Option B
mO = accumarray(mSubsAccu, mY(:), [(numRows * numCols), 1], @(x) mean(x));
```

```
% 将向量重塑为图像
mO = reshape(mO, [numRows, numCols]);
```

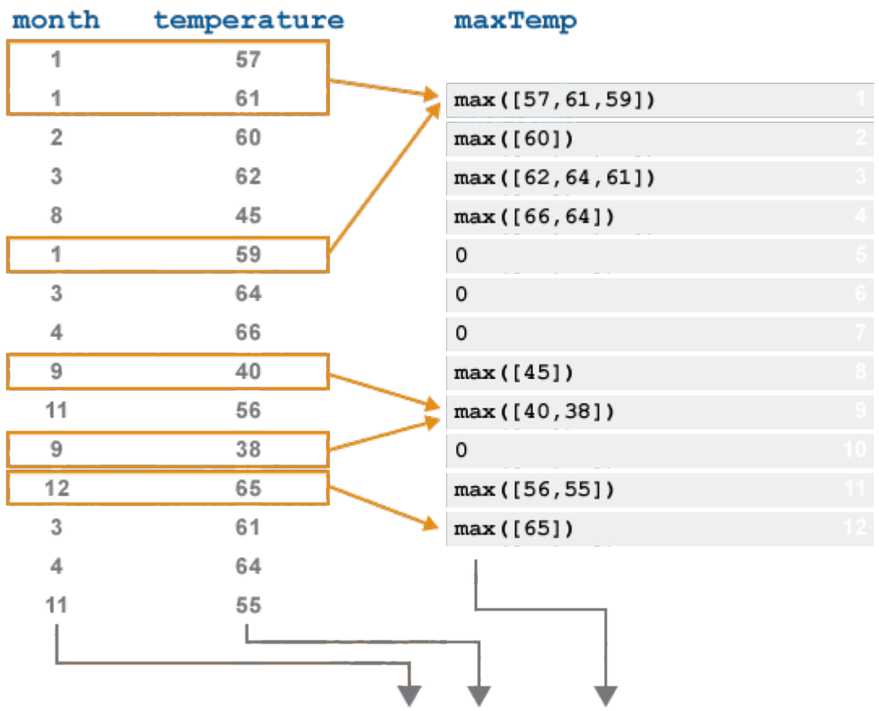
第23.2节：按另一个向量分组元素中查找最大值

这是一个官方的MATLAB示例

请考虑以下代码：

```
month = [1;1;2;3;8;1;3;4;9;11;9;12;3;4;11];
temperature = [57;61;60;62;45;59;64;66;40;56;38;65;61;64;55];
maxTemp = accumarray(month,temperature,[],@max);
```

下图展示了accumarray在此情况下的计算过程：



在本例中，所有具有相同month的值首先被收集，然后对每个这样的集合应用accumarray的第4个输入参数指定的函数（本例中为@max）。


```
% Reshape the Vector into the Image
mO = reshape(mO, [numRows, numCols]);
```

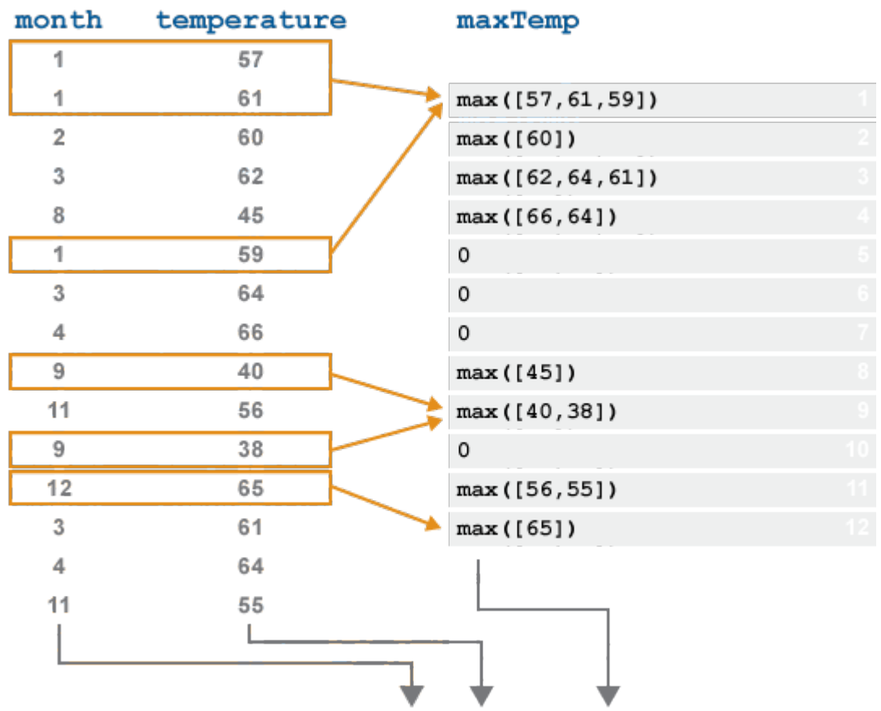
Section 23.2: Finding the maximum value among elements grouped by another vector

This is an official MATLAB example

Consider the following code:

```
month = [1;1;2;3;8;1;3;4;9;11;9;12;3;4;11];
temperature = [57;61;60;62;45;59;64;66;40;56;38;65;61;64;55];
maxTemp = accumarray(month,temperature,[],@max);
```

The image below demonstrates the computation process done by `accumarray` in this case:



In this example, all values that have the same `month` are first collected, and then the function specified by the 4th input to `accumarray` (in this case, `@max`) is applied to each such set.

第24章：MEX API简介

第24.1节：检查C++ MEX文件中的输入/输出数量

在本例中，我们将编写一个基本程序，用于检查传递给MEX函数的输入和输出数量。

作为起点，我们需要创建一个实现“MEX入口点”的C++文件。当从MATLAB调用该文件时，将执行此函数。

testinputs.cpp

```
// MathWorks 提供的头文件
#include "mex.h"

// 网关函数
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    // 如果输入参数数量不是3或4，则该函数会报错
    // 如果输出参数数量超过1，则该函数会报错

    // 检查输入参数：
    if (nrhs < 3 || nrhs > 4) {
        mexErrMsgIdAndTxt("Testinputs:ErrorIdIn",
                "传入 MEX 文件的输入参数数量无效。");
    }

    // 检查输出参数：
    if (nlhs > 1) {
mexErrMsgIdAndTxt("Testinputs:ErrorIdOut",
                "传出 MEX 文件的输出参数数量无效。");
    }
}
```

首先，我们包含了 mex.h 头文件，该文件包含了使用 MEX API 所需的所有函数和数据类型的定义。然后我们实现了函数 mexFunction，如图所示，其函数签名必须保持不变，无论实际使用的输入/输出参数如何。函数参数如下：

- nlhs：请求的输出数量。
- *plhs[]：包含所有输出的数组，采用MEX API格式。
- nrhs：传入的输入数量。
- *prhs[]：包含所有输入的数组，采用MEX API格式。

接下来，我们检查输入/输出参数的数量，如果验证失败，将使用mexErrMsgIdAndTxt函数抛出错误（它期望的格式是somename:iD，简单的“ID”格式无效）。

文件编译为mex testinputs.cpp后，可以在MATLAB中调用该函数：

```
>> testinputs(2,3)
使用testinputs时出错。传递给MEX文件的输入数量无效。

>> testinputs(2,3,5)

>> [~,~] = testinputs(2,3,3)
使用 testinputs 时出错。MEX 文件的输出数量无效。
```


Chapter 24: Introduction to MEX API

Section 24.1: Check number of inputs/outputs in a C++ MEX-file

In this example we will write a basic program that checks the number of inputs and outputs passed to a MEX-function.

As a starting point, we need to create a C++ file implementing the "MEX gateway". This is the function executed when the file is called from MATLAB.

testinputs.cpp

```
// MathWorks provided header file
#include "mex.h"

// gateway function
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    // This function will error if number of inputs it is not 3 or 4
    // This function will error if number of outputs is more than 1

    // Check inputs:
    if (nrhs < 3 || nrhs > 4) {
        mexErrMsgIdAndTxt("Testinputs:ErrorIdIn",
            "Invalid number of inputs to MEX file.");
    }

    // Check outputs:
    if (nlhs > 1) {
        mexErrMsgIdAndTxt("Testinputs:ErrorIdOut",
                "Invalid number of outputs to MEX file.");
    }
}
```

First, we include the `mex.h` header which contains definitions of all the required functions and data types to work with the MEX API. Then we implement the function `mexFunction` as shown, where its signature must not change, independent of the inputs/outputs actually used. The function parameters are as follows:

- `nlhs`: Number of outputs requested.
- `*plhs[]`: Array containing all the outputs in MEX API format.
- `nrhs`: Number of inputs passed.
- `*prhs[]`: Array containing all the inputs in MEX API format.

Next, we check the number of inputs/outputs arguments, and if the validation fails, an error is thrown using `mexErrMsgIdAndTxt` function (it expects `somename:iD` format identifier, a simple "ID" won't work).

Once the file is compiled as `mex testinputs.cpp`, the function can be called in MATLAB as:

```
>> testinputs(2,3)
Error using testinputs. Invalid number of inputs to MEX file.

>> testinputs(2,3,5)

>> [~,~] = testinputs(2,3,3)
Error using testinputs. Invalid number of outputs to MEX file.
```

第 24.2 节：输入字符串，在 C 中修改并输出

在本例中，我们演示了 MATLAB MEX 中的字符串操作。我们将创建一个 MEX 函数，该函数接受来自 MATLAB 的字符串作为输入，将数据复制到 C 字符串中，进行修改，然后转换回mxArray并返回给 MATLAB 端。

本例的主要目的是展示如何在 MATLAB 和 C/C++ 之间相互转换字符串。

stringIO.cpp

```
#include "mex.h"
#include <cstring>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    // 检查参数数量
    if (nrhs != 1 || nlhs > 1) {
mexErrMsgIdAndTxt("StringIO:WrongNumArgs", "参数数量错误。");
    }

    // 检查输入是否为字符串
    if (mxIsChar(prhs[0])) {
mexErrMsgIdAndTxt("StringIO:TypeError", "输入不是字符串");
    }

    // 将 mxArray 中的字符数据复制到 C 风格字符串（以 null 结尾）
    char *str = mxArrayToString(prhs[0]);

    // 以某种方式操作字符串
    if (strcmp("theOneString", str) == 0) {
        str[0] = 'T';  // 将首字母大写
    } else {
str[0] = ' ';  // 做其他操作？
    }

    // 返回修改后的新字符串
    plhs[0] = mxCreateString(str);

    // 释放分配的内存
    mxFree(str);
}
```

本示例中相关的函数有：

- mxIsChar 用于测试 mxArray 是否为 mxCHAR 类型。
- mxArrayToString 用于将 mxArray 字符串的数据复制到 char * 缓冲区。
- mxCreateString 用于从 char* 创建一个 mxArray 字符串。

顺便提一句，如果你只想读取字符串，而不修改它，记得声明为const char*以提高速度和稳定性。

最后，编译完成后，我们可以从MATLAB中这样调用它：

```
>> mex stringIO.cpp

>> strOut = stringIO('theOneString')
strOut =
TheOneString

>> strOut = stringIO('somethingelse')
```


Section 24.2: Input a string, modify it in C, and output it

In this example, we illustrate string manipulation in MATLAB MEX. We will create a MEX-function that accepts a string as input from MATLAB, copy the data into C-string, modify it and convert it back to `mxArray` returned to the MATLAB side.

The main objective of this example is to show how strings can be converted to C/C++ from MATLAB and vice versa.

stringIO.cpp

```
#include "mex.h"
#include <cstring>

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    // check number of arguments
    if (nrhs != 1 || nlhs > 1) {
        mexErrMsgIdAndTxt("StringIO:WrongNumArgs", "Wrong number of arguments.");
    }

    // check if input is a string
    if (mxIsChar(prhs[0])) {
        mexErrMsgIdAndTxt("StringIO:TypeError", "Input is not a string");
    }

    // copy characters data from mxArray to a C-style string (null-terminated)
    char *str = mxArrayToString(prhs[0]);

    // manipulate the string in some way
    if (strcmp("theOneString", str) == 0) {
        str[0] = 'T';  // capitalize first letter
    } else {
        str[0] = ' ';  // do something else?
    }

    // return the new modified string
    plhs[0] = mxCreateString(str);

    // free allocated memory
    mxFree(str);
}
```

The relevant functions in this example are:

- `mxIsChar` to test if an `mxArray` is of `mxCHAR` type.
- `mxArrayToString` to copy the data of a `mxArray` string to a `char *` buffer.
- `mxCreateString` to create an `mxArray` string from a `char*`.

As a side note, if you only want to read the string, and not modify it, remember to declare it as `const char*` for speed and robustness.

Finally, once compiled we can call it from MATLAB as:

```
>> mex stringIO.cpp

>> strOut = stringIO('theOneString')
strOut =
TheOneString

>> strOut = stringIO('somethingelse')
```

```
strOut=
omethingelse
```

第24.3节：通过字段名传递结构体

本例演示了如何从MATLAB读取各种类型的结构体条目，并将其传递给C语言中的等效类型变量。

虽然从示例中推断如何通过数字加载字段是可能且简单的，但这里是通过将字段名与字符串比较来实现的。因此，可以通过字段名访问结构体字段，并由C语言读取其中的变量。

structIn.c

```
#include "mex.h"
#include <string.h> // strcmp


void mexFunction (int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
  // 辅助函数
  double* double_ptr;
  unsigned int i; // 循环变量

  // 待读取变量
  bool optimal;
  int randomseed;
  unsigned int desiredNodes;

  if (!mxIsStruct(prhs[0])) {
mexErrMsgTxt("第一个参数必须是参数结构体！");
  }
  for (i=0; i<mxGetNumberOfFields(prhs[0]); i++) {
    if (0==strcmp(mxGetFieldNameByNumber(prhs[0],i),"randomseed")) {
      mxArray *p = mxGetFieldByNumber(prhs[0],0,i);
      randomseed = *mxGetPr(p);
    }
    if (0==strcmp(mxGetFieldNameByNumber(prhs[0],i),"optimal")) {
      mxArray *p = mxGetFieldByNumber(prhs[0],0,i);
      optimal = (bool)*mxGetPr(p);
    }
    if (0==strcmp(mxGetFieldNameByNumber(prhs[0],i),"numNodes")) {
      mxArray *p = mxGetFieldByNumber(prhs[0],0,i);
      desiredNodes = *mxGetPr(p);
    }
  }
}
```

对 i 的循环遍历给定结构体中的每个字段，而 if(0==strcmp) 部分则将 MATLAB 字段名与给定字符串进行比较。如果匹配，则将对应的值提取到 C 变量中。

第24.4节：从 MATLAB 传递三维矩阵到 C

本例演示如何从 MATLAB 获取一个 double 实数类型的三维矩阵，并将其传递给 C 的 `double*` 数组。

本例的主要目标是展示如何从 MATLAB MEX 数组中获取数据，并强调矩阵存储和处理中的一些细节。


```
strOut=
omethingelse
```

Section 24.3: Passing a struct by field names

This example illustrates how to read various-type struct entries from MATLAB, and pass it to C equivalent type variables.

While it is possible and easy to figure out from the example how to load fields by numbers, it is here achieved via comparing the field names to strings. Thus the struct fields can be addressed by their field names and variables in it can be read by C.

structIn.c

```
#include "mex.h"
#include <string.h> // strcmp


void mexFunction (int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
  // helpers
  double* double_ptr;
  unsigned int i; // loop variable

  // to be read variables
  bool optimal;
  int randomseed;
  unsigned int desiredNodes;

  if (!mxIsStruct(prhs[0])) {
    mexErrMsgTxt("First argument has to be a parameter struct!");
  }
  for (i=0; i<mxGetNumberOfFields(prhs[0]); i++) {
    if (0==strcmp(mxGetFieldNameByNumber(prhs[0],i),"randomseed")) {
      mxArray *p = mxGetFieldByNumber(prhs[0],0,i);
      randomseed = *mxGetPr(p);
    }
    if (0==strcmp(mxGetFieldNameByNumber(prhs[0],i),"optimal")) {
      mxArray *p = mxGetFieldByNumber(prhs[0],0,i);
      optimal = (bool)*mxGetPr(p);
    }
    if (0==strcmp(mxGetFieldNameByNumber(prhs[0],i),"numNodes")) {
      mxArray *p = mxGetFieldByNumber(prhs[0],0,i);
      desiredNodes = *mxGetPr(p);
    }
  }
}
```

The loop over `i` runs over every field in the given struct, while the `if(0==strcmp)`-parts compare the MATLAB field's name to the given string. If it is a match, the corresponding value is extracted to a C variable.

Section 24.4: Pass a 3D matrix from MATLAB to C

In this example we illustrate how to take a double real-type 3D matrix from MATLAB, and pass it to a C `double*` array.

The main objectives of this example are showing how to obtain data from MATLAB MEX arrays and to highlight some small details in matrix storage and handling.

matrixIn.cpp

```
#include "mex.h"

void mexFunction(int   nlhs , mxArray *plhs[],
          int nrhs, mxArray const *prhs[]){
    // 检查输入参数数量
    if (nrhs!=1) {
mexErrMsgIdAndTxt("matrixIn:InvalidInput", "MEX文件的输入数量无效。");
     }

    // 检查输入类型
    if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0])){
         mexErrMsgIdAndTxt("matrixIn:InvalidType",  "输入矩阵必须是双精度非复数数组。");

    }

    // 提取数据
    double const * const matrixAux= static_cast<double const *>(mxGetData(prhs[0]));

    // 获取矩阵大小
    const mwSize *sizeInputMatrix= mxGetDimensions(prhs[0]);

    // 在C中分配数组。注意：这是1维数组，即使我们的输入是3维的
    double*   matrixInC= (double*)malloc(sizeInputMatrix[0] *sizeInputMatrix[1] *sizeInputMatrix[2]*
sizeof(double));


    // MATLAB是列优先，而非行优先（如C语言）。我们需要重新排序数字
    // 基本上是维度的置换

    // 注意：循环的顺序经过优化以实现最快的内存访问！
    // 这能提升大约300%的速度

     const int size0 = sizeInputMatrix[0]; // const使编译器优化生效
     const int size1 = sizeInputMatrix[1];
     const int size2 = sizeInputMatrix[2];

     for (int j = 0; j < size2; j++)
     {
          int jOffset = j*size0*size1; // 这可以节省重新计算的时间
          for (int k = 0; k < size0; k++)
          {
               int kOffset = k*size1; // 这可以节省重新计算的时间
               for (int i = 0; i < size1; i++)
               {
                    int iOffset = i*size0;
matrixInC[i + jOffset + kOffset] = matrixAux[iOffset + jOffset + k];
               }
          }
     }

     // 我们完成了！

     // 在这里使用你的 C 矩阵

     // 释放内存
     free(matrixInC);
     return;
}
```

需要了解的相关概念：

matrixIn.cpp

```
#include "mex.h"

void mexFunction(int   nlhs , mxArray *plhs[],
          int nrhs, mxArray const *prhs[]){
    // check amount of inputs
    if (nrhs!=1) {
         mexErrMsgIdAndTxt("matrixIn:InvalidInput", "Invalid number of inputs to MEX file.");
     }

    // check type of input
    if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0])){
         mexErrMsgIdAndTxt("matrixIn:InvalidType",  "Input matrix must be a double, non-complex
array.");
    }

    // extract the data
    double const * const matrixAux= static_cast<double const *>(mxGetData(prhs[0]));

    // Get matrix size
    const mwSize *sizeInputMatrix= mxGetDimensions(prhs[0]);

    // allocate array in C. Note: its 1D array, not 3D even if our input is 3D
    double*   matrixInC= (double*)malloc(sizeInputMatrix[0] *sizeInputMatrix[1] *sizeInputMatrix[2]*
sizeof(double));


    // MATLAB is column major, not row major (as C). We need to reorder the numbers
    // Basically permutes dimensions

    // NOTE: the ordering of the loops is optimized for fastest memory access!
    // This improves the speed in about 300%

     const int size0 = sizeInputMatrix[0]; // Const makes compiler optimization kick in
     const int size1 = sizeInputMatrix[1];
     const int size2 = sizeInputMatrix[2];

     for (int j = 0; j < size2; j++)
     {
         int jOffset = j*size0*size1; // this saves re-computation time
         for (int k = 0; k < size0; k++)
         {
             int kOffset = k*size1; // this saves re-computation time
             for (int i = 0; i < size1; i++)
             {
                 int iOffset = i*size0;
                 matrixInC[i + jOffset + kOffset] = matrixAux[iOffset + jOffset + k];
             }
         }
     }

     // we are done!

     // Use your C matrix here

     // free memory
     free(matrixInC);
     return;
}
```

The relevant concepts to be aware of:

- MATLAB 矩阵在内存中都是一维的，无论它们在 MATLAB 中有多少维度。
  这对于大多数（如果不是全部的话）C/C++ 库中的主要矩阵表示也是如此，因为这允许优化和更快的执行。
- 你需要在循环中显式地将矩阵从 MATLAB 复制到 C。
- MATLAB 矩阵以列优先顺序存储，类似 Fortran，但 C/C++ 和大多数现代语言是行优先的。对输入矩阵进行置换非常重要，否则数据看起来会完全不同。

本例中相关的函数有：

- mxIsDouble 检查输入是否为 double 类型。
- mxIsComplex 检查输入是实数还是虚数。
- mxGetData 返回指向输入数组中实数数据的指针。 NULL 如果没有实数数据。
- mxGetDimensions 返回指向 mwSize 数组的指针，该数组每个索引包含对应维度的大小。

- MATLAB matrices are all 1D in memory, no matter how many dimensions they have when used in MATLAB. This is also true for most (if not all) main matrix representation in C/C++ libraries, as allows optimization and faster execution.
- You need to explicitly copy matrices from MATLAB to C in a loop.
- MATLAB matrices are stored in column major order, as in Fortran, but C/C++ and most modern languages are row major. It is important to permute the input matrix , or else the data will look completely different.

The relevant function in this example are:

- `mxIsDouble` checks if input is `double` type.
- `mxIsComplex` checks if input is real or imaginary.
- `mxGetData` returns a pointer to the real data in the input array. `NULL` if there is no real data.
- `mxGetDimensions` returns an pointer to a `mwSize` array, with the size of the dimension in each index.

第25章：调试

参数	详细信息
文件	文件名.m（不含扩展名），例如fit。除非设置特殊的条件断点类型，如dbstop if error或dbstop if naninf，否则此参数为（必需）。
位置	断点应设置的行号。如果指定的行不包含可运行代码，断点将被设置在指定行之后的第一条有效代码行上。
表达式	任何求值为布尔值的表达式或其组合。例如：ind == 1，nargin < 4 && isdir('Q:\')。

第25.1节：使用断点

定义

> *在软件开发中，断点是程序中有意设置的停止或暂停点，用于调试目的。*
>
> *更广义地说，断点是在程序执行过程中获取程序信息的一种手段。在中断期间，程序员检查测试环境（通用寄存器、内存、日志、文件等）以确定程序是否按预期运行。实际上，断点由一个或多个条件组成，用于决定何时中断程序的执行。*
>
> -维基百科

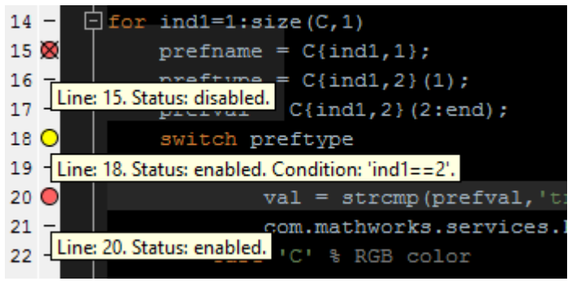
MATLAB中的断点
动机

在MATLAB中，当执行在断点处暂停时，可以检查当前工作区（也称为作用域）或任何调用工作区中存在的变量，通常也可以修改这些变量。

断点类型

MATLAB允许用户在.m文件中设置两种类型的断点：

- 标准（或"无限制"）断点（以红色显示）——每当执行到标记的行时暂停执行。
- "条件"断点（以黄色显示）——每当执行到标记的行且断点中定义的条件被评估为真时暂停执行。



设置断点

这两种断点都可以通过多种方式创建：

Chapter 25: Debugging

Parameter	Details
file	Name of `.m` file (without extension), e.g. `fit`. This parameter is *(Required)* unless setting special conditional breakpoint types such as `dbstop if error` or `dbstop if naninf`.
location	Line number where the breakpoint should be placed. If the specified line does not contain runnable code, the breakpoint will be placed on the first valid line **after** the specified one.
expression	Any expression or combination thereof that evaluates to a boolean value. Examples: `ind == 1`, `nargin < 4 && isdir('Q:\')`.

Section 25.1: Working with Breakpoints

Definition

> *In software development, a **breakpoint** is an intentional stopping or pausing place in a program, put in place for debugging purposes.*
>
> *More generally, a breakpoint is a means of acquiring knowledge about a program during its execution. During the interruption, the programmer inspects the test environment (general purpose registers, memory, logs, files, etc.) to find out whether the program is functioning as expected. In practice, a breakpoint consists of one or more conditions that determine when a program's execution should be interrupted.*
>
> -Wikipedia

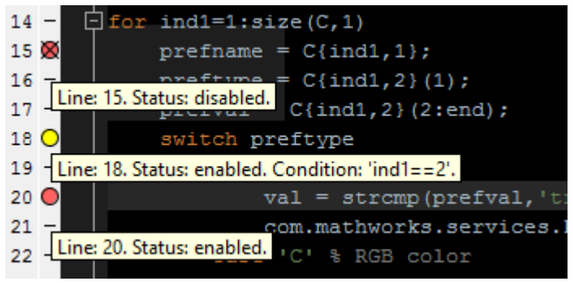
Breakpoints in MATLAB
Motivation

In MATLAB, when execution pauses at a breakpoint, variables existing in the current workspace (a.k.a. *scope*) or any of the calling workspaces, can be inspected (and usually also modified).

Types of Breakpoints

MATLAB allow users to place two types of breakpoints in `.m` files:

- Standard (or "*unrestricted*") breakpoints (shown in red) - pause execution whenever the marked line is reached.
- "Conditional" breakpoints (shown in yellow) - pause execution whenever the marked line is reached AND the condition defined in the breakpoint is evaluated as `true`.



Placing Breakpoints

Both types of breakpoints can be created in several ways:

- 使用MATLAB编辑器GUI，通过右键点击行号旁的水平线。
- 使用dbstop命令：

```
% 创建一个无限制的断点：
dbstop 在文件中指定位置
% 创建一个条件断点：
dbstop 在文件中指定位置 如果 表达式

% 示例和特殊情况：
dbstop 在 fit 的第 99 行 % 标准无限制断点。

dbstop 在 fit 的第 99 行 如果 nargin==3 % 标准条件断点。

dbstop 如果出错 % 这种特殊类型的断点不限于特定文件，
                % 并且会在“可调试”代码中*每当*遇到错误时触发。

dbstop 在文件中 % 这将在“文件”的第一条可执行行创建一个无限制断点。


dbstop 如果 naninf % 这个特殊断点会在计算结果包含 NaN（表示除以0）或 Inf 时触发
```

- 使用键盘快捷键：在 Windows 上创建标准断点的默认按键是 F12；条件断点的默认键是 unset。

禁用和重新启用断点

禁用断点以暂时忽略它：禁用的断点不会暂停执行。禁用断点可以通过多种方式完成：

- 右键点击红色/黄色断点圆圈 > 禁用断点。
- 左键点击条件断点（黄色）。
- 在编辑器标签 > 断点 > 启用\禁用。

移除断点

所有断点会保留在文件中，直到被手动或自动移除。断点会在结束 MATLAB 会话（即终止程序）时自动清除。手动清除断点可以通过以下方式之一完成：

- 使用 dbclear 命令：

```
dbclear all
dbclear 在文件中
dbclear 在位置的文件中
dbclear 如果 条件
```

- 左键单击标准断点图标，或已禁用的条件断点图标。
- 右键单击任意断点 > 清除断点。
- 在编辑器标签 > 断点 > 清除所有。
- 在 MATLAB 2015b 之前的版本中，使用命令 clear。

继续执行

- Using the MATLAB Editor GUI, by right clicking the horizontal line next to the line number.
- Using the `dbstop` command:

```
% Create an unrestricted breakpoint:
dbstop in file at location
% Create a conditional breakpoint:
dbstop in file at location if expression

% Examples and special cases:
dbstop in fit at 99 % Standard unrestricted breakpoint.

dbstop in fit at 99 if nargin==3 % Standard conditional breakpoint.

dbstop if error % This special type of breakpoint is not limited to a specific file, and
                % will trigger *whenever* an error is encountered in "debuggable" code.

dbstop in file % This will create an unrestricted breakpoint on the first executable line
               % of "file".

dbstop if naninf % This special breakpoint will trigger whenever a computation result
                 % contains either a NaN (indicates a division by 0) or an Inf
```

- Using keyboard shortcuts: the default key for creating a standard breakpoint on Windows is F12; the default key for conditional breakpoints is *unset*.

Disabling and Re-enabling Breakpoints

Disable a breakpoint to temporarily ignore it: disabled breakpoints do not pause execution. Disabling a breakpoint can be done in several ways:

- Right click on the red/yellow breakpoint circle > Disable Breakpoint.
- Left click on a conditional (yellow) breakpoint.
- In the Editor tab > Breakpoints > Enable\Disable.

Removing Breakpoints

All breakpoints remain in a file until removed, either manually or automatically. Breakpoints are cleared *automatically* when ending the MATLAB session (i.e. terminating the program). Clearing breakpoints manually is done in one of the following ways:

- Using the `dbclear` command:

```
dbclear all
dbclear in file
dbclear in file at location
dbclear if condition
```

- Left clicking a standard breakpoint icon, or a disabled conditional breakpoint icon.
- Right clicking on any breakpoint > Clear Breakpoint.
- In the Editor tab > Breakpoints > Clear All.
- In pre-R2015b versions of MATLAB, using the command `clear`.

Resuming Execution

当执行在断点处暂停时，有两种方式继续执行程序：

- 执行当前行，并在下一行之前再次暂停。

  F10 编辑器中按 1，命令窗口中输入 dbstep，功能区 > 编辑器 > 调试 中点击“单步执行”。

- 执行直到下一个断点（如果没有更多断点，执行将继续直到程序结束）。

  F12 编辑器中的1，命令窗口中的dbcont，功能区 > 编辑器 > 调试中的“继续”。

Windows上的默认值为1。

第25.2节：调试由MATLAB调用的Java代码

概述

为了调试在MATLAB执行过程中调用的Java类，需要执行以下两个步骤：

1. 以JVM调试模式运行MATLAB。
2. 将Java调试器附加到MATLAB进程。

当MATLAB以JVM调试模式启动时，命令窗口会显示以下信息：

```
JVM正在以启用调试的方式启动。
使用 "jdb -connect com.sun.jdi.SocketAttach:port=4444" 来附加调试器。
```

MATLAB 结束

Windows：

创建一个指向 MATLAB 可执行文件（matlab.exe）的快捷方式，并在末尾添加-jdb标志，如下所示：

When execution is paused at a breakpoint, there are two ways to continue executing the program:

- Execute the current line and pause again before the next line.

  F10 1 in the Editor, `dbstep` in the Command Window, "Step" in Ribbon > Editor > DEBUG.

- Execute until the next breakpoint (if there are no more breakpoints, the execution proceeds until the end of the program).

  F12 1 in the Editor, `dbcont` in the Command Window, "Continue" in Ribbon > Editor > DEBUG.

1 - default on Windows.

Section 25.2: Debugging Java code invoked by MATLAB

Overview

In order to debug Java classes that are called during MATLAB execution, it is necessary to perform two steps:

1. Run MATLAB in JVM debugging mode.
2. Attach a Java debugger to the MATLAB process.

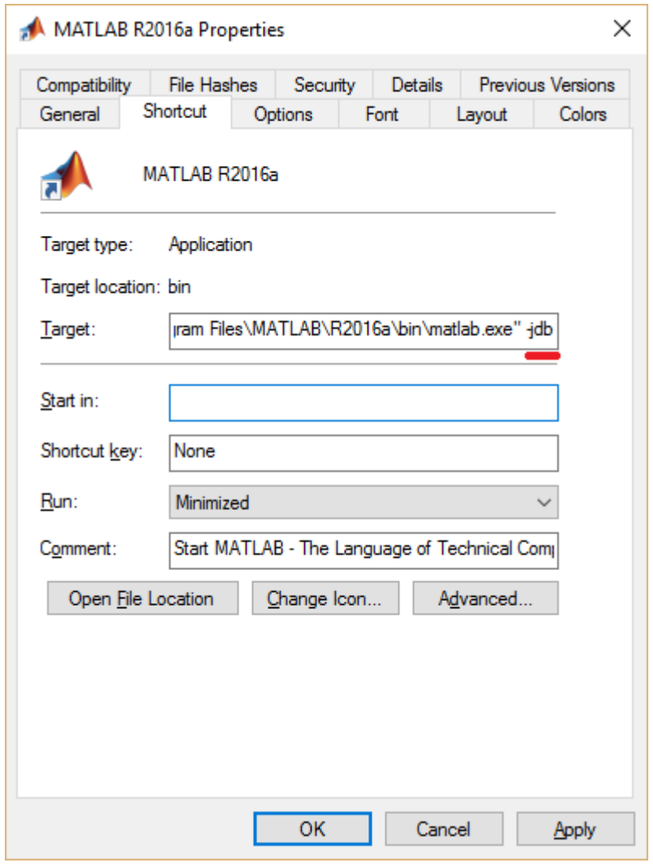
When MATLAB is started in JVM debugging mode, the following message appears in the command window:

```
JVM is being started with debugging enabled.
Use "jdb -connect com.sun.jdi.SocketAttach:port=4444" to attach debugger.
```

MATLAB end

Windows:

Create a shortcut to the MATLAB executable (`matlab.exe`) and add the `-jdb` flag at the end as shown below:



使用此快捷方式运行 MATLAB 时，将启用 JVM 调试。

或者，可以创建/更新java.opts文件。该文件存储在“matlab-root\bin\arch”目录下，其中“matlab-root”是 MATLAB 安装目录，“arch”是架构（例如“win32”）。

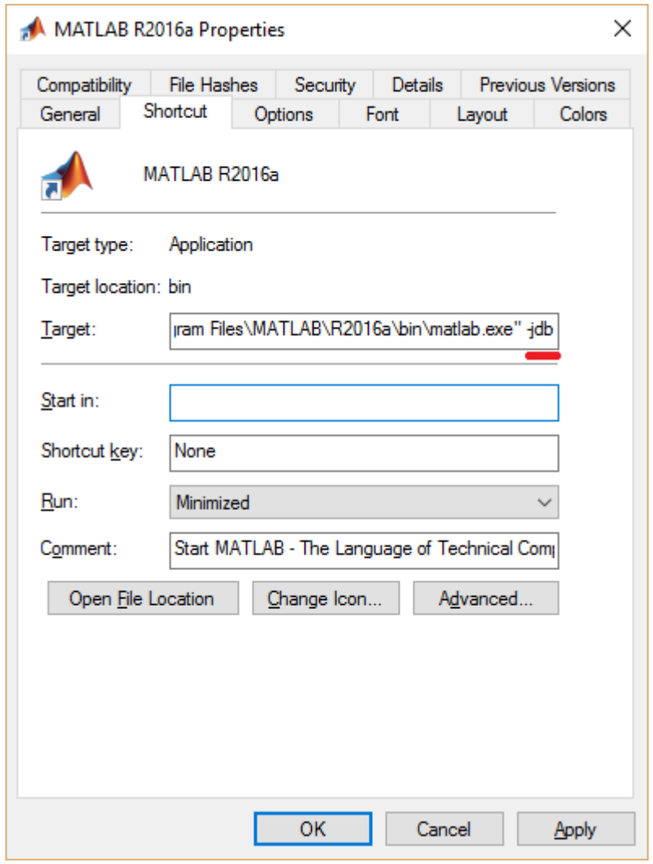
应在文件中添加以下内容：

```
-Xdebug
-Xrunjdwp:transport=dt_socket,address=1044,server=y,suspend=n
```

调试器端
IntelliJ IDEA

附加此调试器需要创建一个“远程调试”配置，端口需与

MATLAB暴露的端口一致：



When running MATLAB using this shortcut JVM debugging will be enabled.

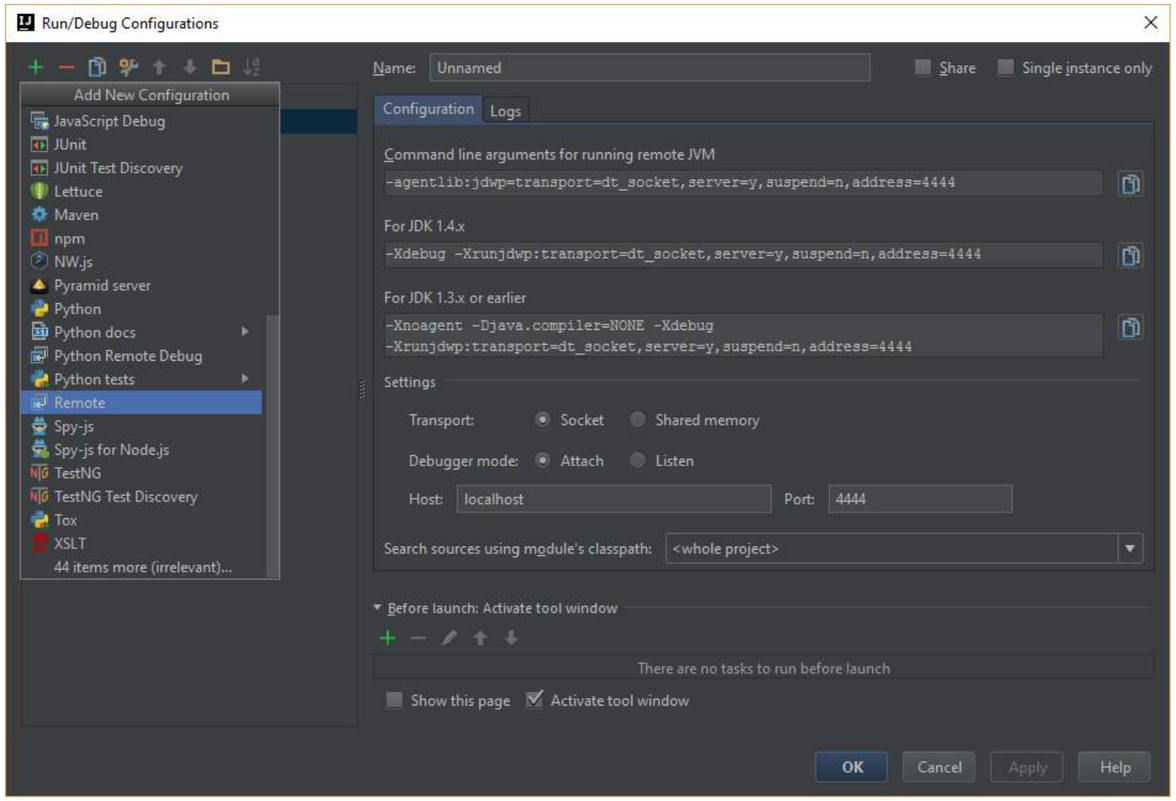
Alternatively the `java.opts` file can be created/updated. This file is stored in "matlab-root\bin\arch", where "matlab-root" is the MATLAB installation directory and "arch" is the architecture (e.g. "win32").

The following should be added in the file:

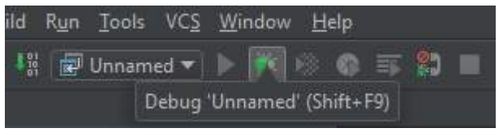
```
-Xdebug
-Xrunjdwp:transport=dt_socket,address=1044,server=y,suspend=n
```

Debugger end
IntelliJ IDEA

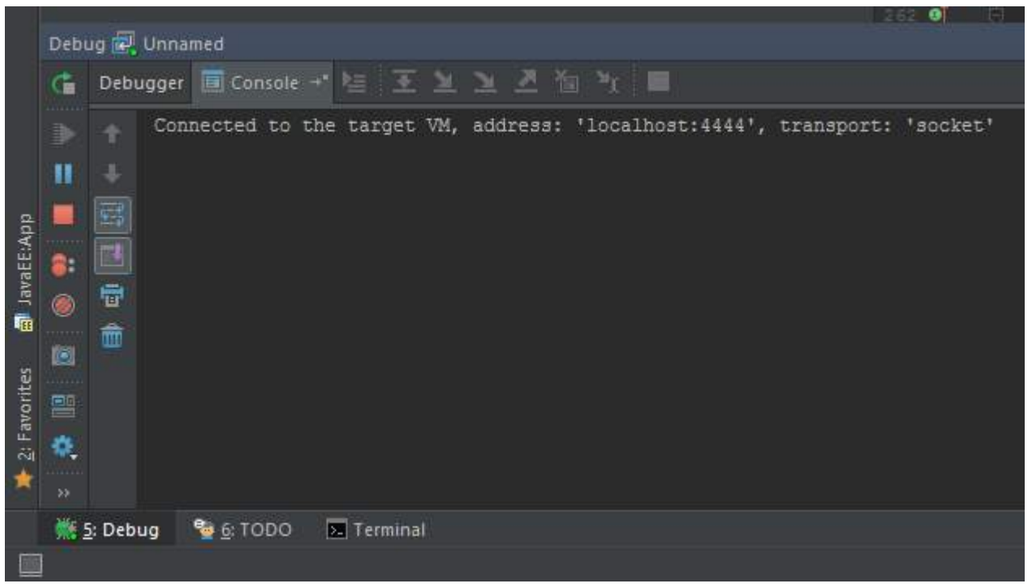
Attaching this debugger requires the creation of a "remote debugging" configuration with the port exposed by MATLAB:

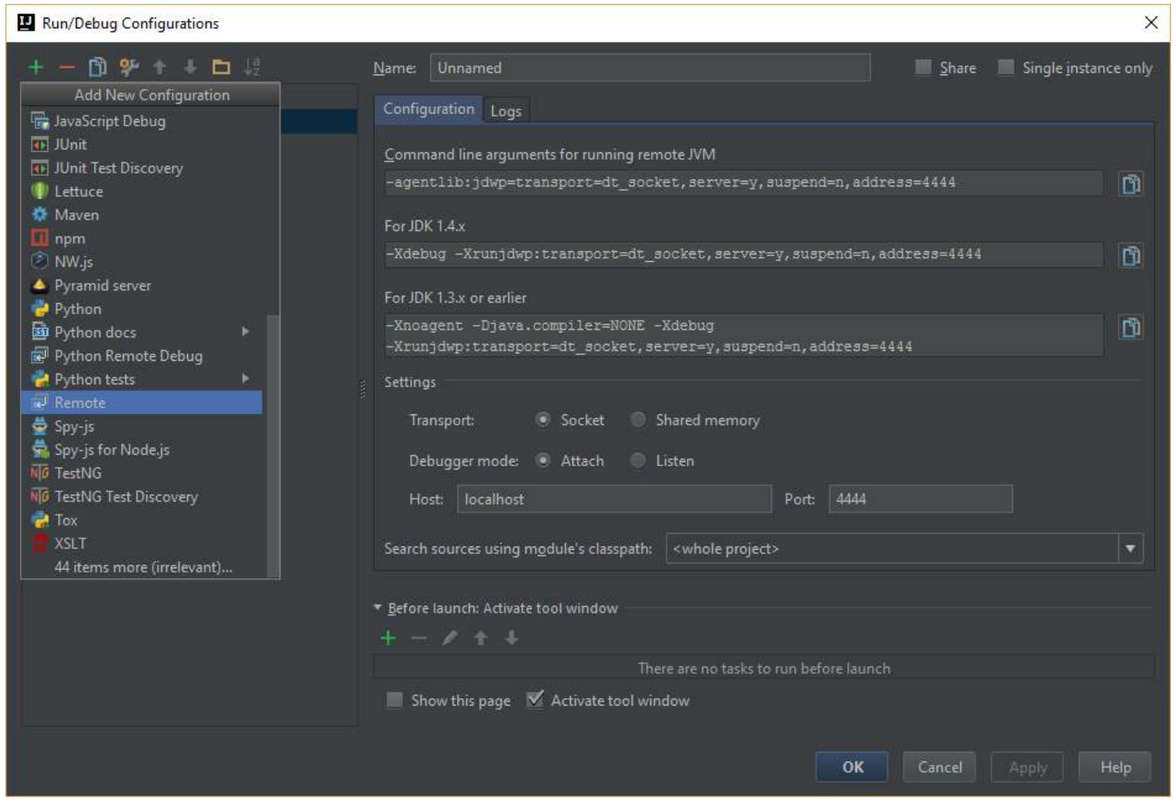


然后启动调试器：

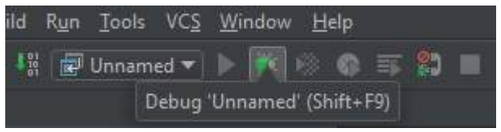


如果一切正常，控制台将显示以下信息：

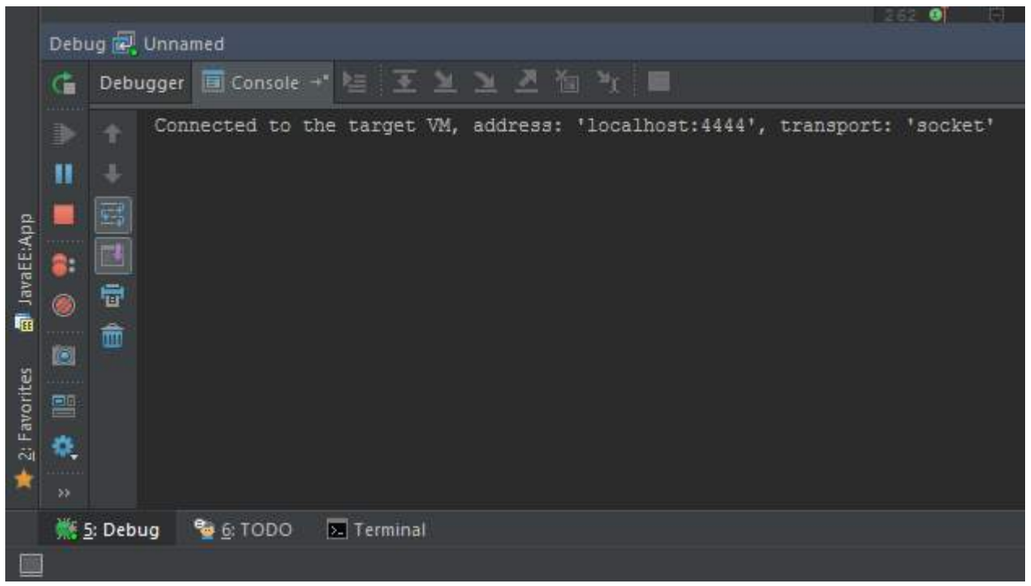




Then the debugger is started:



If everything is working as expected, the following message will appear in the console:



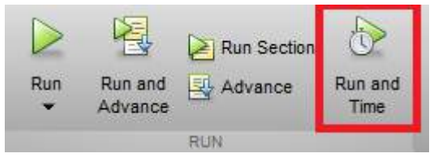
第26章：性能与基准测试

第26.1节：使用Profiler识别性能瓶颈

MATLABProfiler是一个用于MATLAB代码软件性能分析的工具。通过Profiler，可以获得执行时间和内存消耗的可视化表示。

运行Profiler有两种方式：

- 在MATLAB图形界面中打开某个.m文件时，点击“运行并计时”按钮（此功能自R2012b版本起提供）。



- 通过编程方式，使用：

```
profile on
<some code we want to test>
profile 关闭
```

以下是一些示例代码及其性能分析结果：

```
function docTest

for ind1 = 1:100
  [~] = var(...
            sum(...
                randn(1000)));
end

spy
```

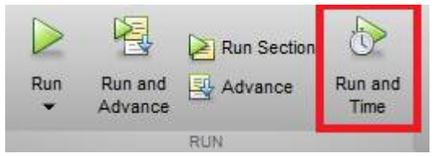

Chapter 26: Performance and Benchmarking

Section 26.1: Identifying performance bottlenecks using the Profiler

The MATLAB Profiler is a tool for software profiling of MATLAB code. Using the Profiler, it is possible to obtain a visual representation of both execution time and memory consumption.

Running the Profiler can be done in two ways:

- Clicking the "Run and Time" button in the MATLAB GUI while having some `.m` file open in the editor (added in **R2012b**).

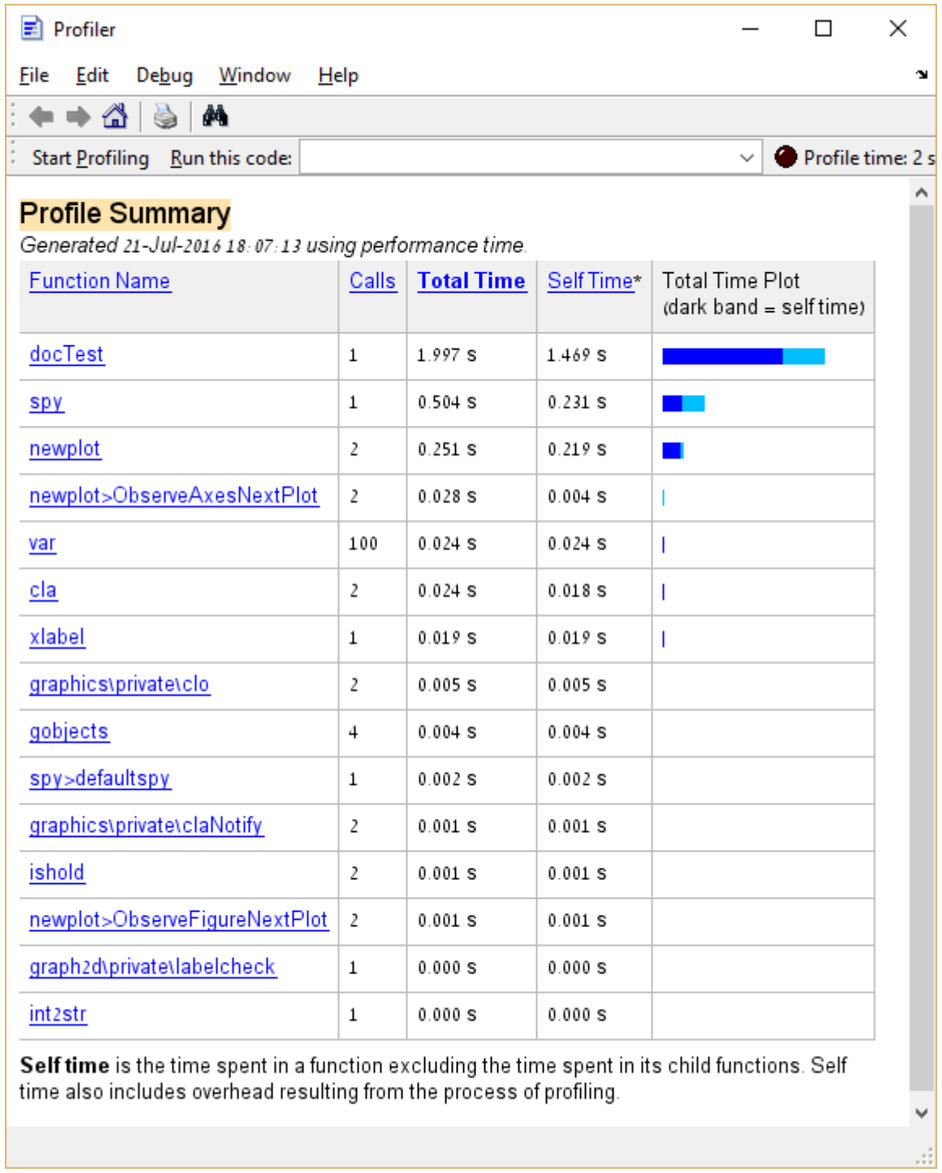


- Programmatically, using:

```
profile on
<some code we want to test>
profile off
```

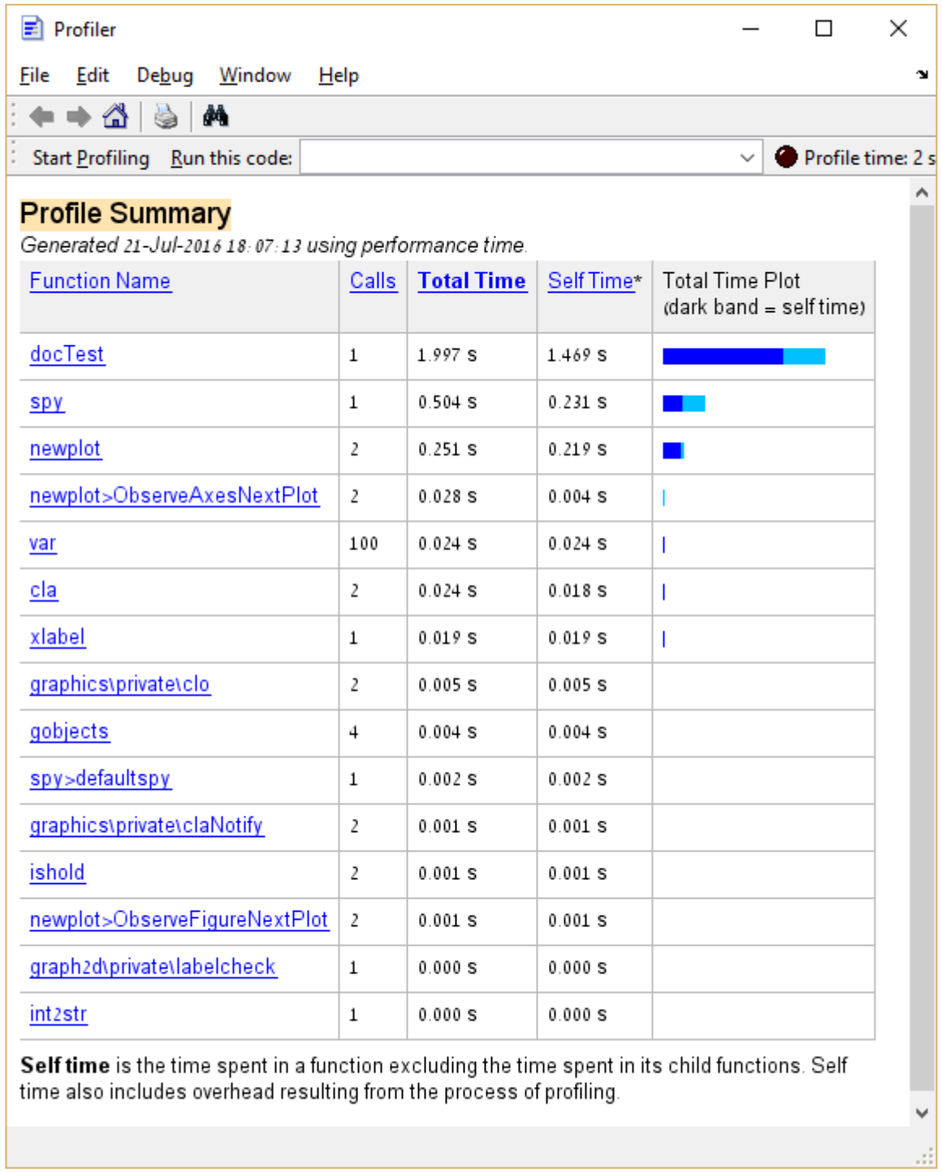
Below is some sample code and the result of its profiling:

```
function docTest

for ind1 = 1:100
  [~] = var(...
            sum(...
                randn(1000)));
end

spy
```



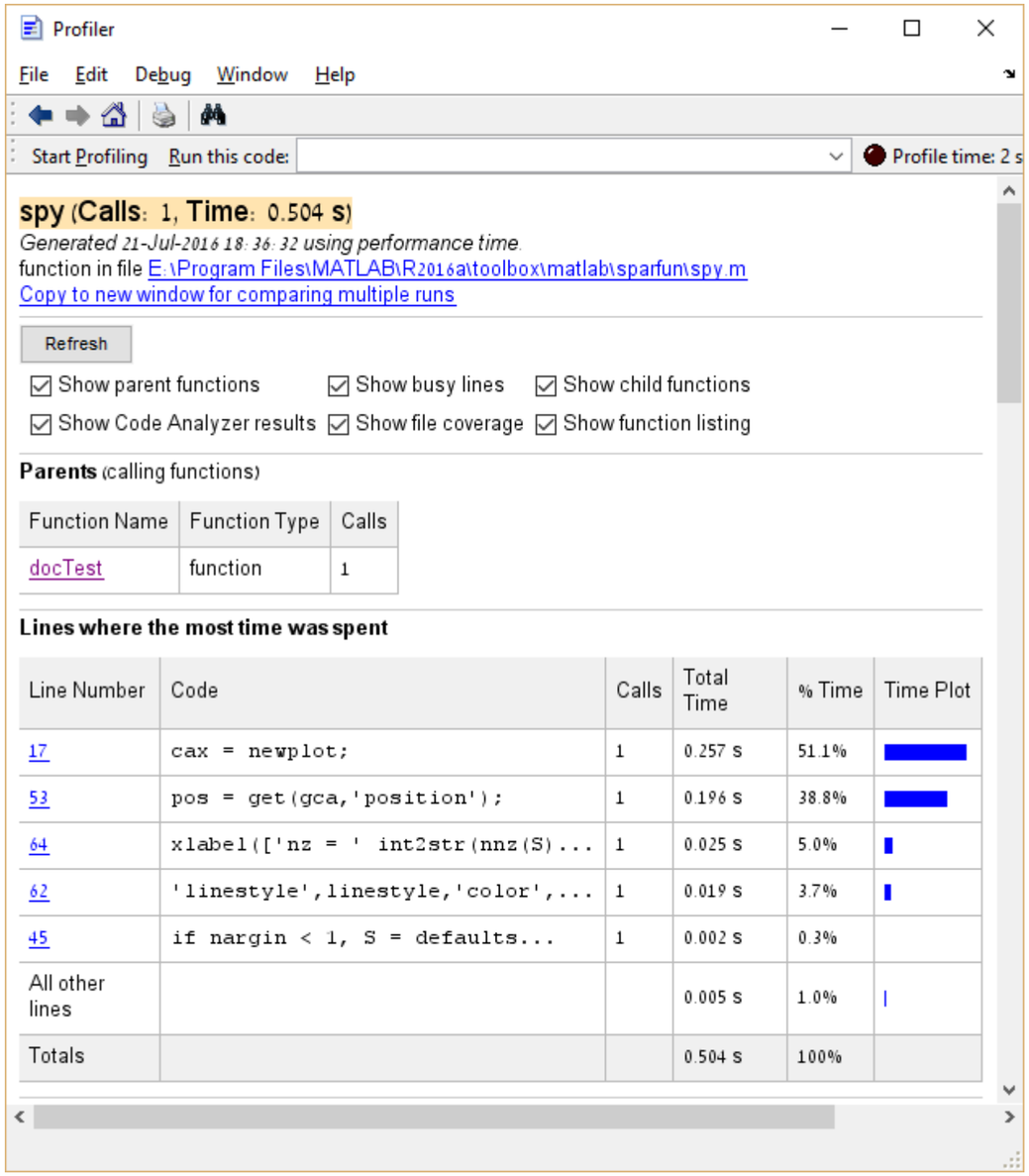
从上面可以看出，spy函数大约占用了总执行时间的25%。在“真实代码”的情况下，执行时间占比如此大的函数是优化的良好候选对象，而类似于`var`和`cla`的函数则应避免优化。

此外，可以点击Function Name列中的条目，查看该条目的执行时间详细分解。以下是点击spy的示例：

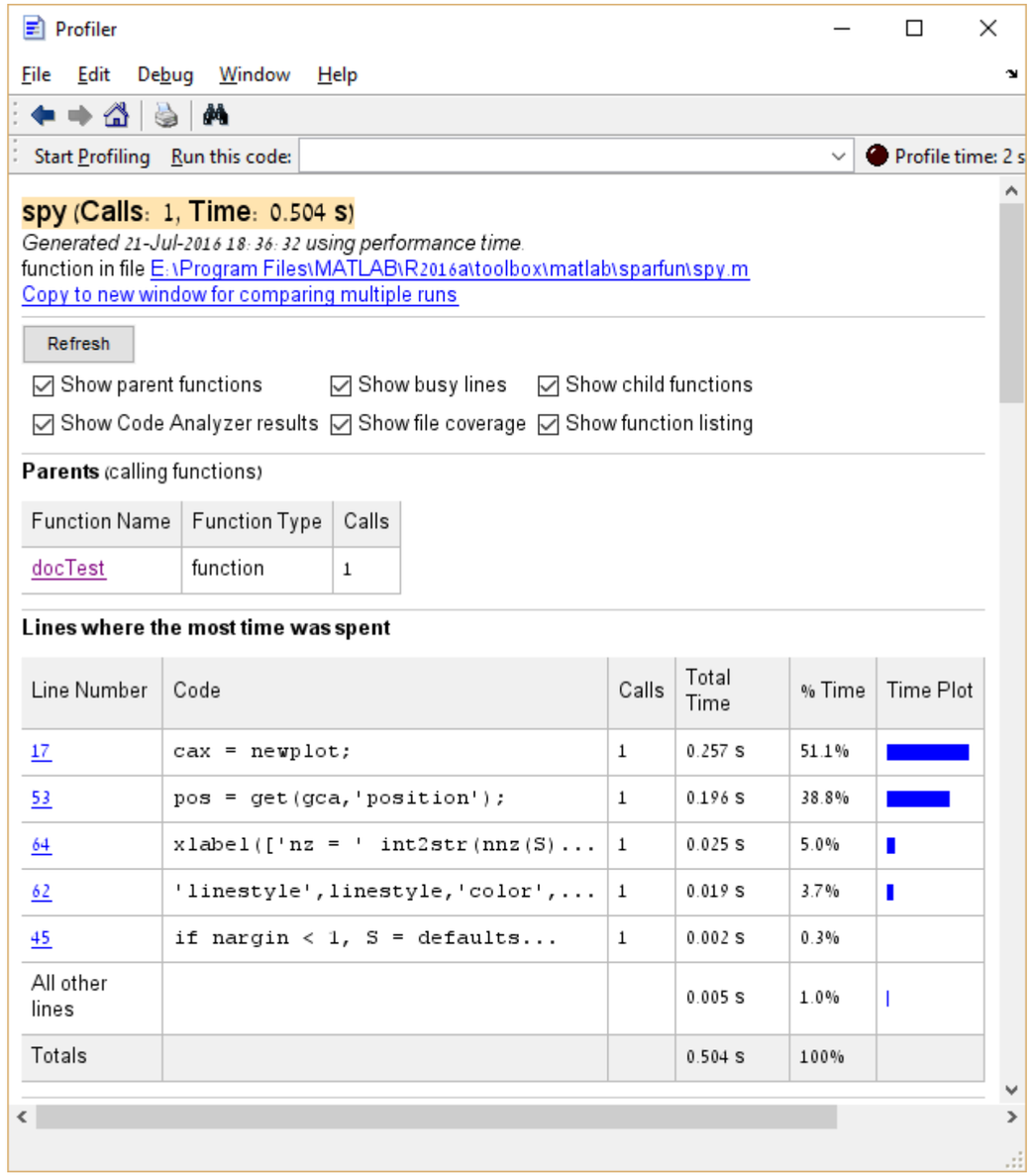


From the above we learn that the spy function takes about 25% of the total execution time. In the case of "real code", a function that takes such a large percentage of execution time would be a good candidate for optimization, as opposed to functions analogous to `var` and `cla` whose optimization should be avoided.

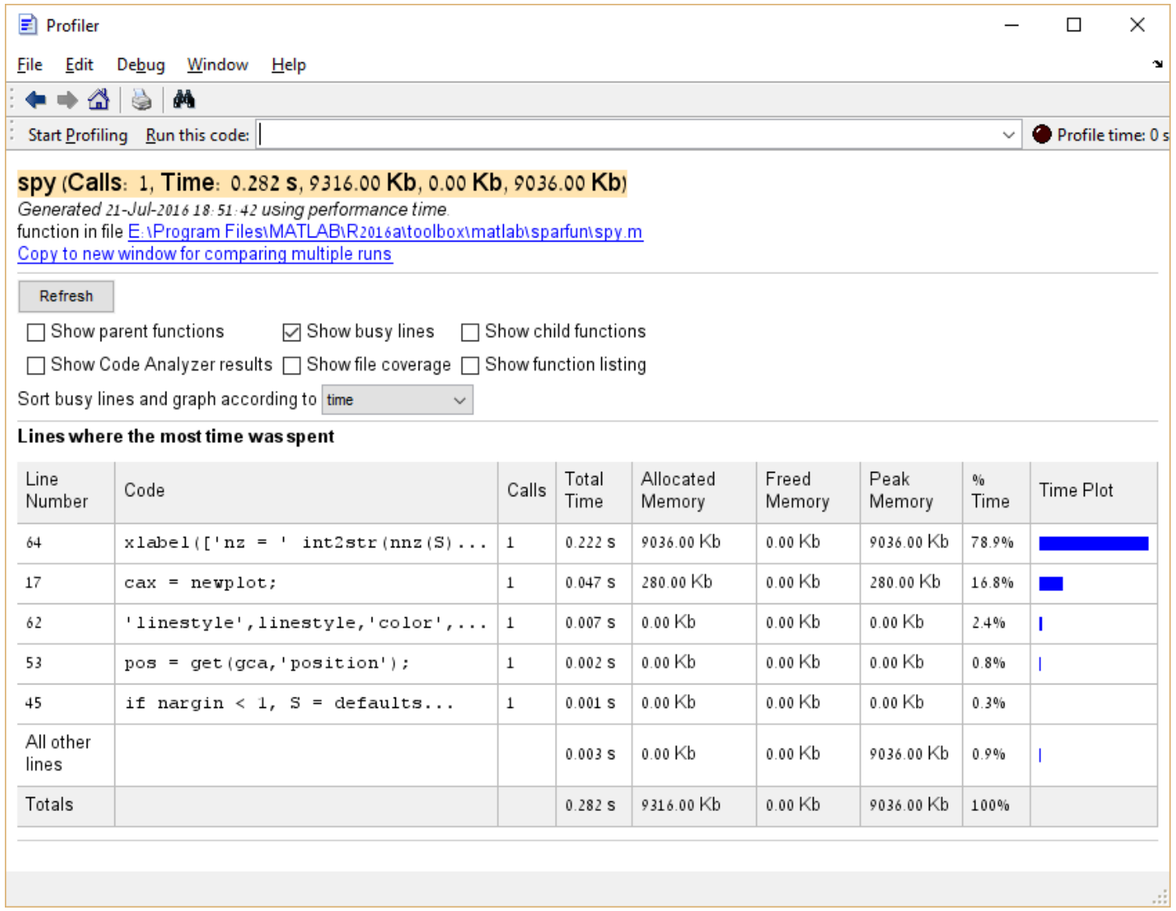
Moreover, it is possible to click on entries in the *Function Name* column to see a detailed breakdown of execution time for that entry. Here's the example of clicking spy:



还可以通过在运行分析器之前执行profile('-memory')来分析内存消耗。



It is also possible to profile memory consumption by executing `profile('-memory')` before running the Profiler.



Line Number	Code	Calls	Total Time	Allocated Memory	Freed Memory	Peak Memory	% Time	Time Plot
64	xlabel(['nz = ' int2str(nnz(S)...	1	0.222 s	9036.00 Kb	0.00 Kb	9036.00 Kb	78.9%	
17	cax = newplot;	1	0.047 s	280.00 Kb	0.00 Kb	280.00 Kb	16.8%	
62	'linestyle',linestyle,'color',...	1	0.007 s	0.00 Kb	0.00 Kb	0.00 Kb	2.4%	
53	pos = get(gca,'position');	1	0.002 s	0.00 Kb	0.00 Kb	0.00 Kb	0.8%	
45	if nargin < 1, S = defaults...	1	0.001 s	0.00 Kb	0.00 Kb	0.00 Kb	0.3%	
All other lines			0.003 s	0.00 Kb	0.00 Kb	9036.00 Kb	0.9%	
Totals			0.282 s	9316.00 Kb	0.00 Kb	9036.00 Kb	100%	

第26.2节：比较多个函数的执行时间

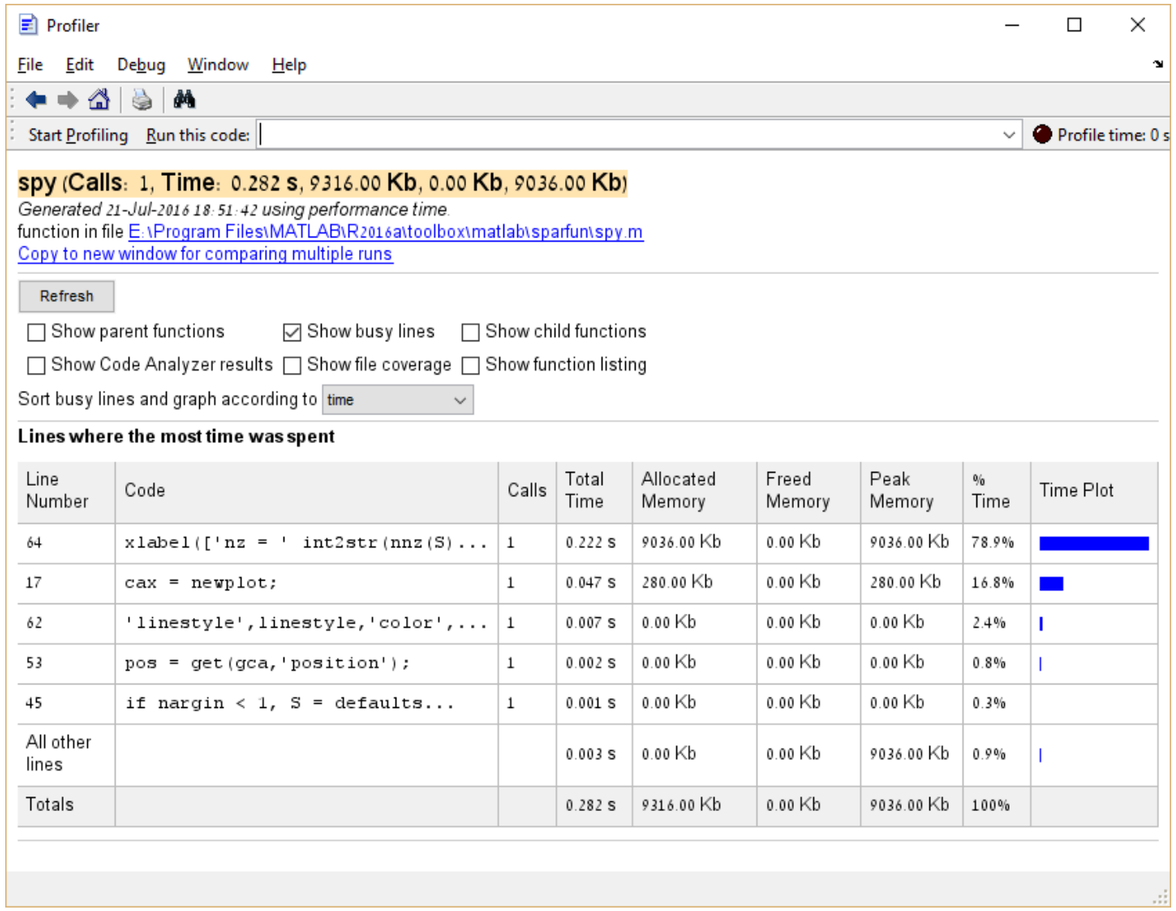
广泛使用的组合 tic 和 toc 可以大致了解函数或代码片段的执行时间。

但不应将其用于比较多个函数。为什么？几乎不可能为所有代码片段在脚本中使用上述方法提供相同的条件进行比较。也许这些函数共享相同的函数空间和公共变量，因此后续调用的函数和代码片段已经利用了先前初始化的变量和函数。此外，也无法判断JIT编译器是否会对这些后续调用的代码片段进行同等处理。

专门用于基准测试的函数是 timeit。以下示例说明了它的用法。

有数组 A 和矩阵 B。应通过计算不同元素的数量来确定 B 的哪一行与 A 最相似。

```
函数 t = bench()
    A = [0 1 1 1 0 0];
    B = perms(A);

    % 要比较的函数
    fcns = {
@() compare1(A,B);
        @() compare2(A,B);
        @() compare3(A,B);
```

Line Number	Code	Calls	Total Time	Allocated Memory	Freed Memory	Peak Memory	% Time	Time Plot
64	xlabel(['nz = ' int2str(nnz(S)...	1	0.222 s	9036.00 Kb	0.00 Kb	9036.00 Kb	78.9%	
17	cax = newplot;	1	0.047 s	280.00 Kb	0.00 Kb	280.00 Kb	16.8%	
62	'linestyle',linestyle,'color',...	1	0.007 s	0.00 Kb	0.00 Kb	0.00 Kb	2.4%	
53	pos = get(gca,'position');	1	0.002 s	0.00 Kb	0.00 Kb	0.00 Kb	0.8%	
45	if nargin < 1, S = defaults...	1	0.001 s	0.00 Kb	0.00 Kb	0.00 Kb	0.3%	
All other lines			0.003 s	0.00 Kb	0.00 Kb	9036.00 Kb	0.9%	
Totals			0.282 s	9316.00 Kb	0.00 Kb	9036.00 Kb	100%	

Section 26.2: Comparing execution time of multiple functions

The widely used combination of **tic** and **toc** can provide a rough idea of the execution time of a function or code snippets.

For comparing several functions it shouldn't be used. Why? It is almost impossible to provide *equal conditions* for all code snippets to compare within a script using above solution. Maybe the functions share the same function space and common variables, so later called functions and code snippets already take advantage of previously initialized variables and functions. Also the there is no insight whether the JIT compiler would handle these subsequently called snippets equally.

The dedicated function for benchmarks is **timeit**. The following example illustrates its use.

There are the array `A` and the matrix `B`. It should be determined which row of `B` is the most similar to `A` by counting the number of different elements.

```
function t = bench()
    A = [0 1 1 1 0 0];
    B = perms(A);

    % functions to compare
    fcns = {
        @() compare1(A,B);
        @() compare2(A,B);
        @() compare3(A,B);
```

```
@() compare4(A,B);
    };

    % timeit
t = cellfun(@timeit, fcns);
end

function Z = compare1(A,B)
    Z = sum(  bsxfun(@eq,  A,B) , 2);
end
function Z = compare2(A,B)
    Z = sum(bsxfun(@xor, A, B),2);
end
function Z = compare3(A,B)
    A = logical(A);
Z = sum(B(:,~A),2) + sum(~B(:,A),2);
end
function Z = compare4(A,B)
     Z = pdist2( A, B, 'hamming', 'Smallest', 1 );
end
```

这种基准测试方法首次见于this answer。

第26.3节：预分配的重要性

MATLAB中的数组以连续的内存块形式存储，由MATLAB自动分配和释放。
MATLAB将数组大小调整等内存管理操作隐藏在易用的语法背后：

```
a = 1:4

a =

     1     2     3     4

a(5) = 10  % 或者也可以写成 a = [a, 10]

a =

     1     2     3     4    10
```

重要的是要理解，上述操作并非简单的操作，a(5) = 10 会导致 MATLAB 分配一个大小为 5 的新内存块，复制前 4 个数字，并将第 5 个设置为 10。这是一个 O(numel(a))操作，而不是 O(1)。

考虑以下情况：

```
clear all
n=12345678;
a=0;
tic
for i = 2:n
a(i) = sqrt(a(i-1)) + i;
end
toc

耗时为 3.004213 秒。
```

a 在此循环中被重新分配了 n 次（不包括 MATLAB 进行的一些优化）！注意 MATLAB 给出了警告：


```
        @() compare4(A,B);
    };

    % timeit
    t = cellfun(@timeit, fcns);
end

function Z = compare1(A,B)
    Z = sum(  bsxfun(@eq,  A,B) , 2);
end
function Z = compare2(A,B)
    Z = sum(bsxfun(@xor, A, B),2);
end
function Z = compare3(A,B)
    A = logical(A);
    Z = sum(B(:,~A),2) + sum(~B(:,A),2);
end
function Z = compare4(A,B)
     Z = pdist2( A, B, 'hamming', 'Smallest', 1 );
end
```

This way of benchmark was first seen in this answer.

Section 26.3: The importance of preallocation

Arrays in MATLAB are held as continuous blocks in memory, allocated and released automatically by MATLAB.
MATLAB hides memory management operations such as resizing of an array behind easy to use syntax:

```
a = 1:4

a =

     1     2     3     4

a(5) = 10  % or alternatively a = [a, 10]

a =

     1     2     3     4    10
```

It is important to understand that the above is not a trivial operation, `a(5) = 10` will cause MATLAB to allocate a new block of memory of size 5, copy the first 4 numbers over, and set the 5'th to 10. That's a `O(numel(a))` operation, and not `O(1)`.

Consider the following:

```
clear all
n=12345678;
a=0;
tic
for i = 2:n
    a(i) = sqrt(a(i-1)) + i;
end
toc

Elapsed time is 3.004213 seconds.
```

a is reallocated n times in this loop (excluding some optimizations undertaken by MATLAB)! Note that MATLAB gives us a warning:

> 变量 'a' 似乎在每次循环迭代时大小都会变化。考虑预分配以提高速度。

预分配时会发生什么？

```
a=zeros(1,n);
tic
for i = 2:n
a(i) = sqrt(a(i-1)) + i;
end
toc

耗时 为0.410531 秒。
```

我们可以看到运行时间减少了一个数量级。

预分配的方法：

MATLAB 提供了多种用于分配向量和矩阵的函数，具体取决于用户的需求
这些包括：zeros、ones、nan、eye、true 等。

```
a = zeros(3)      % 分配一个 3×3 的矩阵，初始化为 0
a =

     0     0     0
     0     0     0
     0     0     0

a = zeros(3, 2)     % 分配一个 3×2 的矩阵，初始化为 0
a =

     0     0
     0     0
     0     0

a = ones(2, 3, 2)      % 分配一个初始化为1的三维数组（2×3×2）
a(:,:,1) =

     1     1     1
     1     1     1


a(:,:,2) =

     1     1     1
     1     1     1

a = ones(1, 3) * 7  % 分配一个长度为3的行向量，初始化为7
a =

     7     7     7
```

还可以指定数据类型：

```
a = zeros(2, 1, 'uint8');  % 分配一个uint8类型的数组
```

也很容易克隆现有数组的大小：

```
a = ones(3, 4);        % a是一个3×4的全1矩阵
```


> "The variable 'a' appears to change size on every loop iteration. Consider preallocating for speed."

What happens when we preallocate?

```
a=zeros(1,n);
tic
for i = 2:n
    a(i) = sqrt(a(i-1)) + i;
end
toc

Elapsed time is 0.410531 seconds.
```

We can see the runtime is reduced by an order of magnitude.

Methods for preallocation:

MATLAB provides various functions for allocation of vectors and matrices, depending on the specific requirements of the user. These include: zeros, ones, nan, eye, true etc.

```
a = zeros(3)       % Allocates a 3-by-3 matrix initialized to 0
a =

     0     0     0
     0     0     0
     0     0     0

a = zeros(3, 2)     % Allocates a 3-by-2 matrix initialized to 0
a =

     0     0
     0     0
     0     0

a = ones(2, 3, 2)      % Allocates a 3 dimensional array (2-by-3-by-2) initialized to 1
a(:,:,1) =

     1     1     1
     1     1     1


a(:,:,2) =

     1     1     1
     1     1     1

a = ones(1, 3) * 7  % Allocates a row vector of length 3 initialized to 7
a =

     7     7     7
```

A data type can also be specified:

```
a = zeros(2, 1, 'uint8');  % allocates an array of type uint8
```

It is also easy to clone the size of an existing array:

```
a = ones(3, 4);       % a is a 3-by-4 matrix of 1's
```

```
b = zeros(size(a));  % b 是一个3行4列的全零矩阵
```

并克隆类型：

```
a = ones(3, 4, 'single');       % a 是一个 3 行 4 列的 single 类型矩阵
b = zeros(2, 'like', a);        % b 是一个 2 行 2 列的 single 类型矩阵
```

注意 'like' 也会克隆complexity和 sparsity。

预分配是通过使用任何返回最终所需大小数组的函数隐式实现的，例如 rand、gallery、kron、bsxfun、colon 以及许多其他函数。例如，分配元素线性变化的向量的常用方法是使用冒号运算符（使用 2 个或 3 个操作数的变体1）：

```
a = 1:3
a =

     1     2     3

a = 2:-3:-4
a =

     2-1    -4
```

单元数组可以使用 cell() 函数分配，方式与 zeros() 类似。

```
a = cell(2,3)
a =

    []    []    []
    []    []    []
```

请注意，单元数组通过保存指向单元内容在内存中位置的指针来工作。因此，所有预分配的技巧同样适用于各个单元数组元素。

进一步阅读：

- 官方 MATLAB 文档 关于 "预分配内存"。
- 官方 MATLAB 文档 关于 "MATLAB 如何分配内存"。
- 预分配性能 在 Undocumented matlab 上。
- 理解数组预分配 在 Loren on the Art of MATLAB

第 26.4 节：使用 `single` 是可以的！

概述：

MATLAB 中数值数组的默认数据类型是 double。 double 是一种 浮点数表示，该格式每个数值占用 8 字节（或 64 位）。在 some 情况下，例如仅处理整数或数值不稳定性不是紧迫问题时，可能不需要如此高的位深。因此，建议考虑使用 single 精度（或其他合适的 类型）的好处：

- 更快的执行时间（在 GPU 上尤为明显）。
- 内存消耗减半：在double因内存不足错误而失败的情况下可能成功；以文件形式存储时更紧凑。

将变量从任何支持的数据类型转换为single的方式如下：


```
b = zeros(size(a));  % b is a 3-by-4 matrix of 0's
```

And clone the type:

```
a = ones(3, 4, 'single');       % a is a 3-by-4 matrix of type single
b = zeros(2, 'like', a);        % b is a 2-by-2 matrix of type single
```

note that 'like' also clones *complexity* and *sparsity*.

Preallocation is implicitly achieved using any function that returns an array of the final required size, such as `rand`, `gallery`, `kron`, `bsxfun`, `colon` and many others. For example, a common way to allocate vectors with linearly varying elements is by using the colon operator (with either the 2- or 3-operand variant1):

```
a = 1:3
a =

     1     2     3

a = 2:-3:-4
a =

     2    -1    -4
```

Cell arrays can be allocated using the `cell()` function in much the same way as `zeros()`.

```
a = cell(2,3)
a =

    []    []    []
    []    []    []
```

Note that cell arrays work by holding pointers to the locations in memory of cell contents. So all preallocation tips apply to the individual cell array elements as well.

Further reading:

- Official MATLAB documentation on "**Preallocating Memory**".
- Official MATLAB documentation on "**How MATLAB Allocates Memory**".
- Preallocation performance on **Undocumented matlab**.
- Understanding Array Preallocation on **Loren on the Art of MATLAB**

Section 26.4: It's ok to be `single`!

Overview:

The default data type for numeric arrays in MATLAB is `double`. `double` is a floating point representation of numbers, and this format takes 8 bytes (or 64 bits) per value. In **some** cases, where e.g. dealing only with integers or when numerical instability is not an imminent issue, such high bit depth may not be required. For this reason, it is advised to consider the benefits of `single` precision (or other appropriate types):

- Faster execution time (especially noticeable on GPUs).
- Half the memory consumption: may succeed where `double` fails due to an out-of-memory error; more compact when storing as files.

Converting a variable from any supported data type to `single` is done using:

```
sing_var = single(var);
```

一些常用函数（例如：zeros、eye、ones等）默认输出为double类型，允许指定输出的类型/类。

在脚本中将变量转换为非默认精度/类型/类：

截至2016年7月，没有文档记录的方法可以将MATLAB的default数据类型从double更改。

在MATLAB中，新变量通常会模仿创建它们时所用变量的数据类型。为说明这一点，
请考虑以下示例：

```
A = magic(3);
B = diag(A);
C = 20*B;
>> whos C
  名称      大小            字节  类别     属性
  C         3x1               24  双精度
```

```
A = single(magic(3)); % A 被转换为 "single"
B = diag(A);
C = B*double(20);     % 更严格的类型，在本例中是“single”，占主导地位
D = single(size(C));  % 通常建议显式转换为所需类型。
>> whos C
  名称      大小            字节  类别     属性
  C         3x1               12  单精度
```

因此，似乎只需将几个初始变量进行类型转换/转换，就能使更改渗透到整个代码中——但这是不推荐的（见下文的注意事项与陷阱）。

注意事项与陷阱：

1. 由于引入了数值噪声（从单精度转换时），不建议重复转换
   转换为double类型）或信息丢失（当从double转换为single，或在某些整数类型之间转换时），例如
   ：

   ```
   double(single(1.2)) == double(1.2)
   ans =
        0
   ```

   这可以通过使用类型转换在一定程度上缓解。另请参见注意浮点数不准确性。

2. 仅依赖隐式数据类型（即 MATLAB 猜测计算输出的类型）是不推荐的，因为可能会产生以下不良影响：

   - 信息丢失：当预期得到 double 结果时，但不小心将 single 和 double 操作数混合，导致结果为 single 精度。
   - *意外的高内存消耗：当预期得到 single 结果时，但不小心计算导致输出为 double。*
   - 使用 GPU 时的不必要开销：当混合使用 gpuArray 类型（即存储在显存中的变量）和非 gpuArray 变量（即通常存储在内存中的变量）时，数据必须在计算前进行单向传输。此操作耗时，在重复计算中尤为明显。


```
sing_var = single(var);
```

Some commonly used functions (such as: zeros, eye, ones, etc.) that output double values by default, allow specifying the type/class of the output.

Converting variables in a script to a non-default precision/type/class:

As of July 2016, there exists no documented way to change the *default* MATLAB data type from double.

In MATLAB, new variables usually mimic the data types of variables used when creating them. To illustrate this, consider the following example:

```
A = magic(3);
B = diag(A);
C = 20*B;
>> whos C
  Name      Size            Bytes  Class     Attributes
  C         3x1                24  double
```

```
A = single(magic(3)); % A is converted to "single"
B = diag(A);
C = B*double(20);     % The stricter type, which in this case is "single", prevails
D = single(size(C));  % It is generally advised to cast to the desired type explicitly.
>> whos C
  Name      Size            Bytes  Class     Attributes
  C         3x1                12  single
```

Thus, it may seem sufficient to cast/convert several initial variables to have the change permeate throughout the code - however this is **discouraged** (see *Caveats & Pitfalls* below).

Caveats & Pitfalls:

1. Repeated conversions are **discouraged** due to the introduction of numeric noise (when casting from single to double) or loss of information (when casting from double to single, or between certain integer types), e.g.
   :

   ```
   double(single(1.2)) == double(1.2)
   ans =
        0
   ```

   This can be mitigated somewhat using typecast. See also Be aware of floating point inaccuracy.

2. Relying solely on implicit data-typing (i.e. what MATLAB guesses the type of the output of a computation should be) is **discouraged** due to several undesired effects that might arise:

   - *Loss of information*: when a double result is expected, but a careless combination of single and double operands yields single precision.
   - *Unexpectedly high memory consumption*: when a single result is expected but a careless computation results in a double output.
   - *Unnecessary overhead when working with GPUs*: when mixing gpuArray types (i.e. variables stored in VRAM) with non-gpuArray variables (i.e. those *usually* stored in RAM) the data will have to be transferred one way or the other before the computation can be performed. This operation takes time, and can be very noticeable in repetitive computations.

- 混合浮点类型和整数类型时的错误：像 mtimes（*）这样的函数不支持整数和浮点类型混合输入，会报错。像 times（.*）这样的函数完全不支持整数类型输入，也会报错。

```
>> ones(3, 3, 'int32') * ones(3, 3, 'int32')
使用 * 时出错
MTIMES 不完全支持整数类。至少一个输入必须是标量。

>> ones(3, 3, 'int32') .* ones(3, 3, 'double')
使用 .* 时出错
整数只能与相同类的整数或标量 double 结合。
```

为了更好的代码可读性和降低出现不期望类型的风险，建议采取防御性方法，即变量被显式地转换为所需类型。

另请参见：

- MATLAB 文档：浮点数。
- MathWorks 技术文章：将 MATLAB 代码转换为定点数的最佳实践。

- *Errors when mixing floating-point types with integer types*: functions like `mtimes` (*) are not defined for mixed inputs of integer and floating point types - and will error. Functions like `times` (.*) are not defined at all for integer-type inputs - and will again error.

```
>> ones(3,3,'int32')*ones(3,3,'int32')
Error using  *
MTIMES is not fully supported for integer classes. At least one input must be scalar.

>> ones(3,3,'int32').*ones(3,3,'double')
Error using  .*
Integers can only be combined with integers of the same class, or scalar doubles.
```

For better code readability and reduced risk of unwanted types, a defensive approach is **advised**, where variables are *explicitly* cast to the desired type.

See Also:

- MATLAB Documentation: Floating-Point Numbers.
- MathWorks' Technical Article: Best Practices for Converting MATLAB Code to Fixed Point.

第27章：多线程

第27.1节：使用 parfor 并行化循环

您可以使用parfor来并行执行循环的迭代：

示例：

```
poolobj = parpool(2);       % 打开一个包含2个工作线程的并行池

s = 0;                      % 执行一些并行计算
parfor i=0:9
s = s + 1;
end
disp(s)                     % 输出 '10'

delete(poolobj);            % 关闭并行池
```

注意：parfor 不能直接嵌套。要嵌套 parfor，请在第一个 parfor 中使用一个函数，并在该函数中添加第二个 parfor 。

示例：

```
parfor i = 1:n
[op] = fun_name(ip);
end

function [op] = fun_name(ip)
parfor j = 1:length(ip)
% 一些计算
end
```

第27.2节：使用“单程序，多数据”（SPMD）语句并行执行命令

与将循环迭代分配给多个线程的并行for循环（parfor）不同，单程序多数据（spmd）语句将一系列命令分发给所有线程，使每个线程执行该命令并存储结果。考虑以下示例：

```
poolobj = parpool(2);    % 打开一个包含两个工作线程的并行池

spmd
q = rand(3);         % 每个线程生成一个独特的3x3随机数数组
end

q{1}               % q像单元格向量一样调用
q{2}               % 可以通过索引访问每个线程中存储的值

delete(poolobj)  % 如果关闭池，则无法再访问q中的数据
```

需要注意的是，可以在spmd块中通过线程索引（也称为实验室索引，或labindex）访问每个线程：

```
poolobj = parpool(2);        % 打开一个包含两个工作线程的并行池

spmd
```


Chapter 27: Multithreading

Section 27.1: Using parfor to parallelize a loop

You can use `parfor` to execute the iterations of a loop in parallel:

Example:

```
poolobj = parpool(2);       % Open a parallel pool with 2 workers

s = 0;                      % Performing some parallel Computations
parfor i=0:9
    s = s + 1;
end
disp(s)                     % Outputs '10'

delete(poolobj);            % Close the parallel pool
```

Note: `parfor` cannot be nested directly. For `parfor` nesting use a function in first `parfor` and add second `parfor` in that function.

Example:

```
parfor i = 1:n
[op] = fun_name(ip);
end

function [op] = fun_name(ip)
parfor j = 1:length(ip)
% Some Computation
end
```

Section 27.2: Executing commands in parallel using a "Single Program, Multiple Data" (SPMD) statement

Unlike a parallel for-loop (`parfor`), which takes the iterations of a loop and distributes them among multiple threads, a single program, multiple data (`spmd`) statement takes a series of commands and distributes them to **all** the threads, so that each thread performs the command and stores the results. Consider this:

```
poolobj = parpool(2);    % open a parallel pool with two workers

spmd
    q = rand(3);         % each thread generates a unique 3x3 array of random numbers
end

q{1}             % q is called like a cell vector
q{2}             % the values stored in each thread may be accessed by their index

delete(poolobj)  % if the pool is closed, then the data in q will no longer be accessible
```

It is important to note that each thread may be accessed during the `spmd` block by its thread index (also called lab index, or `labindex`):

```
poolobj = parpool(2);        % open a parallel pool with two workers

spmd
```

```
q = rand(labindex + 1);  % 每个线程生成一组唯一的随机数数组
end

size(q{1})                    % q{1} 的大小是 2x2
size(q{2})                    % q{2} 的大小是 3x3

delete(poolobj)               % q 不再可访问
```

在这两个例子中，q 是一个复合对象，可以用命令 q = Composite() 初始化。需要注意的是，复合对象仅在池运行时可访问。

第27.3节：使用 batch 命令进行各种并行计算

要在 MATLAB 中使用多线程，可以使用 batch 命令。请注意，必须安装并行计算工具箱。

对于一个耗时的脚本，例如，

```
for ii=1:1e8
A(ii)=sin(ii*2*pi/1e8);
end
```

要以批处理模式运行它，可以使用以下命令：

```
job=batch("da")
```

这使得 MATLAB 能够以批处理模式运行，并且可以在此期间使用 MATLAB 执行其他操作，例如添加更多批处理进程。

完成作业后检索结果并将数组A加载到工作区：

```
load(job, 'A')
```

最后，从主页→环境→并行→监视作业打开“监视作业 GUI”，并通过以下方式删除作业：

```
delete(job)
```

要加载用于批处理的函数，只需使用此语句，其中fcn是函数名，N是输出数组的数量，x1、...、xn是输入数组：

```
j=batch(fcn, N, {x1, x2, ..., xn})
```

第27.4节：何时使用parfor

基本上，parfor推荐在两种情况下使用：循环中有大量迭代（例如，像1e10），或者每次迭代耗时很长（例如，eig(magic(1e4))）。在第二种情况下，你可能想考虑使用spmd。之所以parfor在短范围或快速迭代时比for循环慢，是因为需要管理所有工作线程的开销，而不仅仅是执行计算。

此外，许多函数内置了隐式多线程，使得在使用这些函数时，parfor循环并不比串行for循环更高效，因为所有核心已经被使用。在这种情况下，parfor实际上会带来负面影响，因为它有分配开销，而并行程度与所使用的函数相同。


```
    q = rand(labindex + 1);  % each thread generates a unique array of random numbers
end

size(q{1})                    % the size of q{1} is 2x2
size(q{2})                    % the size of q{2} is 3x3

delete(poolobj)               % q is no longer accessible
```

In both examples, `q` is a composite object, which may be initialized with the command `q = Composite()`. It is important to note that composite objects are only accessible while the pool is running.

Section 27.3: Using the batch command to do various computations in parallel

To use multi-threading in MATLAB one can use the `batch` command. Note that you must have the Parallel Computing toolbox installed.

For a time-consuming script, for example,

```
for ii=1:1e8
   A(ii)=sin(ii*2*pi/1e8);
end
```

to run it in batch mode one would use the following:

```
job=batch("da")
```

which is enables MATLAB to run in batch mode and makes it possible to use MATLAB in the meantime to do other things, such as add more batch processes.

To retrieve the results after finishing the job and load the array `A` into the workspace:

```
load(job, 'A')
```

Finally, open the "monitor job gui" from *Home → Environment → Parallel → Monitor jobs* and delete the job through:

```
delete(job)
```

To load a function for batch processing, simply use this statement where `fcn` is the function name, `N` is number of output arrays and `x1, ..., xn` are input arrays:

```
 j=batch(fcn, N, {x1, x2, ..., xn})
```

Section 27.4: When to use parfor

Basically, `parfor` is recommended in two cases: lots of iterations in your loop (i.e., like `1e10`), or if each iteration takes a very long time (e.g., `eig(magic(1e4))`). In the second case you might want to consider using `spmd` . The reason `parfor` is slower than a `for` loop for short ranges or fast iterations is the overhead needed to manage all workers correctly, as opposed to just doing the calculation.

Also a lot of functions have implicit multi-threading built-in, making a `parfor` loop not more efficient, when using these functions, than a serial `for` loop, since all cores are already being used. `parfor` will actually be a detriment in this case, since it has the allocation overhead, whilst being as parallel as the function you are trying to use.

考虑以下示例以观察for与parfor的行为差异。首先，如果尚未打开并行池，请打开：

```
gcp; % 使用当前设置打开并行池
```

然后执行几个大型循环：

```
n = 1000; % 迭代次数
EigenValues = cell(n,1); % 准备存储数据
Time = zeros(n,1);
for ii = 1:n
tic
EigenValues{ii,1} = eig(magic(1e3)); % 如果运行时间过长，可以考虑减小magic的大小
Time(ii,1) = toc; % 记录每次迭代后的时间
end

figure; % 创建结果图
plot(1:n,t)
title '每次迭代时间'
ylabel '时间 [秒]'
xlabel '迭代次数[-]';
```

然后用parfor替代for执行相同操作。你会注意到每次迭代的平均时间增加了。但请注意，parfor使用了所有可用的工作线程，因此总时间（sum(Time)）需要除以
您计算机中的核心数量。

因此，虽然使用parfor相比使用for每次单独迭代的时间增加了，但总时间却大幅减少。

Consider the following example to see the behaviour of `for` as opposed to that of `parfor`. First open the parallel pool if you've not already done so:

```
gcp; % Opens a parallel pool using your current settings
```

Then execute a couple of large loops:

```
n = 1000; % Iteration number
EigenValues = cell(n,1); % Prepare to store the data
Time = zeros(n,1);
for ii = 1:n
tic
    EigenValues{ii,1} = eig(magic(1e3)); % Might want to lower the magic if it takes too long
Time(ii,1) = toc; % Collect time after each iteration
end

figure; % Create a plot of results
plot(1:n,t)
title 'Time per iteration'
ylabel 'Time [s]'
xlabel 'Iteration number[-]';
```

Then do the same with `parfor` instead of `for`. You will notice that the average time per iteration goes up. Do realise however that the `parfor` used all available workers, thus the total time (`sum(Time)`) has to be divided by the number of cores in your computer.

So, whilst the time to do each separate iteration goes up using `parfor` with respect to using `for`, the total time goes down considerably.

第28章：使用串口

串口参数	功能说明
波特率	设置波特率。如今最常见的是57600，但4800、9600和115200也经常见到。
输入缓冲区大小	内存中保存的字节数。MATLAB有一个先进先出（FIFO）机制，这意味着新字节将被丢弃。默认是512字节，但可以轻松设置到20MB而不会有问题。只有在少数极端情况下，用户才希望这个值较小
可用字节数	等待读取的字节数
ValuesSent	自端口打开以来发送的字节数
ValuesReceived	自端口打开以来读取的字节数
BytesAvailableFcn	指定当输入缓冲区中有指定数量的字节可用，或读取到终止符时要执行的回调函数
BytesAvailableFcnCount	指定输入缓冲区中必须可用的字节数，以触发 bytes-available 事件
BytesAvailableFcnMode	指定bytes-available 事件是在输入缓冲区中有指定数量的字节可用后生成，还是在读取到终止符后生成输入缓冲区中有字节可用，或读取到终止符后生成

串口是与外部传感器或嵌入式系统（如Arduino）通信的常见接口。现代串行通信通常通过使用USB转串口适配器的USB连接来实现。
MATLAB 提供了用于串行通信的内置函数，包括 RS-232 和 RS-485 协议。这些函数可用于硬件串口或“虚拟”USB-串口连接。这里的示例演示了 MATLAB 中的串行通信。

第 28.1 节：在 Mac/Linux/Windows 上创建串口

```
% 定义波特率为 115200 的串口
rate = 115200;
if 是pc
s = serial('COM1', 'BaudRate',rate);
elseif 是mac
    % 注意在 OSX 上串口设备是唯一枚举的。你需要% 查看 /dev/tty.* 来确定你的串口设备的
    确切标识

s = serial('/dev/tty.usbserial-A104VFT7', 'BaudRate',rate);
elseif 是unix
s = serial('/dev/ttyusb0', 'BaudRate',rate);
end

% 将输入缓冲区大小设置为 1,000,000 字节（默认：512 字节）。
s.InputBufferSize = 0000000;

% 打开串口
fopen(s);
```

第28.2节：选择您的通信模式

MATLAB支持与串口的同步和异步通信。选择正确的通信模式非常重要。选择将取决于：

- 您所通信的仪器的行为方式。
- 您的主程序（或图形用户界面）除了管理串口外，还需要执行的其他功能。

我将定义3种不同的情况进行说明，从最简单到最复杂。对于这3个例子，

Chapter 28: Using serial ports

Serial port parameter	what it does
BaudRate	Sets the baudrate. The most common today is 57600, but 4800, 9600, and 115200 are frequently seen as well
InputBufferSize	The number of bytes kept in memory. MATLAB has a FIFO, which means that new bytes will be discarded. The default is 512 bytes, but it can easily be set to 20MB without issue. There are only a few edge cases where the user would want this to be small
BytesAvailable	The number of bytes waiting to be read
ValuesSent	The number of bytes sent since the port was opened
ValuesReceived	The number of bytes read since the port was opened
BytesAvailableFcn	Specify the callback function to execute when a specified number of bytes is available in the input buffer, or a terminator is read
BytesAvailableFcnCount	Specify the number of bytes that must be available in the input buffer to generate a bytes-available event
BytesAvailableFcnMode	Specify if the bytes-available event is generated after a specified number of bytes is available in the input buffer, or after a terminator is read

Serial ports are a common interface for communicating with external sensors or embedded systems such as Arduinos. Modern serial communications are often implemented over USB connections using USB-serial adapters. MATLAB provides built-in functions for serial communications, including RS-232 and RS-485 protocols. These functions can be used for hardware serial ports or "virtual" USB-serial connections. The examples here illustrate serial communications in MATLAB.

Section 28.1: Creating a serial port on Mac/Linux/Windows

```
% Define serial port with a baud rate of 115200
rate = 115200;
if ispc
    s = serial('COM1', 'BaudRate',rate);
elseif ismac
    % Note that on OSX the serial device is uniquely enumerated. You will
    % have to look at /dev/tty.* to discover the exact signature of your
    % serial device
    s = serial('/dev/tty.usbserial-A104VFT7', 'BaudRate',rate);
elseif isunix
    s = serial('/dev/ttyusb0', 'BaudRate',rate);
end

% Set the input buffer size to 1,000,000 bytes (default: 512 bytes).
s.InputBufferSize = 1000000;

% Open serial port
fopen(s);
```

Section 28.2: Choosing your communication mode

MATLAB supports *synchronous* and *asynchronous* communication with a serial port. It is important to choose the right communication mode. The choice will depend on:

- how the instrument you are communicating with behave.
- what other functions your main program (or GUI) will have to do aside from managing the serial port.

I'll define 3 different cases to illustrate, from the simplest to the most demanding. For the 3 examples, the

我连接的仪器是一个带有倾斜仪的电路板，它可以在下面描述的3种模式中工作。

模式1：同步（主/从）

这种模式是最简单的。它对应于PC为主机，仪器为从机的情况。
该仪器本身不会向串口发送任何信息，它只会在被主机（PC，您的程序）询问问题/发送命令后回复一个答案。例如：

- 电脑发送命令："现在给我一个测量值"仪器接收命令，进行测
- 量，然后将测量值发送回串口："倾斜仪的值是XXX"。

或者

- 电脑发送命令："从模式X切换到模式Y"
- 仪器接收命令，执行命令，然后向串口发送确认信息：
  "命令已执行"（或"命令未执行"）。这通常称为ACK/NACK回复（表示"已确认"/"未确认"）。

总结：在此模式下，仪器（从设备）仅在被电脑（主设备）请求后立即向串口发送数据

模式2：异步

现在假设我启动了仪器，但它不仅仅是一个简单的传感器。它会持续监测自身的倾斜度，只要保持垂直（在容差范围内，比如±15度），它保持静默。如果设备倾斜超过15度，接近水平位置，它会向串口发送报警信息，紧接着发送倾斜度读数。只要倾斜度超过阈值，它每5秒继续发送一次倾斜度读数。

如果你的主程序（或图形界面）一直"等待"串口消息的到来，它可以很好地完成这项工作.....但与此同时无法执行其他操作。如果主程序是图形界面，界面看似"冻结"，无法接受用户输入，这会非常令人沮丧。实际上，它变成了从设备，而仪器是主设备。除非你有高级方法从仪器控制图形界面，否则应避免这种情况。幸运的是，异步通信模式允许你：

- 定义一个独立的函数，告诉程序在接收到消息时该做什么

instrument I am connecting to is a circuit board with an inclinometer, which can work in the 3 modes I will be describing below.

Mode 1: Synchronous (Master/Slave)

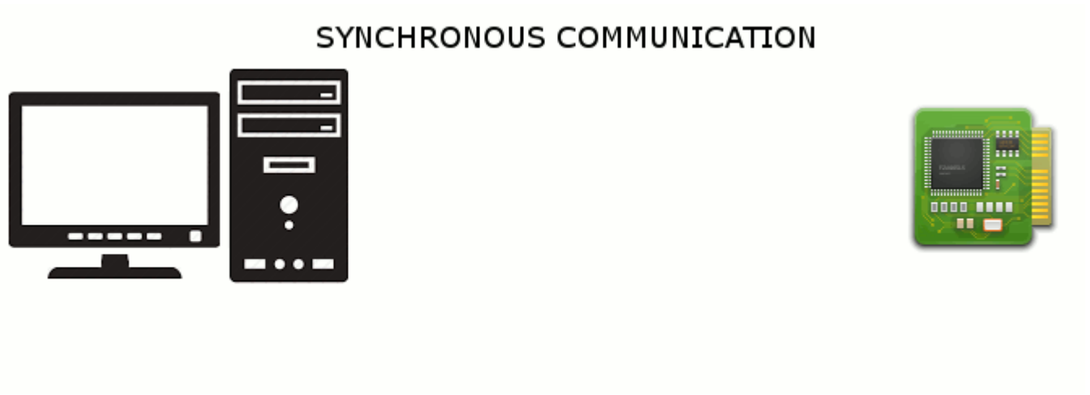
This mode is the simplest one. It correspond to the case where the PC is the *Master* and the instrument is the *slave*. The instrument does not send anything to the serial port on its own, it only **replies** an answer after being asked a question/command by the Master (the PC, your program). For example:

- The PC sends a command: "Give me a measurement now"
- The instrument receive the command, take the measurement then send back the measurement value to the serial line: "The inclinometer value is XXX".

OR

- The PC sends a command: "Change from mode X to mode Y"
- The instrument receive the command, execute it, then send a confirmation message back to the serial line: "*Command executed*" (or "*Command NOT executed*"). This is commonly called an ACK/NACK reply (for "Acknowledge(d)" / "NOT Acknowledged").

Summary: in this mode, the instrument (the *Slave*) only send data to the serial line **immediately after** having been asked by the PC (the *Master*)



Mode 2: Asynchronous

Now suppose I started my instrument, but it is more than just a dumb sensor. It constantly monitor's its own inclination and as long as it is vertical (within a tolerance, let's say +/-15 degrees), it stays silent. If the device is tilted by more than 15 degrees and get close to horizontal, it sends an alarm message to the serial line, immediately followed by a reading of the inclination. As long as the inclination is above the threshold, it continues to send an inclination reading every 5s.

If your main program (or GUI) is constantly "waiting" for message arriving on the serial line, it can do that well ... but it cannot do anything else in the meantime. If the main program is a GUI, it is highly frustrating to have a GUI seemingly "frozen" because it won't accept any input from the user. Essentially, it became the *Slave* and the instrument is the *Master*. Unless you have a fancy way of controlling your GUI from the instrument, this is something to avoid. Fortunately, the *asynchronous* communication mode will let you:

- define a separate function which tells your program what to do when a message is received

- 将此函数保留在一角，只有当串行线上有消息到达时才会被调用和执行。
  其余时间，图形用户界面可以执行它需要运行的任何其他代码。

摘要： 在此模式下，仪器可以随时向串行线发送消息（但不一定是所有时间）。电脑不会永久等待消息处理。它可以运行任何其他代码。只有当消息到达时，才会激活一个函数来读取并处理该消息。

模式3：流式传输（实时）

现在让我们释放我的仪器的全部威力。我将其设置为一个模式，使其不断向串行线发送测量数据。我的程序想接收这些数据包并将其显示在曲线或数字显示上。如果它像上面那样每5秒只发送一个值，那没问题，保持上述模式即可。但我的仪器在全速运行时以1000Hz的频率向串行线发送数据点，也就是说它每毫秒发送一个新值。如果我保持上述的异步模式，存在很高的风险（实际上是必然的）——我们定义的特殊函数会

处理每个新数据包将花费超过1毫秒的时间（如果你想绘图或显示数值，图形函数相当慢，甚至不考虑对信号进行滤波或傅里叶变换）。这意味着函数会开始执行，但在它完成之前，会有一个新数据包到达并再次触发该函数。第二个函数被放入执行队列，只有当第一个函数完成后才会开始执行……但此时已经有几个新数据包

到达后，每个都将一个要执行的函数放入队列中。你可以很快预见结果：当我正在绘制第5个点时，已经有数百个点等待绘制……界面变慢，最终冻结，堆栈增长，缓冲区填满，直到某处出问题。最终你会得到一个完全冻结的程序，或者干脆崩溃。

为了解决这个问题，我们将进一步断开电脑与仪器之间的同步连接。我们将让仪器以自己的节奏发送数据，而不是在每个数据包到达时立即触发一个功能。
串口缓冲区将不断累积接收到的数据包。电脑只会以其能够处理的速度（在电脑端设置的固定时间间隔）从缓冲区收集数据，处理这些数据（此时缓冲区由仪器继续填充），然后再从缓冲区收集新一批数据……如此循环。

摘要： 在此模式下，仪器连续发送数据，这些数据被串口缓冲区收集。在
定期间隔内，电脑从缓冲区收集数据并进行处理。电脑与仪器之间没有硬同步连接，
双方各自按照自己的时间执行任务。

- keep this function in a corner, it will only be called and executed **when a message arrives** on the serial line.
  The rest of the time the GUI can execute any other code it has to run.

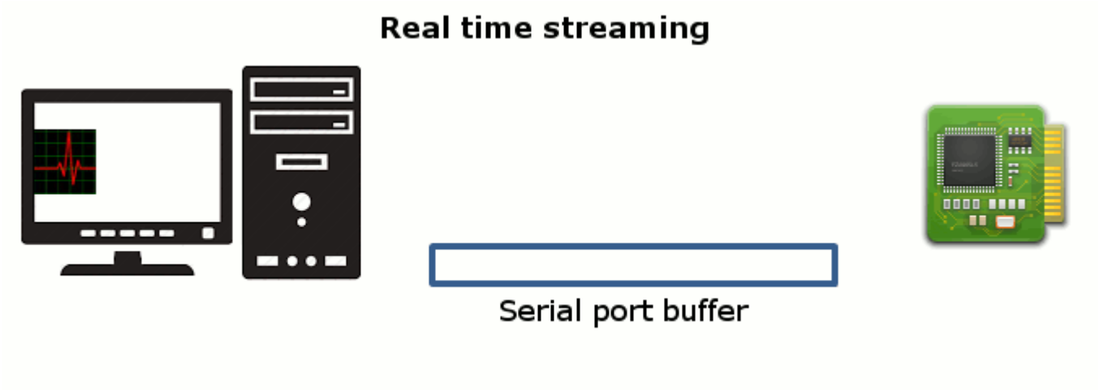
Summary: In this mode, the instrument may send message to the serial line at anytime (but not necessarily *all* the time). The PC does not *wait* permanently for a message to process. It is allowed to run any other code. Only when a message arrives, it activates a function which will then read and process this message.

Mode 3: Streaming (*Real time*)

Now let's unleash the full power of my instrument. I put it in a mode where it will constantly send measurements to the serial line. My program want to receive these packets and display that on a curve or a digital display. If it only send a value every 5s as above, no problem, keep the above mode. But my instrument at full whack sends a data point to the serial line at 1000Hz, i.e. it sends a new value every single millisecond. If I stay in the *asynchronous mode* described above, there is a high risk (actually a guaranteed certainty) that the special function we defined to process every new packet will take more than 1ms to execute (if you want to plot or display the value, graphic functions are quite slow, not even considering filtering or FFT'ing the signal). It means the function will start to execute, but before it finishes, a new packet will arrive and trigger the function again. The second function is placed in a queue for execution, and will only starts when the first one is done … but by this time a few new packets arrived and each placed a function to execute in the queue. You can quickly foresee the result: By the time I am plotting the 5th points, I have already hundreds waiting to be plotted too … the gui slows down, eventually freezes, the stack grows, the buffers fill up, until something gives. Eventually you are left with a completely frozen program or simply a crashed one.

To overcome this, we will disconnect even further the synchronisation link between the PC and the instrument. We will let the instrument send data at its own pace, without immediately triggering a function at each packet arrival. The serial port buffer will just accumulate the packets received. The PC will only collect data in the buffer at a pace it can manage (a regular interval, set up on the PC side), do something with it (while the buffer is getting refilled by the instrument), then collect a new batch of data from the buffer … and so on.

Summary: In this mode, the instrument sends data continuously, which are collected by the serial port buffer. At regular interval, the PC collect data from the buffer and do something with it. There is no hard synchronisation link between the PC and the instrument. Both execute their tasks on their own timing.



第28.3节：自动处理从串口接收的数据

通过串口连接的一些设备以恒定速率（流式数据）向您的程序发送数据，或者以不可预测的间隔发送数据。您可以配置串口，在数据到达时自动执行一个函数来处理数据。这称为串口对象的“回调函数”。

要使用此功能，必须设置串口的两个属性：您希望用于回调的函数名称（BytesAvailableFcn），以及触发执行回调函数的条件（BytesAvailableFcnMode）。

触发回调函数有两种方式：

1. 当串口接收到一定数量的字节时（通常用于二进制数据）
2. 当串口接收到某个特定字符时（通常用于文本或ASCII数据）

回调函数有两个必需的输入参数，称为obj和event。 obj是串口。例如，如果您想打印从串口接收到的数据，可以定义一个用于打印数据的函数，称为newdata：

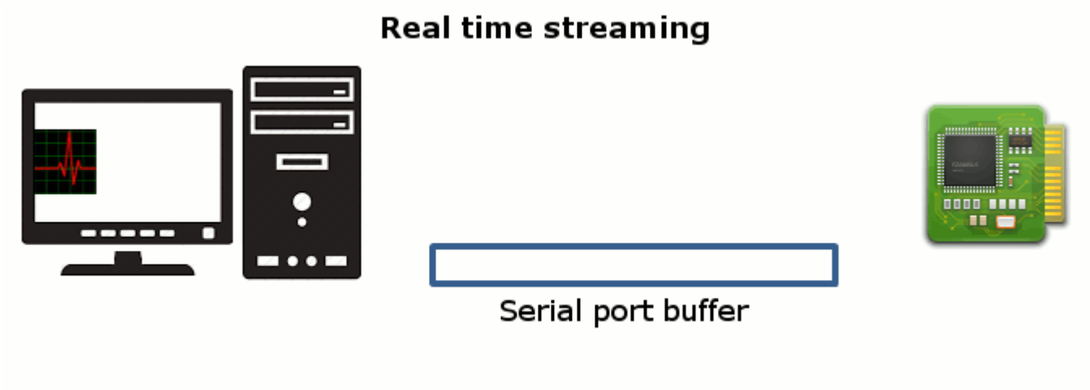
```
function  newdata(obj,event)
    [d,c] = fread(obj);  % 从串口获取数据% 注意：对于ASCII数据，使用fsca
    nf(obj)返回字符而非二进制值fprintf(1,'接收到 %d 字节',c);

    disp(d)
end
```

例如，要在接收到64字节数据时执行newdata函数，可以这样配置串口：

```
s = serial(port_name);
s.BytesAvailableFcnMode = 'byte';
s.BytesAvailableFcnCount = 64;
s.BytesAvailableFcn = @newdata;
```

对于文本或ASCII数据，数据通常通过“终止符字符”分成多行，就像页面上的文本一样。要在接收到回车字符时执行newdata函数，请按如下方式配置串口：

```
s = serial(port_name);
```

Section 28.3: Automatically processing data received from a serial port

Some devices connected through a serial port send data to your program at a constant rate (streaming data) or send data at unpredictable intervals. You can configure the serial port to execute a function automatically to handle data whenever it arrives. This is called the "callback function" for the serial port object.

There are two properties of the serial port that must be set to use this feature: the name of the function you want for the callback (`BytesAvailableFcn`), and the condition which should trigger executing the callback function (`BytesAvailableFcnMode`).

There are two ways to trigger a callback function:

1. When a certain number of bytes have been received at the serial port (typically used for binary data)
2. When a certain character is received at the serial port (typically used for text or ASCII data)

Callback functions have two required input arguments, called `obj` and `event`. `obj` is the serial port. For example, if you want to print the data received from the serial port, define a function for printing the data called `newdata`:

```
function newdata(obj,event)
    [d,c] = fread(obj);  % get the data from the serial port
    % Note: for ASCII data, use fscanf(obj) to return characters instead of binary values
    fprintf(1,'Received %d bytes\n',c);
    disp(d)
end
```

For example, to execute the `newdata` function whenever 64 bytes of data are received, configure the serial port like this:

```
s = serial(port_name);
s.BytesAvailableFcnMode = 'byte';
s.BytesAvailableFcnCount = 64;
s.BytesAvailableFcn = @newdata;
```

With text or ASCII data, the data is typically divided into lines with a "terminator character", just like text on a page. To execute the `newdata` function whenever the carriage return character is received, configure the serial port like this:

```
s = serial(port_name);
```

```
s.BytesAvailableFcnMode = 'terminator';
s.Terminator = 'CR';  % 回车，ASCII码13
s.BytesAvailableFcn = @newdata;
```

第28.4节：从串口读取数据

假设你像本例中那样创建了串口对象 `s`，则

```
% 读取一个字节
data = fread(s, 1);

% 读取所有字节，版本1
data = fread(s);

% 读取所有字节，版本2
data = fread(s, s.BytesAvailable);

% 关闭串口
fclose(s);
```

第28.5节：即使丢失、删除或被覆盖也能关闭串口

假设你像本例中那样创建了串口对象 `s`，那么关闭它的方法是

```
fclose(s)
```

但是，有时你可能会意外丢失端口（例如清除、覆盖、更改作用域等），这时 `fclose(s)` 将不再起作用。解决方法很简单

```
fclose(instrfindall)
```

更多信息见 instrfindall()。

第28.6节：向串口写入数据

假设你像本例中那样创建了串口对象 `s`，则

```
% 写入一个字节
fwrite(s, 255);

% 写入一个16位有符号整数
fwrite(s, 32767, 'int16');

% 写入一个无符号8位整数数组
fwrite(s,[48 49 50],'uchar');

% 关闭串口
fclose(s);
```



```
s.BytesAvailableFcnMode = 'terminator';
s.Terminator = 'CR';  % the carriage return, ASCII code 13
s.BytesAvailableFcn = @newdata;
```

Section 28.4: Reading from the serial port

Assuming you created the serial port object `s` as in this example, then

```
% Read one byte
data = fread(s, 1);

% Read all the bytes, version 1
data = fread(s);

% Read all the bytes, version 2
data = fread(s, s.BytesAvailable);

% Close the serial port
fclose(s);
```

Section 28.5: Closing a serial port even if lost, deleted or overwritten

Assuming you created the serial port object `s` as in this example, then to close it

```
fclose(s)
```

However, sometimes you can accidentally lose the port (e.g. clear, overwrite, change scope, etc...), and `fclose(s)` will no longer work. The solution is easy

```
fclose(instrfindall)
```

More info at `instrfindall()`.

Section 28.6: Writing to the serial port

Assuming you created the serial port object `s` as in this example, then

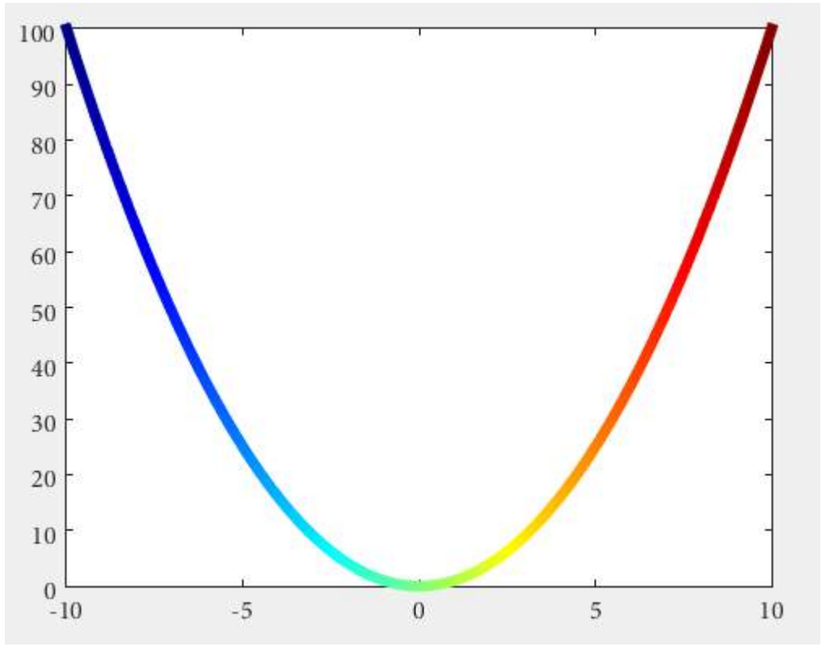
```
% Write one byte
fwrite(s, 255);

% Write one 16-bit signed integer
fwrite(s, 32767, 'int16');

% Write an array of unsigned 8-bit integers
fwrite(s,[48 49 50],'uchar');

% Close the serial port
fclose(s);
```

第29章：未记录的功能

第29.1节：带有第三维颜色数据的彩色编码二维折线图

在R2014b之前的MATLAB版本中，使用旧的HG1图形引擎时，创建彩色编码二维折线图并不直观。随着新HG2图形引擎的发布，出现了由Yair Altman引入的一个新的未记录功能：

```
n = 100;
x = linspace(-10,10,n); y = x.^2;
p = plot(x,y,'r', 'LineWidth',5);

% 修改后的jet色图
cd = [uint8(jet(n)*255) uint8(ones(n,1))].';

drawnow
set(p.Edge, 'ColorBinding','interpolated', 'ColorData',cd)
```



第29.2节：折线图和散点图中的半透明标记

自MATLAB R2014b起，可以轻松实现折线图和散点图的半透明标记，这得益于Yair Altman引入的未公开功能。

基本思路是获取标记的隐藏句柄，并对
EdgeColorData的最后一个值应用一个小于1的数值，以实现所需的透明度。

这里以scatter为例：

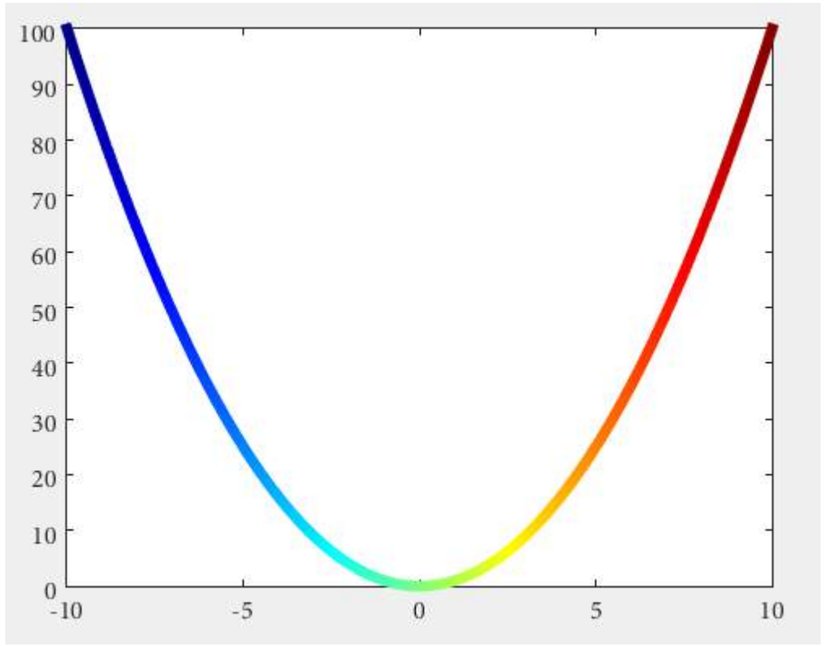
```
%// 示例数据
x = linspace(0,3*pi,200);
```


Chapter 29: Undocumented Features

Section 29.1: Color-coded 2D line plots with color data in third dimension

In MATLAB versions prior to **R2014b**, using the old HG1 graphics engine, it was not obvious how to create color coded 2D line plots. With the release of the new HG2 graphics engine arose a new **undocumented feature introduced by Yair Altman**:

```
n = 100;
x = linspace(-10,10,n); y = x.^2;
p = plot(x,y,'r', 'LineWidth',5);

% modified jet-colormap
cd = [uint8(jet(n)*255) uint8(ones(n,1))].';

drawnow
set(p.Edge, 'ColorBinding','interpolated', 'ColorData',cd)
```



Section 29.2: Semi-transparent markers in line and scatter plots

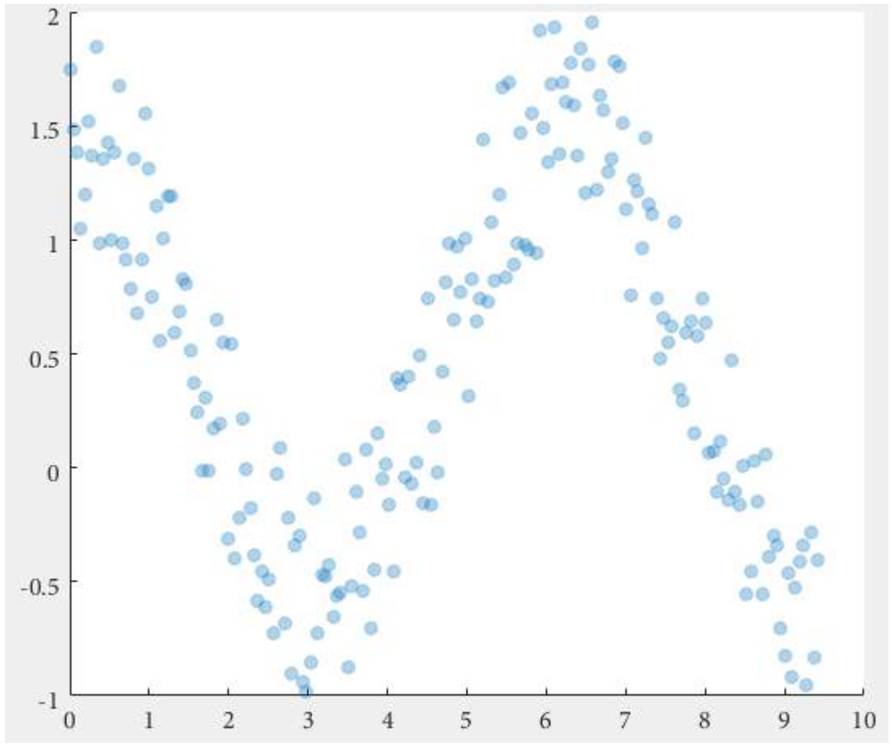
Since **MATLAB R2014b** it is easily possible to achieve semi-transparent markers for line and scatter plots using **undocumented features introduced by Yair Altman**.

The basic idea is to get the hidden handle of the markers and apply a value **< 1** for the last value in the `EdgeColorData` to achieve the desired transparency.

Here we go for `scatter`:

```
%// example data
x = linspace(0,3*pi,200);
```

```
y = cos(x) + rand(1,200);

%// 绘制散点图，获取句柄
h = scatter(x,y);
drawnow; %// 重要

%// 获取标记句柄
hMarkers = h.MarkerHandle;

%// 获取当前边缘和面颜色
edgeColor = hMarkers.EdgeColorData
faceColor = hMarkers.FaceColorData

%// 将面颜色设置为与边缘颜色相同
faceColor = edgeColor;

%// 透明度
opa = 0.3;

%// 设置标记边缘和面颜色
hMarkers.EdgeColorData = uint8( [edgeColor(1:3); 255*opa] );
hMarkers.FaceColorData = uint8( [faceColor(1:3); 255*opa] );
```

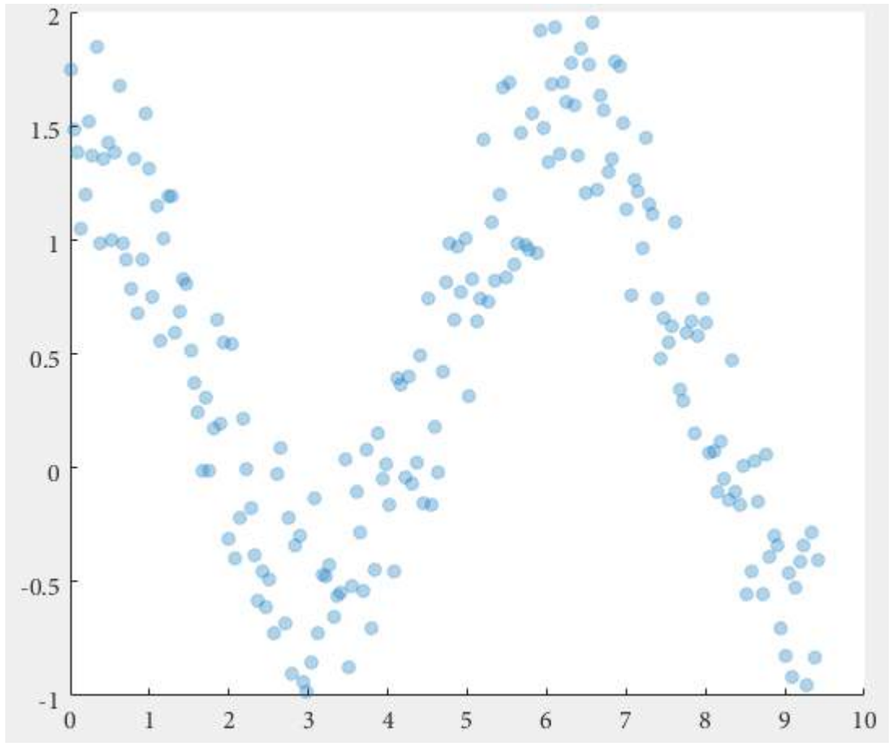


以及用于线条的 plot

```
%// 示例数据
x = linspace(0,3*pi,200);
y1 = cos(x);
y2 = sin(x);

%// 绘制散点图，获取句柄
h1 = plot(x,y1,'o-','MarkerSize',15); hold on
h2 = plot(x,y2,'o-','MarkerSize',15);
```



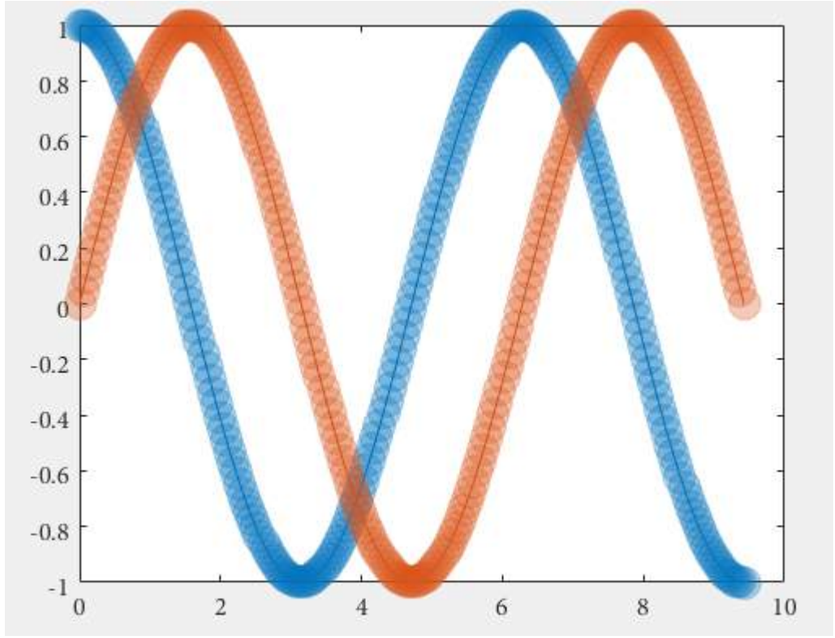
```
y = cos(x) + rand(1,200);

%// plot scatter, get handle
h = scatter(x,y);
drawnow; %// important

%// get marker handle
hMarkers = h.MarkerHandle;

%// get current edge and face color
edgeColor = hMarkers.EdgeColorData
faceColor = hMarkers.FaceColorData

%// set face color to the same as edge color
faceColor = edgeColor;

%// opacity
opa = 0.3;

%// set marker edge and face color
hMarkers.EdgeColorData = uint8( [edgeColor(1:3); 255*opa] );
hMarkers.FaceColorData = uint8( [faceColor(1:3); 255*opa] );
```



and for a line `plot`

```
%// example data
x = linspace(0,3*pi,200);
y1 = cos(x);
y2 = sin(x);

%// plot scatter, get handle
h1 = plot(x,y1,'o-','MarkerSize',15); hold on
h2 = plot(x,y2,'o-','MarkerSize',15);
```

```
drawnow; %// 重要

%// 获取标记句柄
h1Markers = h1.MarkerHandle;
h2Markers = h2.MarkerHandle;

%// 获取当前边缘和面颜色
edgeColor1 = h1Markers.EdgeColorData;
edgeColor2 = h2Markers.EdgeColorData;

%// 将面颜色设置为与边缘颜色相同
faceColor1 = edgeColor1;
faceColor2 = edgeColor2;

%// 透明度
opa = 0.3;

%// 设置标记边缘和面颜色
h1Markers.EdgeColorData = uint8( [edgeColor1(1:3); 255*opa] );
h1Markers.FaceColorData = uint8( [faceColor1(1:3); 255*opa] );
h2Markers.EdgeColorData = uint8( [edgeColor2(1:3); 255*opa] );
h2Markers.FaceColorData = uint8( [faceColor2(1:3); 255*opa] );
```



用于操作的标记句柄是随图形一起创建的。 drawnow 命令确保在调用后续命令之前图形已创建，并避免因延迟而产生错误。

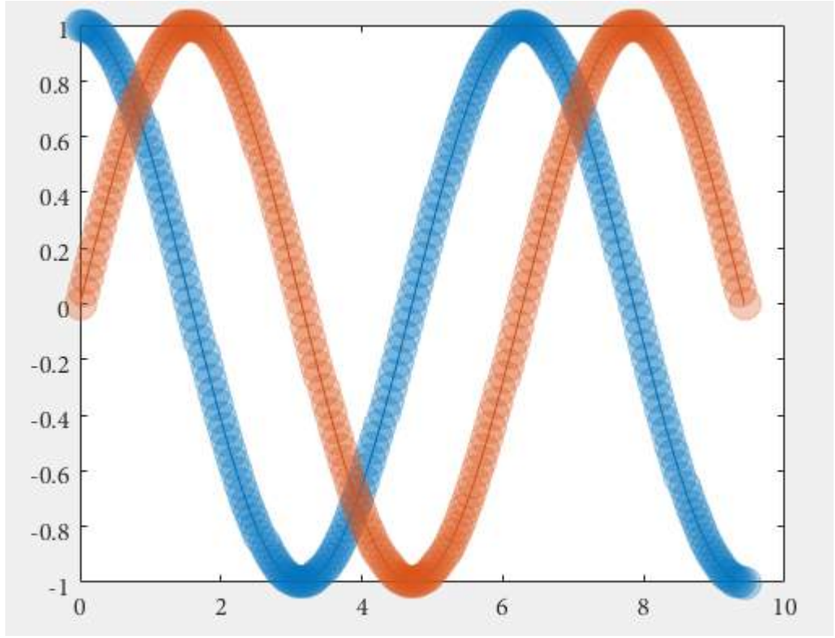
第29.3节：兼容C++的辅助函数

使用 MATLAB Coder 有时会禁止使用某些非常常见的函数，如果它们不兼容C++。相对常见的是存在一些 未公开的辅助函数，可以用作替代。

以下是支持函数的完整列表。。

以及以下是对不支持函数的替代集合：


```
drawnow; %// important

%// get marker handle
h1Markers = h1.MarkerHandle;
h2Markers = h2.MarkerHandle;

%// get current edge and face color
edgeColor1 = h1Markers.EdgeColorData;
edgeColor2 = h2Markers.EdgeColorData;

%// set face color to the same as edge color
faceColor1 = edgeColor1;
faceColor2 = edgeColor2;

%// opacity
opa = 0.3;

%// set marker edge and face color
h1Markers.EdgeColorData = uint8( [edgeColor1(1:3); 255*opa] );
h1Markers.FaceColorData = uint8( [faceColor1(1:3); 255*opa] );
h2Markers.EdgeColorData = uint8( [edgeColor2(1:3); 255*opa] );
h2Markers.FaceColorData = uint8( [faceColor2(1:3); 255*opa] );
```



The marker handles, which are used for the manipulation, are created with the figure. The `drawnow` command is ensuring the creation of the figure before subsequent commands are called and avoids errors in case of delays.

Section 29.3: C++ compatible helper functions

The use of **MATLAB Coder** sometimes denies the use of some very common functions, if they are not compatible to C++. Relatively often there exist **undocumented helper functions**, which can be used as replacements.

Here is a comprehensive list of supported functions..

And following a collection of alternatives, for non-supported functions:

函数 sprintf 和 sprintfc 非常相似，前者返回一个 字符数组，后者返回一个 单元字符串：

```
str = sprintf('%i',x)    % 当 x = 5 时返回 '5'
 str = sprintfc('%i',x)  % 当 x = 5 时返回 {'5'}
```

然而，sprintfc 兼容 MATLAB Coder 支持的 C++，而 sprintf 不兼容。

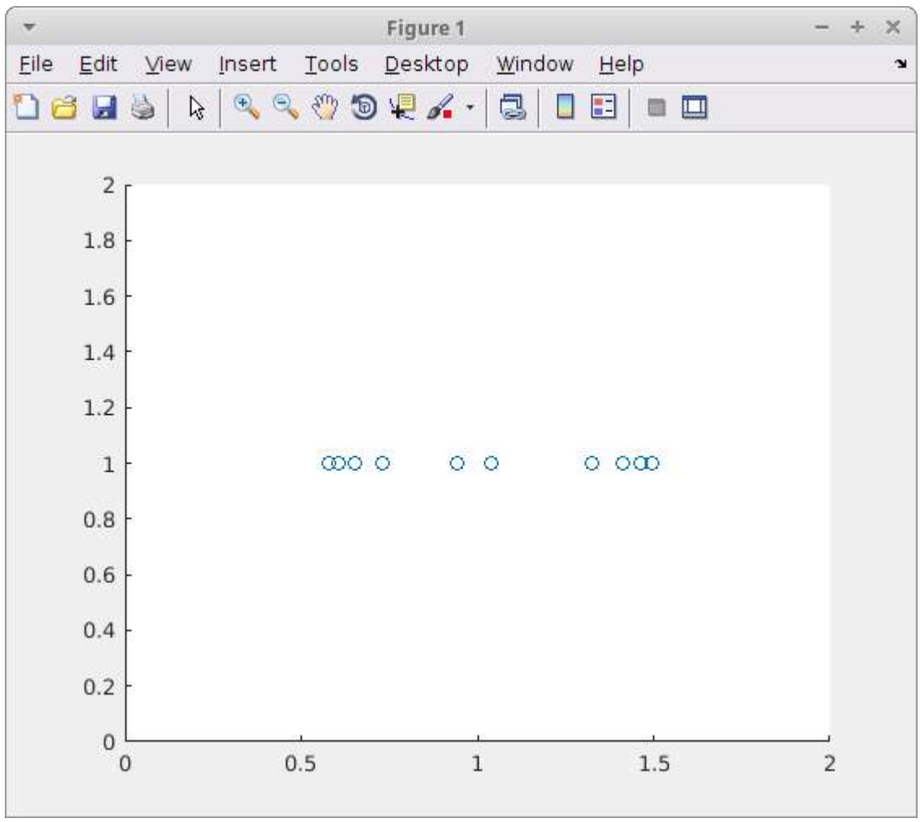
第29.4节：散点图抖动

scatter函数有两个未公开的属性'jitter'和'jitterAmount'，它们允许仅在x轴上对数据进行抖动。这可以追溯到MATLAB 7.1（2005年），甚至可能更早。

要启用此功能，将'jitter'属性设置为'on'，并将'jitterAmount'属性设置为所需的绝对值（默认值为0.2）。

当我们想要可视化重叠数据时，这非常有用，例如：

```
scatter(ones(1,10), ones(1,10), 'jitter', 'on', 'jitterAmount', 0.5);
```



更多内容请参见Undocumented Matlab

第29.5节：等高线图 - 自定义文本标签

在显示等高线标签时，MATLAB不允许你控制数字的格式，例如切换为科学计数法。

单个文本对象是普通的文本对象，但获取它们的方法未公开。你可以通过

The functions sprintf and sprintfc are quite similar, the former returns a *character array*, the latter a *cell string*:

```
 str = sprintf('%i',x)   % returns '5' for x = 5
 str = sprintfc('%i',x)  % returns {'5'} for x = 5
```

However, sprintfc is compatible with C++ supported by MATLAB Coder, and sprintf is not.

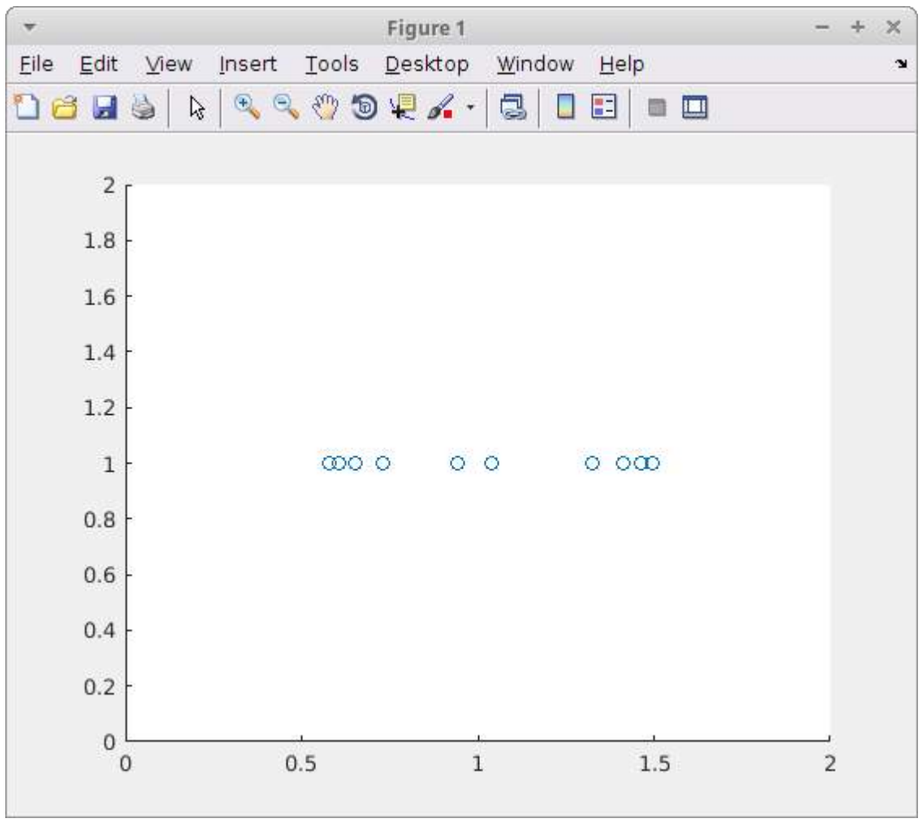
Section 29.4: Scatter plot jitter

The scatter function has two undocumented properties 'jitter' and 'jitterAmount' that allow to jitter the data on the x-axis only. This dates back to MATLAB 7.1 (2005), and possibly earlier.

To enable this feature set the 'jitter' property to 'on' and set the 'jitterAmount' property to the desired absolute value (the default is 0.2).

This is very useful when we want to visualize overlapping data, for example:

```
scatter(ones(1,10), ones(1,10), 'jitter', 'on', 'jitterAmount', 0.5);
```



Read more on Undocumented Matlab

Section 29.5: Contour Plots - Customise the Text Labels

When displaying labels on contours MATLAB doesn't allow you to control the format of the numbers, for example to change to scientific notation.

The individual text objects are normal text objects but how you get them is undocumented. You access them from

等高线句柄的TextPrims属性访问它们。

```
  图
  [X,Y]=meshgrid(0:100,0:100);
  Z=(X+Y.^2)*1e10;
  [C,h]=contour(X,Y,Z);
h.ShowText='on';
  drawnow();
txt = get(h,'TextPrims');
  v = str2double(get(txt,'String'));
  for ii=1:length(v)
    set(txt(ii),'String',sprintf('%0.3e',v(ii)))
  end
```

注意：必须添加一个drawnow命令以强制MATLAB绘制等高线，txt对象的数量和位置只有在等高线实际绘制时才确定，因此文本对象也是在那时创建的。

txt对象是在绘制等高线时创建的，这意味着每次重新绘制图形（例如调整图形大小）时都会重新计算它们。为此，您需要监听undocumented event MarkedClean:

```
function customiseContour
  figure
  [X,Y]=meshgrid(0:100,0:100);
  Z=(X+Y.^2)*1e10;
  [C,h]=contour(X,Y,Z);
h.ShowText='on';
  % 添加监听器并调用您的新格式函数
  addlistener(h,'MarkedClean',@(a,b)ReFormatText(a))
end
function ReFormatText(h)
  % 从轮廓中获取所有文本项
  t = get(h,'TextPrims');
  for ii=1:length(t)
    % 获取当前值（MATLAB在重新绘制图形时会将其恢复）

v = str2double(get(t(ii),'String'));
    % 使用你想要的格式更新 - 例如科学计数法
    set(t(ii),'String',sprintf('%0.3e',v));
  end
end
```

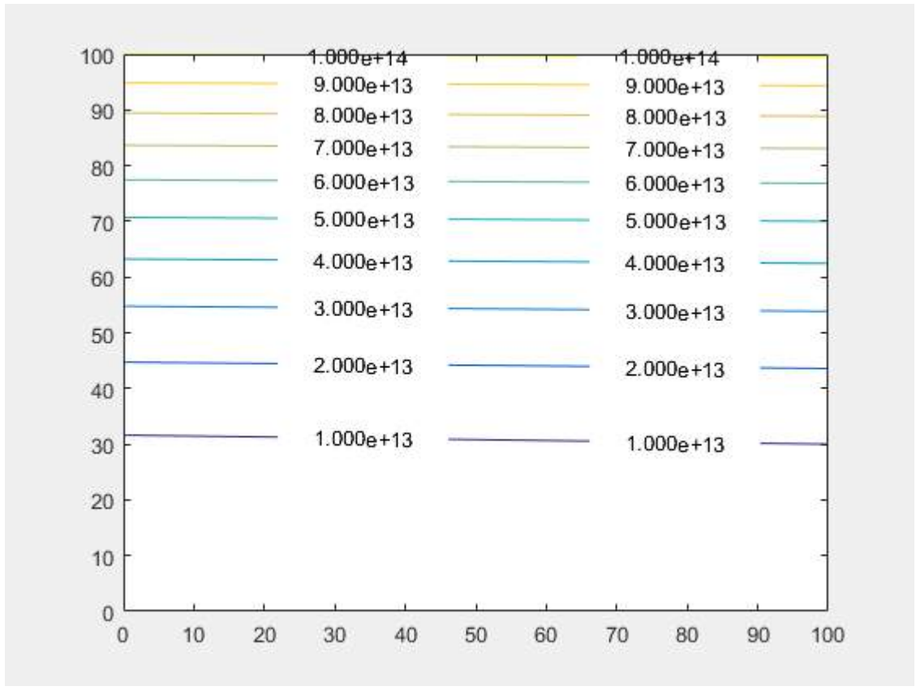

the `TextPrims` property of the contour handle.

```
figure
[X,Y]=meshgrid(0:100,0:100);
Z=(X+Y.^2)*1e10;
[C,h]=contour(X,Y,Z);
h.ShowText='on';
drawnow();
txt = get(h,'TextPrims');
v = str2double(get(txt,'String'));
for ii=1:length(v)
  set(txt(ii),'String',sprintf('%0.3e',v(ii)))
end
```

Note: that you must add a `drawnow` command to force MATLAB to draw the contours, the number and location of the txt objects are only determined when the contours are actually drawn so the text objects are only created then.

The fact the txt objects are created when the contours are drawn means that they are recalculated every time the plot is redrawn (for example figure resize). To manage this you need to listen to the `undocumented event MarkedClean`:

```
function customiseContour
  figure
  [X,Y]=meshgrid(0:100,0:100);
  Z=(X+Y.^2)*1e10;
  [C,h]=contour(X,Y,Z);
  h.ShowText='on';
  % add a listener and call your new format function
  addlistener(h,'MarkedClean',@(a,b)ReFormatText(a))
end
function ReFormatText(h)
  % get all the text items from the contour
  t = get(h,'TextPrims');
  for ii=1:length(t)
    % get the current value (MATLAB changes this back when it
    %   redraws the plot)
    v = str2double(get(t(ii),'String'));
    % Update with the format you want - scientific for example
    set(t(ii),'String',sprintf('%0.3e',v));
  end
end
```



示例在Windows上的MATLAB r2015b中测试

第29.6节：向现有图例追加/添加条目

现有图例可能难以管理。例如，如果你的图中有两条线，但只有其中一条有图例条目且应保持如此，那么添加第三条带图例条目的线可能会很困难。示例：

```
图
hold on
fplot(@sin)
fplot(@cos)
legend sin  % 仅为 sin 添加图例条目
hTan = fplot(@tan);  % 确保获取要添加到图例中的图形对象的句柄 hTan
```

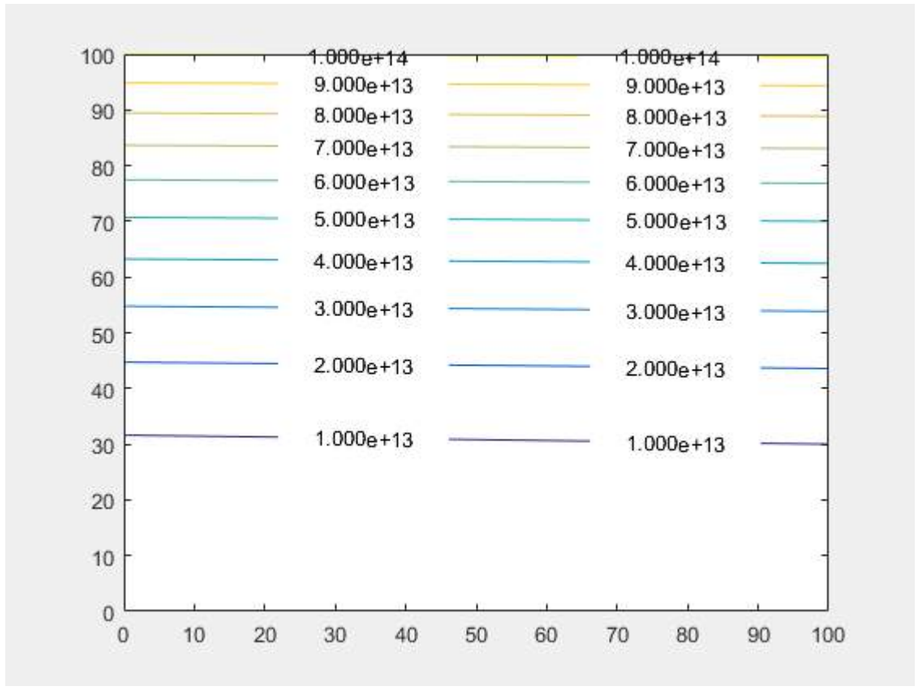
现在，要为 tan 添加图例条目，但不为 cos 添加，以下任意一行都无法实现；它们都会以某种方式失败：

```
legend tangent  % 替换当前图例 -> 失败
legend -DynamicLegend  % 未记录的功能，添加了不应添加的 'cos' -> 失败
legend sine tangent  % 将 cos 的 DisplayName 设置为 'tangent' -> 失败
legend sine '' tangent  % 设置 cos 的 DisplayName，虽然为空 -> 失败
legend(f)
```

幸运的是，有一个未记录的图例属性叫做 PlotChildren，用于跟踪父图形 figure1 的子对象。
因此，正确的方法是通过图例的 PlotChildren 属性显式设置图例的子对象，如下所示：

```
hTan.DisplayName = 'tangent';  % 设置图形对象的显示名称
l = legend;
l.PlotChildren(end + 1) = hTan;  % 将图形句柄追加到图例的 PlotChildren 中
```

如果向 PlotChildren 属性添加或移除对象，图例会自动更新。



Example tested using MATLAB r2015b on Windows

Section 29.6: Appending / adding entries to an existing legend

Existing legends can be difficult to manage. For example, if your plot has two lines, but only one of them has a legend entry and that should stay this way, then adding a third line with a legend entry can be difficult. Example:

```
figure
hold on
fplot(@sin)
fplot(@cos)
legend sin  % Add only a legend entry for sin
hTan = fplot(@tan);  % Make sure to get the handle, hTan, to the graphics object you want to add to
the legend
```

Now, to add a legend entry for `tan`, but not for `cos`, any of the following lines won't do the trick; they all fail in some way:

```
legend tangent  % Replaces current legend -> fail
legend -DynamicLegend  % Undocumented feature, adds 'cos', which shouldn't be added -> fail
legend sine tangent  % Sets cos DisplayName to 'tangent' -> fail
legend sine '' tangent  % Sets cos DisplayName, albeit empty -> fail
legend(f)
```

Luckily, an undocumented legend property called `PlotChildren` keeps track of the children of the parent figure1. So, the way to go is to explicitly set the legend's children through its `PlotChildren` property as follows:

```
hTan.DisplayName = 'tangent';  % Set the graphics object's display name
l = legend;
l.PlotChildren(end + 1) = hTan;  % Append the graphics handle to legend's plot children
```

The legend updates automatically if an object is added or removed from its `PlotChildren` property.

1 确实：图形。您可以将任何图形的子对象通过DisplayName属性添加到图形中的任何图例中，例如来自不同的子图。这是因为图例本质上就是一个坐标轴对象。

在 MATLAB R2016b 上测试

1 Indeed: figure. You can add any figure's child with the `DisplayName` property to any legend in the figure, e.g. from a different subplot. This is because a legend in itself is basically an axes object.

Tested on MATLAB R2016b

第30章：MATLAB最佳实践

第30.1节：正确缩进代码

正确的缩进不仅使代码美观，还提高了代码的可读性。

例如，考虑以下代码：

```
%无需理解代码，只需看一眼
n = 2;
bf = false;
while n>1
for ii = 1:n
for jj = 1:n
if ii+jj>30
bf = true;
break
end
end
end
if bf
break
end
end
if bf
break
end
n = n + 1;
end
```

如您所见，您需要仔细查看以确定哪个循环和if语句在哪里结束。
使用智能缩进后，你将获得如下效果：

```
n = 2;
bf = false;
while n>1
    for ii = 1:n
        for jj = 1:n
            if ii+jj>30
bf = true;
                break
            end
        end
        if bf
            break
        end
    end
    if bf
        break
    end
n = n + 1;
end
```

这清楚地表明了循环/条件语句的开始和结束。

你可以通过以下方式进行智能缩进：
•选择你所有的代码（ Ctrl+A)
•然后按下 Ctrl+我 或点击  从编辑栏。

Chapter 30: MATLAB Best Practices

Section 30.1: Indent code properly

Proper indentation gives not only the aesthetic look but also increases the readability of the code.

For example, consider the following code:

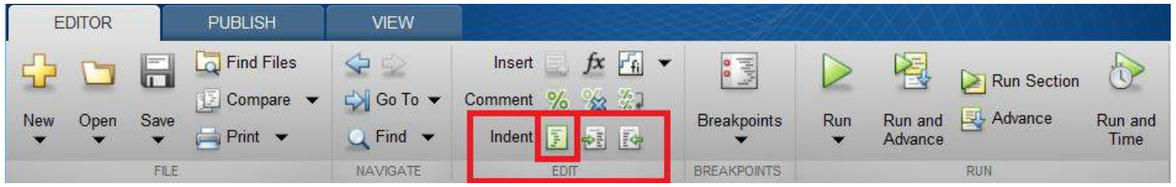
```
%no need to understand the code, just give it a look
n = 2;
bf = false;
while n>1
for ii = 1:n
for jj = 1:n
if ii+jj>30
bf = true;
break
end
end
end
if bf
break
end
end
if bf
break
end
n = n + 1;
end
```

As you can see, you need to give a careful look to see which loop and `if` statements are ending where.
With smart indentation, you'll get this look:

```
n = 2;
bf = false;
while n>1
    for ii = 1:n
        for jj = 1:n
            if ii+jj>30
                bf = true;
                break
            end
        end
        if bf
            break
        end
    end
    if bf
        break
    end
    n = n + 1;
end
```

This clearly indicates the starting and ending of loops/`if` statement.

You can do smart indentation by:
• selecting all your code (Ctrl+A)
• and then pressing Ctrl+I or clicking  from edit bar.



第30.2节：避免循环

大多数情况下，使用MATLAB时循环计算开销较大。如果使用向量化，代码速度将快几个数量级。它还常常使代码更模块化，更易修改和调试。主要缺点是需要花时间规划数据结构，而且维度错误更容易出现。

示例

不要写

```
for t=0:0.1:2*pi
    R(end+1)=cos(t);
end
```

而是

```
t=0:0.1:2*pi;
R=cos(t)
```

不要写

```
for i=1:n
     for j=1:m
c(i,j)=a(i)+2*b(j);
     end
end
```

但类似于

```
c=repmat(a.',1,m)+2*repmat(b,n,1)
```

更多细节请参见向量化

第30.3节：保持行短

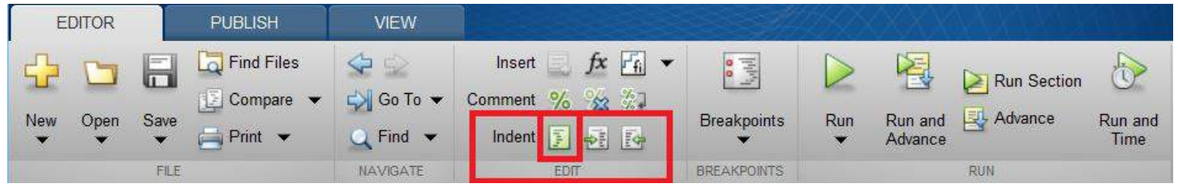
使用续行符（省略号）...来继续长语句。

示例：

```
MyFunc( parameter1,parameter2,parameter3,parameter4, parameter5,parameter6,parameter7, parameter8,
parameter9)
```

可以替换为：

```
MyFunc( 参数1, ...
          parameter2, ...
          parameter3, ...
```

Section 30.2: Avoid loops

Most of the time, loops are computationally expensive with MATLAB. Your code will be orders of magnitudes faster if you use vectorization. It also often makes your code more modular, easily modifiable, and easier to debug. The major downside is that you have to take time to plan the data structures, and dimension errors are easier to come by.

Examples

Don't write

```
for t=0:0.1:2*pi
    R(end+1)=cos(t);
end
```

but

```
t=0:0.1:2*pi;
R=cos(t)
```

Don't write

```
for i=1:n
    for j=1:m
        c(i,j)=a(i)+2*b(j);
    end
end
```

But something similar to

```
c=repmat(a.',1,m)+2*repmat(b,n,1)
```

For more details, see vectorization

Section 30.3: Keep lines short

Use the continuation character (ellipsis) `...` to continue long statement.

Example:

```
MyFunc( parameter1,parameter2,parameter3,parameter4, parameter5, parameter6,parameter7, parameter8,
parameter9)
```

can be replaced by:

```
MyFunc( parameter1, ...
        parameter2, ...
        parameter3, ...
```

```
        parameter4, ...
        parameter5, ...
        parameter6, ...
        parameter7, ...
        parameter8, ...
        parameter9)
```

第30.4节：使用assert

MATLAB允许一些非常微小的错误悄无声息地通过，这可能导致错误在运行的后期才被触发——这使得调试变得困难。如果你假设你的变量具有某些属性，验证它们。

```
function out1 = get_cell_value_at_index(scalar1,cell2)
assert(isscalar(scalar1),'第1个输入必须是标量')assert(iscell(cell2),'第2
个输入必须是单元数组')assert(numel(cell2) >= scalar1),'第2个输入的元

素数量必须大于第1个输入的值')

assert(~isempty(cell2{scalar1}),'第2个输入在该位置为空')

out1 = cell2{scalar1};
```

第30.5节：块注释操作符

添加描述代码的注释是一种良好习惯。这对他人以及以后返回查看代码的编写者都很有帮助。单行注释可以使用%符号或快捷键 Ctrl+R。
要取消注释之前注释的行，删除%符号或使用快捷键Crtl+T。

虽然可以通过在每行开头添加%符号来注释一段代码块，但MATLAB的新版本（2015a之后）允许使用块注释操作符%{ 代码 %}。该操作符提高了代码的可读性。它既可用于代码注释，也可用于函数帮助文档。代码块可以被折叠和展开以提高代码的可读性。

```
function y = myFunction(x)
%{
myFunction  Binary Singleton Expansion Function
y = myFunction(x) applies the element-by-element binary operation
specified by the function handle FUNC to arrays A and B, with implicit
expansion enabled.
%}

%%  Compute z(x, y) = x.*sin(y) on a grid:
% x = 1:10;
y = x.';

%{
z = zeros(numel(x),numel(y));
for ii=1:numel(x)
    for jj=1:numel(y)
        z(ii,jj) = x(ii)*sin(y(jj));
    end
end
%}

z = bsxfun(@(x, y) x.*sin(y), x, y);
y = y + z;

end
```


如图所示，%{和%}操作符必须单独占据一行。不要在这些行中包含其他文本。


```
        parameter4, ...
        parameter5, ...
        parameter6, ...
        parameter7, ...
        parameter8, ...
        parameter9)
```

Section 30.4: Use assert

MATLAB allows some very trivial mistakes to go by silently, which might cause an error to be raised much later in the run - making debugging hard. If you **assume** something about your variables, **validate** it.

```
function out1 = get_cell_value_at_index(scalar1,cell2)
assert(isscalar(scalar1),'1st input must be a scalar')
assert(iscell(cell2),'2nd input must be a cell array')

assert(numel(cell2) >= scalar1),'2nd input must have more elements than the value of the 1st
input')
assert(~isempty(cell2{scalar1}),'2nd input at location is empty')

out1 = cell2{scalar1};
```

Section 30.5: Block Comment Operator

It is a good practice to add comments that describe the code. It is helpful for others and even for the coder when returned later. A single line can be commented using the % symbol or using the shortkey Ctrl+R. To uncomment a previously commented line remove the % symbol or use shortkey Crtl+T.

While commenting a block of code can be done by adding a % symbol at the beginning of each line, newer versions of MATLAB (after 2015a) let you use the **Block Comment Operator** %{ code %}. This operator increases the readability of the code. It can be used for both code commenting and function help documentation. The Block can be **folded** and **unfolded** to increase the readability of the code.

```
function y = myFunction(x)
%{
myFunction  Binary Singleton Expansion Function
y = myFunction(x) applies the element-by-element binary operation
specified by the function handle FUNC to arrays A and B, with implicit
expansion enabled.
%}

%%  Compute z(x, y) = x.*sin(y) on a grid:
% x = 1:10;
y = x.';

%{
z = zeros(numel(x),numel(y));
for ii=1:numel(x)
    for jj=1:numel(y)
        z(ii,jj) = x(ii)*sin(y(jj));
    end
end
%}

z = bsxfun(@(x, y) x.*sin(y), x, y);
y = y + z;

end
```


As it can be seen the %{ and %} operators must appear alone on the lines. Do not include any other text on these lines.

```
function y = myFunction(x)
%{
myFunction  二元单例扩展函数y = myFunction(x)对数组A和
B应用由函数句柄FUNC指定的逐元素二元操作，启用隐式扩展。


%}

%% 计算网格上的 z(x, y) = x.*sin(y)：
% x = 1:10;
y = x.';

%{
z = zeros(numel(x),numel(y));
for ii=1:numel(x)
    for jj=1:numel(y)
z(ii,jj) = x(ii)*sin(y(jj));
    end
end
%}

z = bsxfun(@(x, y) x.*sin(y), x, y);
y = y + z;

end
```

第30.6节：为临时文件创建唯一名称

在编写脚本或函数时，可能需要一个或多个临时文件，例如用于存储一些数据。

为了避免覆盖现有文件或遮蔽 MATLAB 函数，可以使用tempname函数在系统临时文件夹中生成临时文件的唯一名称。

```
my_temp_file=tempname
```

文件名是在没有扩展名的情况下生成的；可以通过将所需的扩展名连接到由 tempname生成的名称来添加扩展名

```
my_temp_file_with_ext=[tempname '.txt']
```

系统临时文件夹的位置可以通过调用 tempdir函数来获取。

如果在函数/脚本执行过程中临时文件不再需要，可以使用 delete函数将其删除

由于 delete不会请求确认，设置 on选项将要删除的文件移动到 recycle文件夹中可能会很有用。

这可以通过使用 recycle函数来实现，方法如下：

```
recycle('on')
```

在下面的示例中，提出了 tempname、 delete和 recycle函数的可能用法。

```
%
% 创建一些示例数据
```



```
function y = myFunction(x)
%{
myFunction  Binary Singleton Expansion Function
y = myFunction(x) applies the element-by-element binary operation
specified by the function handle FUNC to arrays A and B, with implicit
expansion enabled.
%}

%%  Compute z(x, y) = x.*sin(y) on a grid:
% x = 1:10;
y = x.';

%{
z = zeros(numel(x),numel(y));
for ii=1:numel(x)
    for jj=1:numel(y)
        z(ii,jj) = x(ii)*sin(y(jj));
    end
end
%}

z = bsxfun(@(x, y) x.*sin(y), x, y);
y = y + z;

end
```

Section 30.6: Create Unique Name for Temporary File

While coding a script or a function, it can be the case that one or more than one temporary file be needed in order to, for example, store some data.

In order to avoid overwriting an existing file or to shadow a MATLAB function the tempname function can be used to generate a **unique name** for a temporary file in the system temporary folder.

```
my_temp_file=tempname
```

The filename is generated without the extension; it can be added by concatenating the desired extension to the name generated by `tempname`

```
my_temp_file_with_ext=[tempname '.txt']
```

The location of the system temporary folder can be retrieved by calling the tempdir function.

If, during the execution of the function / script, the temporary file is no longer needed, it can be deleted by using the function delete

Since `delete` does not ask for confirmation, it might be useful to set `on` the option to move the file to be deleted in the `recycle` folder.

This can be done by using the function recycle this way:

```
recycle('on')
```

In the following example, a possible usage of the functions `tempname`, `delete` and `recycle` is proposed.

```
%
% Create some example data
```

```
%
theta=0:.1:2*pi;
x=cos(theta);
y=sin(theta);
%
% 生成临时文件名
%
my_temp_file=[tempname '.mat'];
%
% 分割文件名（路径、名称、扩展名）并在消息框中显示
[tmp_file_path,tmp_file_name, tmp_file_ext]=fileparts(my_temp_file)
uiwait(msgbox(sprintf('路径= %s名称= %s扩展名= %s', ...
                    tmp_file_path,tmp_file_name,tmp_file_ext),'临时文件'))
%
% 将变量保存到临时文件
%
save(my_temp_file,'x','y','theta')
%
% 从临时文件加载变量
%
load(my_temp_file)
%
% 开启回收选项
%
recycle('on')
%
% 删除临时文件
%
delete(my_temp_file)
```

注意事项

临时文件名是通过使用java.util.UUID.randomUUID方法（randomUUID）生成的。

如果MATLAB在无JVM环境下运行，临时文件名是通过使用
matlab.internal.timing.timing基于CPU计数器和时间生成的。在这种情况下，临时文件名不保证唯一性。


```
%
theta=0:.1:2*pi;
x=cos(theta);
y=sin(theta);
%
% Generate the temporary filename
%
my_temp_file=[tempname '.mat'];
%
% Split the filename (path, name, extension) and display them in a message box
[tmp_file_path,tmp_file_name, tmp_file_ext]=fileparts(my_temp_file)
uiwait(msgbox(sprintf('Path= %s\nName= %s\nExt= %s', ...
                tmp_file_path,tmp_file_name,tmp_file_ext),'TEMPORARY FILE'))
%
% Save the variables in a temporary file
%
save(my_temp_file,'x','y','theta')
%
% Load the variables from the temporary file
%
load(my_temp_file)
%
% Set the recycle option on
%
recycle('on')
%
% Delete the temporary file
%
delete(my_temp_file)
```

Caveat

The temporary filename is generated by using the `java.util.UUID.randomUUID` method (randomUUID).

If MATLAB is run without JVM, the temporary filename is generated by using `matlab.internal.timing.timing` based on the CPU counter and time. In this case the temporary filename is not guaranteed to be unique.

第31章：MATLAB用户界面

第31.1节：在用户界面中传递数据

大多数高级用户界面要求用户能够在构成用户界面的各个函数之间传递信息。MATLAB有多种不同的方法来实现这一点。

guidata

MATLAB 自带的GUI 开发环境 (GUIDE)倾向于使用一个名为handles的struct在回调函数之间传递数据。该struct 包含了各种 UI 组件的所有图形句柄以及用户指定的数据。如果你没有使用 GUIDE 创建的自动传递handles的回调函数，可以使用guidata来获取当前值

```
% hObject 是你 GUI 中任何 UI 组件的图形句柄
handles = guidata(hObject);
```

如果你想修改存储在该数据结构中的值，可以进行修改，但必须将其重新存储回 hObject中，才能让其他回调函数看到更改。你可以通过向guidata指定第二个输入参数来存储它。

```
% 更新值
handles.myValue = 2;

% 保存更改
guidata(hObject, handles)
```

hObject的值无关紧要，只要它是同一figure内的 UI 组件，因为数据最终存储在包含 hObject的 figure 中。

适用场景：

- 存储handles结构体，你可以在其中存储所有 GUI 组件的句柄。
- 存储需要被大多数回调访问的“较小”其他变量。

不推荐用于：

- 存储不需要所有回调函数和子函数访问的大型变量（使用 setappdata/getappdata 来实现）。

setappdata/getappdata

类似于guidata方法，你可以使用setappdata和getappdata在图形句柄内存储和检索值。使用这些方法的优点是你可以只检索你想要的值，而不是整个包含所有存储数据的struct。它类似于一个键/值存储。

在图形对象中存储数据

```
% 创建一些你想要存储的数据
myvalue = 2

% 使用键 'mykey' 存储它
```


Chapter 31: MATLAB User Interfaces

Section 31.1: Passing Data Around User Interface

Most advanced user interfaces require the user to be able to pass information between the various functions which make up a user interface. MATLAB has a number of different methods to do so.

guidata

MATLAB's own GUI Development Environment (GUIDE) prefers to use a `struct` named `handles` to pass data between callbacks. This `struct` contains all of the graphics handles to the various UI components as well as user-specified data. If you aren't using a GUIDE-created callback which automatically passes `handles`, you can retrieve the current value using `guidata`

```
% hObject is a graphics handle to any UI component in your GUI
handles = guidata(hObject);
```

If you want to modify a value stored in this data structure, you can modify but then you must store it back within the `hObject` for the changes to be visible by other callbacks. You can store it by specifying a second input argument to `guidata`.

```
% Update the value
handles.myValue = 2;

% Save changes
guidata(hObject, handles)
```

The value of `hObject` doesn't matter as long as it is a UI component *within the same* `figure` because ultimately the data is stored within the figure containing `hObject`.

Best for:

- Storing the `handles` structure, in which you can store all the handles of your GUI components.
- Storing "small" other variables which need to be accessed by most callbacks.

Not recommended for:

- Storing large variables which do not have to be accessed by all callbacks and sub-functions (use `setappdata/getappdata` for these).

setappdata/getappdata

Similar to the `guidata` approach, you can use `setappdata` and `getappdata` to store and retrieve values from within a graphics handle. The advantage of using these methods is that you can retrieve *only the value you want* rather than an entire `struct` containing *all* stored data. It is similar to a key/value store.

To store data within a graphics object

```
% Create some data you would like to store
myvalue = 2

% Store it using the key 'mykey'
```

```
setappdata(hObject, 'mykey', myvalue)
```

并且在不同的回调函数中检索相同的值

```
value = getappdata(hObject, 'mykey');
```

注意： 如果在调用 getappdata 之前没有存储任何值，它将返回一个空数组（[]）。

类似于 guidata，数据存储在包含 hObject 的图形对象中。

适用场景：

- 存储不需要被所有回调函数和子函数访问的大型变量。

UserData

每个图形句柄都有一个特殊属性，UserData，可以包含你想要的任何数据。它可以包含一个单元数组、一个 struct，甚至是一个标量。你可以利用这个属性，将任何你希望与给定图形句柄关联的数据存储在此字段中。你可以使用图形对象的标准 get/set方法，或者如果你使用的是 R2014b 或更高版本，可以使用点符号来保存和检索该值。

```
% 创建一些要存储的数据
mydata = {1, 2, 3};

% 将其存储在 UserData 属性中
set(hObject, 'UserData', mydata)

% 如果你使用的是 R2014b 或更高版本：
% hObject.UserData = mydata;
```

然后在另一个回调函数中，你可以检索这些数据：

```
their_data = get(hObject, 'UserData');

% 或者如果你使用的是 R2014b 或更新版本：
% their_data = hObject.UserData;
```

适用场景：

- 存储作用域有限的变量（这些变量很可能仅被存储它们的对象或与其有直接关系的对象使用）。

嵌套函数

在 MATLAB 中，嵌套函数可以读取和修改父函数中定义的任何变量。这样，如果你指定回调函数为嵌套函数，它就可以检索并修改主函数中存储的任何数据。

```
function mygui()
hButton = uicontrol('String', '点击我', 'Callback', @callback);

    % 创建一个计数器以跟踪按钮被点击的次数
    nClicks = 0;

    % 回调函数是嵌套的，因此可以读取和修改 nClicks
    function callback(source, event)
```



```
setappdata(hObject, 'mykey', myvalue)
```

And to retrieve that same value from within a different callback

```
value = getappdata(hObject, 'mykey');
```

Note: If no value was stored prior to calling `getappdata`, it will return an empty array ([ ]).

Similar to `guidata`, the data is stored in the figure that contains `hObject`.

Best for:

- Storing large variables which do not have to be accessed by all callbacks and sub-functions.

UserData

Every graphics handle has a special property, `UserData` which can contain any data you wish. It could contain a cell array, a `struct`, or even a scalar. You can take advantage of this property and store any data you wish to be associated with a given graphics handle in this field. You can save and retrieve the value using the standard `get`/`set` methods for graphics objects or dot notation if you're using R2014b or newer.

```
% Create some data to store
mydata = {1, 2, 3};

% Store it within the UserData property
set(hObject, 'UserData', mydata)

% Of if you're using R2014b or newer:
% hObject.UserData = mydata;
```

Then from within another callback, you can retrieve this data:

```
their_data = get(hObject, 'UserData');

% Or if you're using R2014b or newer:
% their_data = hObject.UserData;
```

Best for:

- Storing variables with a limited scope (variables which are likely to be used only by the object in which they are stored, or objects having a direct relationship to it).

Nested Functions

In MATLAB, a nested function can read and modify any variable defined in the parent function. In this way, if you specify a callback to be a nested function, it can retrieve and modify any data stored in the main function.

```
function mygui()
    hButton = uicontrol('String', 'Click Me', 'Callback', @callback);

    % Create a counter to keep track of the number of times the button is clicked
    nClicks = 0;

    % Callback function is nested and can therefore read and modify nClicks
    function callback(source, event)
```

```
        % 点击次数加一
        nClicks = nClicks + 1;

        % 打印到目前为止的点击次数fprintf('点击次数
        : %d', nClicks);end

end
```

适用场景：

- 小型、简单的图形用户界面（GUI）。(用于快速原型设计，无需实现`guidata`和/或`set`/`getappdata`方法)。

不推荐用于：

- 中型、大型或复杂的图形用户界面（GUI）。
- 使用GUIDE创建的图形用户界面（GUI）。

显式输入参数

如果您需要将数据发送到回调函数，并且不需要在回调中修改数据，您可以始终考虑使用精心设计的回调定义将数据传递给回调函数。

您可以使用添加输入参数的匿名函数

```
% 创建一些数据以传递给mycallback
data = [1, 2, 3];

% 将数据作为第三个输入传递给mycallback
set(hObject, 'Callback', @(source, event)mycallback(source, event, data))
```

或者您可以使用单元数组语法来指定回调，同样指定额外的输入参数。

```
set(hObject, 'Callback', {@mycallback, data})
```

适用场景：

- 当回调需要data来执行某些操作，但data变量不需要被修改并保存为新状态时。

第31.2节：在用户界面中制作一个暂停回调执行的按钮

有时我们希望暂停代码执行以检查应用程序的状态（参见调试）。当通过MATLAB编辑器运行代码时，可以使用用户界面中的"暂停"按钮或按 `Ctrl`+`c` （在Windows上）来实现。然而，当计算是由GUI（通过某个 uicontrol 的回调）启动时，这种方法不再有效，回调应通过另一个回调来中断。以下是该原理的演示：
该原理的演示：

```
function interruptibleUI
dbclear in interruptibleUI % 重置此文件中的断点
figure('Position',[400,500,329,160]);
```



```
        % Increment the number of clicks
        nClicks = nClicks + 1;

        % Print the number of clicks so far
        fprintf('Number of clicks: %d\n', nClicks);
    end
end
```

Best for:

- Small, simple GUIs. (for quick prototyping, to not have to implement the `guidata` and/or `set`/`getappdata` methods).

Not recommended for:

- Medium, large or complex GUIs.
- GUI created with `GUIDE`.

Explicit input arguments

If you need to send data to a callback function and don't need to modify the data within the callback, you can always consider passing the data to the callback using a carefully crafted callback definition.

You could use an anonymous function which adds inputs

```
% Create some data to send to mycallback
data = [1, 2, 3];

% Pass data as a third input to mycallback
set(hObject, 'Callback', @(source, event)mycallback(source, event, data))
```

Or you could use the cell array syntax to specify a callback, again specifying additional inputs.

```
set(hObject, 'Callback', {@mycallback, data})
```

Best for:

- When the callback needs `data` to perform some operations but the `data` variable does not need to be modified and saved in a new state.

Section 31.2: Making a button in your UI that pauses callback execution

Sometimes we'd like to pause code execution to inspect the state of the application (see Debugging). When running code through the MATLAB editor, this can be done using the "Pause" button in the UI or by pressing `Ctrl`+`c` (on Windows). However, when a computation was initiated from a GUI (via the callback of some `uicontrol`), this method does not work anymore, and the callback should be *interrupted* via another callback. Below is a demonstration of this principle:

```
function interruptibleUI
dbclear in interruptibleUI % reset breakpoints in this file
figure('Position',[400,500,329,160]);
```

```
uicontrol('Style', 'pushbutton',...
          'String', '计算',...
          "位置", [24 55 131 63],...
          '回调', @longComputation,...
          '可中断','开启'); % 默认也是'开启'

uicontrol('Style', 'pushbutton',...
          '字符串', '暂停 #1',...
          '位置', [180 87 131 63],...
          '回调', @interrupt1);

uicontrol('Style', 'pushbutton',...
          '字符串', '暂停 #2',...
          '位置', [180 12 131 63],...
          '回调', @interrupt2);

end

函数 longComputation(src,event)
  superSecretVar = rand(1);
  暂停(15);
  print('done!'); % 我们将用它来判断代码是否继续在"后台"运行。
end

function interrupt1(src,event) % 根据你想停止的位置
  dbstop 在 interruptibleUI 的第 27 行 % 会在 print('done!'); 之后停止
  dbstop 在 interruptibleUI 的第 32 行 % 会在**这一**行之后停止。
end

function interrupt2(src,event) % 方法2
  keyboard;
  dbup; % 代码在上一行停止后，需要手动执行此命令。
end
```

为了确保你理解这个示例，请执行以下操作：

1. 将上述代码粘贴到一个新文件中，并保存为interruptibleUI.m，确保代码从 文件的第一行开始（这对第一种方法的有效性很重要）。
2. 运行脚本。
3. 点击 计算 然后不久点击任一 暂停 #1 或开启暂停 #2.
4. 确保你能找到变量 superSecretVar 的值。

第31.3节：使用"handles"结构传递数据

这是一个基本GUI的示例，包含两个按钮，用于更改存储在GUI的 handles 结构中的值。

```
function gui_passing_data()
    % 一个带有两个按钮的基本GUI，用于展示在GUI构建中'handles'
    % 结构的简单用法

    % 创建一个新图形窗口。
f = figure();

    % 获取句柄结构体
    handles = guidata(f);

    % 存储图形窗口句柄
    handles.figure = f;
```



```
uicontrol('Style', 'pushbutton',...
          'String', 'Compute',...
          'Position', [24 55 131 63],...
          'Callback', @longComputation,...
          'Interruptible','on'); % 'on' by default anyway

uicontrol('Style', 'pushbutton',...
          'String', 'Pause #1',...
          'Position', [180 87 131 63],...
          'Callback', @interrupt1);

uicontrol('Style', 'pushbutton',...
          'String', 'Pause #2',...
          'Position', [180 12 131 63],...
          'Callback', @interrupt2);

end

function longComputation(src,event)
  superSecretVar = rand(1);
  pause(15);
  print('done!'); % we'll use this to determine if code kept running "in the background".
end

function interrupt1(src,event) % depending on where you want to stop
  dbstop in interruptibleUI at 27 % will stop after print('done!');
  dbstop in interruptibleUI at 32 % will stop after **this** line.
end

function interrupt2(src,event) % method 2
  keyboard;
  dbup; % this will need to be executed manually once the code stops on the previous line.
end
```

To make sure you understand this example do the following:

1. Paste the above code into a new file called and save it as `interruptibleUI.m`, such that the code starts on the **very first line** of the file (this is important for the 1st method to work).
2. Run the script.
3. Click `Compute` and shortly afterwards click either `Pause #1` or on `Pause #2`.
4. Make sure you can find the value of `superSecretVar`.

Section 31.3: Passing data around using the "handles" structure

This is an example of a basic GUI with two buttons that change a value stored in the GUI's `handles` structure.

```
function gui_passing_data()
    % A basic GUI with two buttons to show a simple use of the 'handles'
    % structure in GUI building

    %  Create a new figure.
    f = figure();

    % Retrieve the handles structure
    handles = guidata(f);

    % Store the figure handle
    handles.figure = f;
```

```
    % 创建一个编辑框和两个按钮（加号和减号），
    % 并存储它们的句柄以备后用
handles.hedit  = uicontrol('Style','edit','Position',[10,200,60,20] , 'Enable', 'Inactive');

    handles.hbutton_plus  = uicontrol('Style','pushbutton','String','+',...
                'Position',[80,200,60,20] , 'Callback' , @ButtonPress);

handles.hbutton_minus  = uicontrol('Style','pushbutton','String','-',...
                'Position',[150,200,60,20] , 'Callback' , @ButtonPress);

    % 定义一个初始值，将其存储在句柄结构体中并显示
    % 在编辑框中
handles.value = 1;
    set(handles.hedit , 'String' , num2str(handles.value))

    % 存储 handles
    guidata(f, handles);



function  ButtonPress(hObject, eventdata)
    % 按钮被按下
    % 获取 handles
    handles = guidata(hObject);

    % 判断哪个按钮被按下；hObject 是调用对象
    switch(get(hObject , 'String'))
        case '+'
            % 值加 1
handles.value = handles.value + 1;
            set(handles.hedit , 'String', num2str(handles.value))
        case '-'
            % 值减 1
handles.value = handles.value - 1;
    end

    % 显示新值
    set(handles.hedit , 'String', num2str(handles.value))

    % 存储 handles
guidata(hObject, handles);
```

要测试此示例，请将其保存为名为gui_passing_data.m的文件，并按F5启动。请注意，在这种简单情况下，您甚至不需要将值存储在 handles 结构中，因为您可以直接从编辑框的String属性访问它。

第31.4节：传递数据时的性能问题用户界面

有两种主要技术允许在GUI函数和回调之间传递数据：setappdata/getappdata 和 guidata（了解更多）。前者应当用于较大的变量，因为它更节省时间。以下示例测试了这两种方法的效率。

创建了一个带有简单按钮的GUI，并用 guidata 和 setappdata 两种方法分别存储了一个大变量（10000x10000 双精度矩阵）。按钮使用这两种方法重新加载并存储该变量，同时计时它们的执行时间。运行时间和使用 setappdata 的百分比提升将在命令窗口显示。

```
function gui_passing_data_performance()
```



```
    % Create an edit box and two buttons (plus and minus),
    % and store their handles for future use
    handles.hedit  = uicontrol('Style','edit','Position',[10,200,60,20] , 'Enable', 'Inactive');

    handles.hbutton_plus  = uicontrol('Style','pushbutton','String','+',...
              'Position',[80,200,60,20] , 'Callback' , @ButtonPress);

    handles.hbutton_minus  = uicontrol('Style','pushbutton','String','-',...
              'Position',[150,200,60,20] , 'Callback' , @ButtonPress);

    % Define an initial value, store it in the handles structure and show
    % it in the Edit box
    handles.value = 1;
    set(handles.hedit , 'String' , num2str(handles.value))

    % Store handles
    guidata(f, handles);



function  ButtonPress(hObject, eventdata)
    % A button was pressed
    % Retrieve the handles
    handles = guidata(hObject);

    % Determine which button was pressed; hObject is the calling object
    switch(get(hObject , 'String'))
        case '+'
            % Add 1 to the value
            handles.value = handles.value + 1;
            set(handles.hedit , 'String', num2str(handles.value))
        case '-'
            % Subtract 1 from the value
            handles.value = handles.value - 1;
    end

    % Display the new value
    set(handles.hedit , 'String', num2str(handles.value))

    % Store handles
    guidata(hObject, handles);
```

To test the example, save it in a file called `gui_passing_data.m` and launch it with F5. Please note that in such a simple case, you would not even need to store the value in the handles structure because you could directly access it from the edit box's `String` property.

Section 31.4: Performance Issues when Passing Data Around User Interface

Two main techniques allow passing data between GUI functions and Callbacks: setappdata/getappdata and guidata (read more about it). The former should be used for larger variables as it is more time efficient. The following example tests the two methods' efficiency.

A GUI with a simple button is created and a large variable (10000x10000 double) is stored both with guidata and with setappdata. The button reloads and stores back the variable using the two methods while timing their execution. The running time and percentage improvement using setappdata are displayed in the command window.

```
function gui_passing_data_performance()
```

```
    % 一个带按钮的基础GUI，用于显示
    % guidata 和 setappdata 之间的性能差异

    % 创建一个新图形窗口。
f = figure('Units' , 'normalized');

    % 获取句柄结构体
    handles = guidata(f);

    % 存储图形句柄
handles.figure = f;

                        handles.hbutton = uicontrol('样式','pushbutton','字符串','计算','单位','归一化',...
                    '位置',[0.4 , 0.45 , 0.2 , 0.1] , '回调' , @ButtonPress);

    % 创建一个无趣的大数组
    data = zeros(10000);

    % 存储到应用数据中
setappdata(handles.figure , 'data' , data);

    % 存储到 handles 中
    handles.data = data;

    % 保存 handles
    guidata(f, handles);



function   ButtonPress(hObject, eventdata)

    % 计算使用 guidata 和 appdata 时的时间差
    t_handles = timeit(@use_handles);
    t_appdata = timeit(@use_appdata);

    % 绝对差和百分比差
    t_diff = t_handles - t_appdata;
    t_perc = round(t_diff / t_handles * 100);

    disp(['差异: ' num2str(t_diff) ' 毫秒 / ' num2str(t_perc) ' %'])



function   use_appdata()

    % 从 appdata 中获取数据
    data = getappdata(gcf , 'data');

    % 对数据进行某些操作 %

    % 再次存储该值
    setappdata(gcf , 'data' , data);


function  use_handles()

    % 从 handles 中获取数据
    handles = guidata(gcf);
    data = handles.data;

    % 对数据进行某些操作 %
```



```
    % A basic GUI with a button to show performance difference between
    % guidata and setappdata

    %  Create a new figure.
    f = figure('Units' , 'normalized');

    % Retrieve the handles structure
    handles = guidata(f);

    % Store the figure handle
    handles.figure = f;

    handles.hbutton = uicontrol('Style','pushbutton','String','Calculate','units','normalized',...
              'Position',[0.4 , 0.45 , 0.2 , 0.1] , 'Callback' , @ButtonPress);

    % Create an uninteresting large array
    data = zeros(10000);

    % Store it in appdata
    setappdata(handles.figure , 'data' , data);

    % Store it in handles
    handles.data = data;

    % Save handles
    guidata(f, handles);



function  ButtonPress(hObject, eventdata)

    % Calculate the time difference when using guidata and appdata
    t_handles = timeit(@use_handles);
    t_appdata = timeit(@use_appdata);

    % Absolute and percentage difference
    t_diff = t_handles - t_appdata;
    t_perc = round(t_diff / t_handles * 100);

    disp(['Difference: ' num2str(t_diff) ' ms / ' num2str(t_perc) ' %'])



function  use_appdata()

    % Retrieve the data from appdata
    data = getappdata(gcf , 'data');

    % Do something with data %

    % Store the value again
    setappdata(gcf , 'data' , data);


function use_handles()

    % Retrieve the data from handles
    handles = guidata(gcf);
    data = handles.data;

    % Do something with data %
```

```
% 将其存回句柄中
handles.data = data;
guidata(gcf, handles);
```

在我的 Xeon W3530@2.80 GHz 上，我得到的差异：`0.00018957`毫秒 / `73 %`，因此使用 getappdata/setappdata 我获得了 73% 的性能提升！注意，如果使用 10x10 的双精度变量，结果不会改变，
但是，如果`handles`包含许多带有大量数据的字段，结果将会改变。


```
% Store it back in the handles
handles.data = data;
guidata(gcf, handles);
```

On my Xeon W3530@2.80 GHz I get `Difference: 0.00018957 ms / 73 %`, thus using getappdata/setappdata I get a performance improvement of 73%! Note that the result does not change if a 10x10 double variable is used, however, result will change if `handles` contains many fields with large data.

第32章：实用技巧

第32.1节：提取图形数据

有几次，我保存了一个有趣的图形，但失去了对其数据的访问权限。这个例子展示了
如何从图形中提取信息的技巧。

关键函数是findobj和get。 findobj根据对象的属性或特性返回对象的句柄，例如Type或Color等。一旦找到线对象，get可以返回属性所持有的任何值。事实证明，Line对象在以下属性中保存所有数据：XData、YData和ZData；最后一个通常为0，除非图形包含三维绘图。

下面的代码创建了一个示例图形，显示了两条线：一个正弦函数和一个阈值线，以及一个图例

```
t = (0:1/10:1-1/10)';
y = sin(2*pi*t);
plot(t,y);
hold on;
plot([0 0.9],[0 0], 'k-');
hold off;
legend({'sin' 'threshold'});
```

第一次使用findobj返回两个句柄，分别对应两条线：

```
findobj(gcf, '类型', '线')
ans =
2x1 线 数组：

  线    (threshold)
  线    (sin)
```

为了缩小结果范围，findobj也可以使用逻辑运算符-and、-or和属性名的组合。例如，我可以找到一个其DisplayName为'sin'的线对象，并读取它的XData和YData。

```
lineh = findobj(gcf, '类型', '线', '-and', 'DisplayName', 'sin');
xdata = get(lineh, 'XData');
ydata = get(lineh, 'YData');
```

并检查数据是否相等。

```
isequal(t(:),xdata(:))
ans =
     1
isequal(y(:),ydata(:))
ans =
     1
```

同样，我可以通过排除黑线（阈值）来缩小结果范围：

```
lineh = findobj(gcf, '类型', '线', '-not', '颜色', 'k');
xdata = get(lineh, 'X数据');
ydata = get(lineh, 'YData');
```

最后检查确认从该图中提取的数据是相同的：

Chapter 32: Useful tricks

Section 32.1: Extract figure data

On a few occasions, I have had an interesting figure I saved but I lost an access to its data. This example shows a trick how to achieve extract information from a figure.

The key functions are findobj and get. findobj returns a handler to an object given attributes or properties of the object, such as `Type` or `Color`, etc. Once a line object has been found, get can return any value held by properties. It turns out that the `Line` objects hold all data in following properties: `XData`, `YData`, and `ZData`; the last one is usually 0 unless a figure contains a 3D plot.

The following code creates an example figure that shows two lines a sin function and a threshold and a legend

```
t = (0:1/10:1-1/10)';
y = sin(2*pi*t);
plot(t,y);
hold on;
plot([0 0.9],[0 0], 'k-');
hold off;
legend({'sin' 'threshold'});
```

The first use of findobj returns two handlers to both lines:

```
findobj(gcf, 'Type', 'Line')
ans =
  2x1 Line array:

  Line    (threshold)
  Line    (sin)
```

To narrow the result, findobj can also use combination of logical operators `-and`, `-or` and property names. For instance, I can find a line object whose `DiplayName` is `sin` and read its `XData` and `YData`.

```
lineh = findobj(gcf, 'Type', 'Line', '-and', 'DisplayName', 'sin');
xdata = get(lineh, 'XData');
ydata = get(lineh, 'YData');
```

and check if the data are equal.

```
isequal(t(:),xdata(:))
ans =
     1
isequal(y(:),ydata(:))
ans =
     1
```

Similarly, I can narrow my results by excluding the black line (threshold):

```
lineh = findobj(gcf, 'Type', 'Line', '-not', 'Color', 'k');
xdata = get(lineh, 'XData');
ydata = get(lineh, 'YData');
```

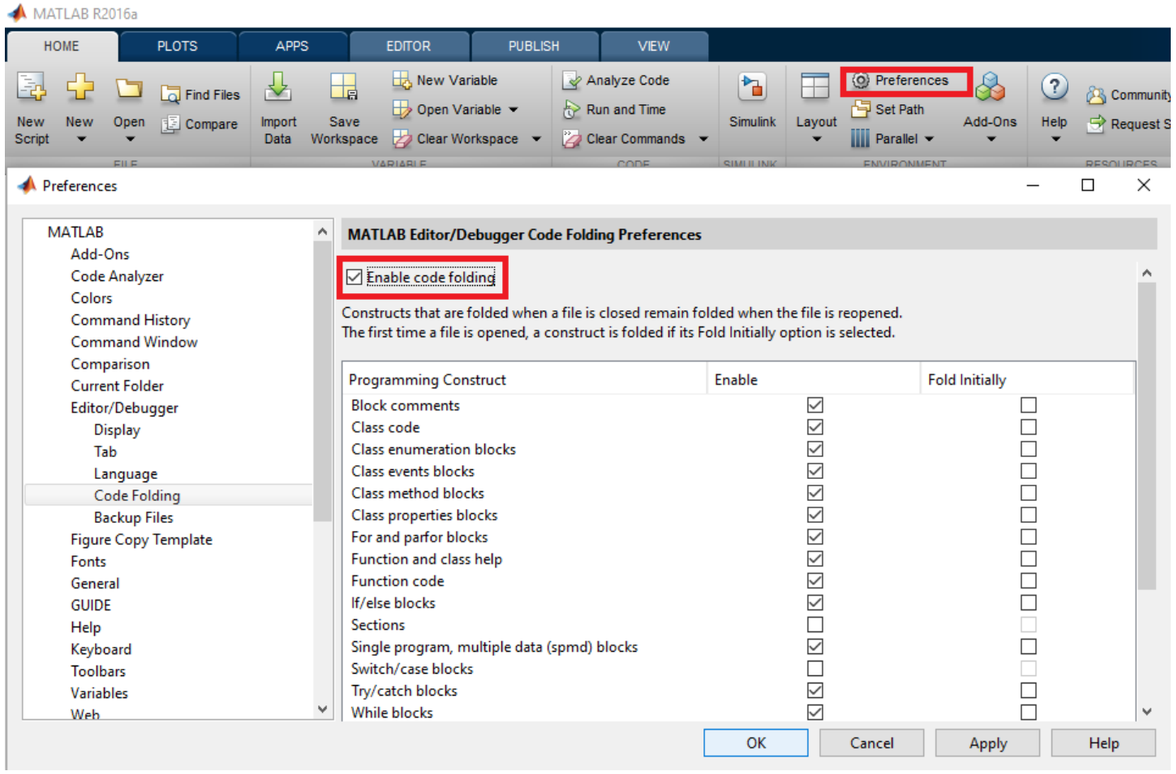
and last check confirms that data extracted from this figure are the same:

```
isequal(t(:),xdata(:))
ans =
     1
isequal(y(:),ydata(:))
ans =
     1
```

第32.2节：代码折叠偏好设置

可以根据需要更改代码折叠偏好设置。因此，代码折叠可以针对特定结构（例如：if块，for循环，章节等）设置为启用或禁用。

要更改折叠偏好设置，请进入“偏好设置”->“代码折叠”：



然后您可以选择代码的哪些部分可以折叠。

一些信息：

- 请注意，您还可以通过将光标放置在文件中的任意位置，右键点击，然后从上下文菜单中选择“代码折叠 > 全部展开”或“代码折叠 > 全部折叠”来展开或折叠文件中的所有代码。
- 请注意，折叠是持久的，意味着已经展开/折叠的代码部分在 MATLAB 或 m 文件关闭并重新打开后仍会保持其状态。

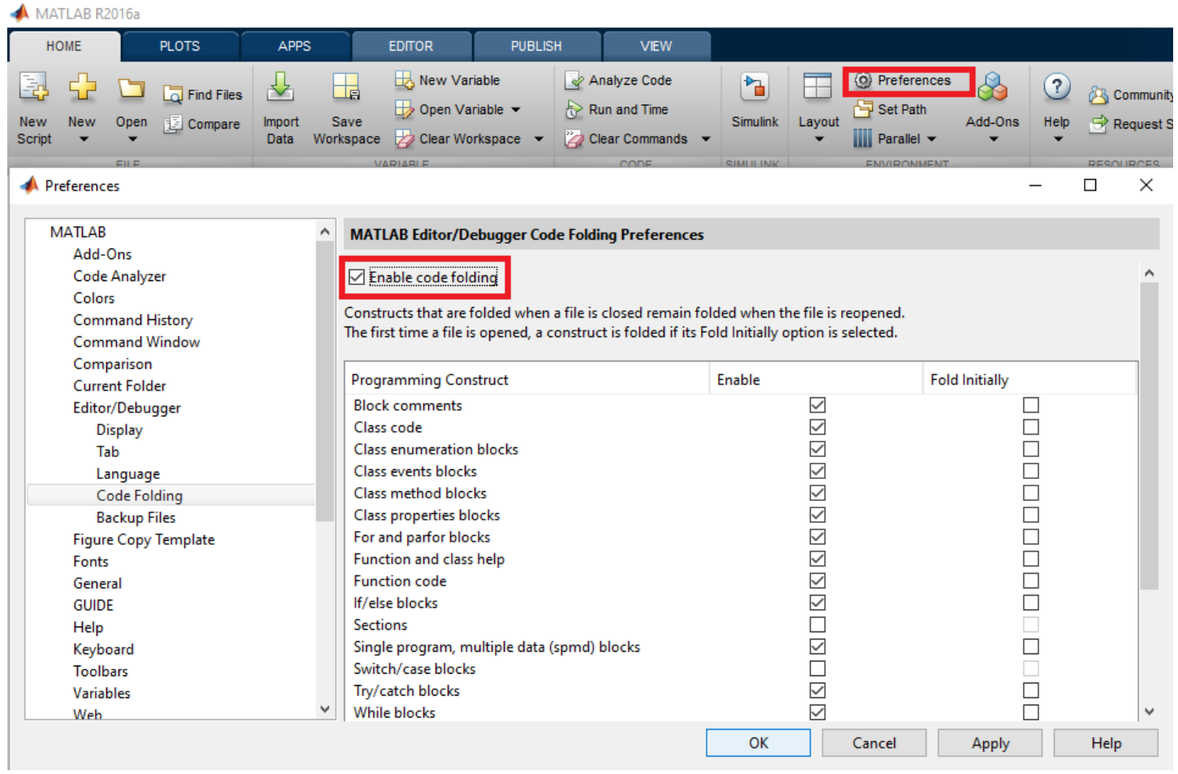
> **示例：启用代码段折叠：**
>
> 一个有趣的选项是启用折叠代码段。代码段由两个百分号（%%）分隔。


```
isequal(t(:),xdata(:))
ans =
     1
isequal(y(:),ydata(:))
ans =
     1
```

Section 32.2: Code Folding Preferences

It is possible to change Code Folding preference to suit your need. Thus code folding can be set enable/unable for specific constructs (ex: `if` block, `for` loop, `Sections` ...).

To change folding preferences, go to Preferences -> Code Folding:



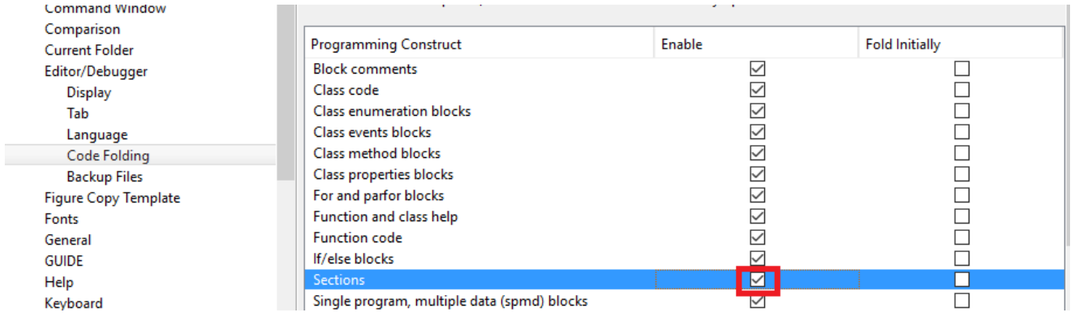
Then you can choose which part of the code can be folded.

Some information:

- Note that you can also expand or collapse all of the code in a file by placing your cursor anywhere within the file, right-click, and then select Code Folding > Expand All or Code Folding > Fold All from the context menu.
- Note that folding is persistent, in the sense that part of the code that has been expanded/collapsed will keep their status after MATLAB or the m-file has been closed and is re-open.

> **Example: To enable folding for sections:**
>
> An interesting option is to enable to fold Sections. Sections are delimited by two percent signs (%%).

示例：要启用它，请勾选“代码段”框：

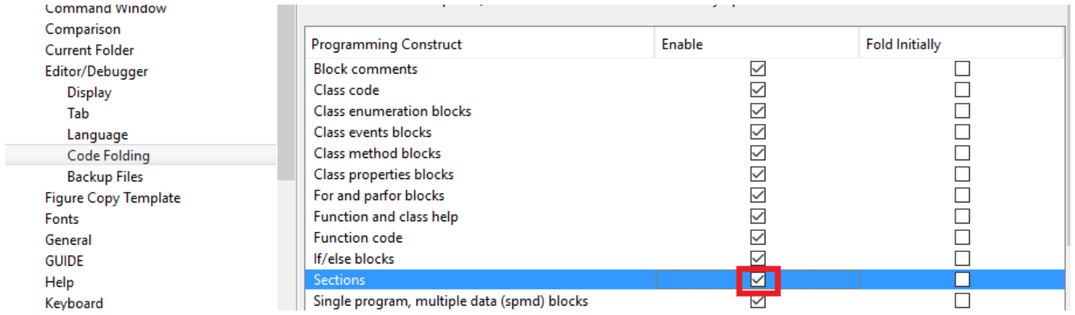


然后你将不再看到类似于以下的长源代码：

```
%% LOAD

        %code to load data
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...

%% TREAT
        % code to run the model
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...

%% OUTPUT

        % code to output results
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
```

你将能够折叠代码段，以便对代码有一个总体的概览：

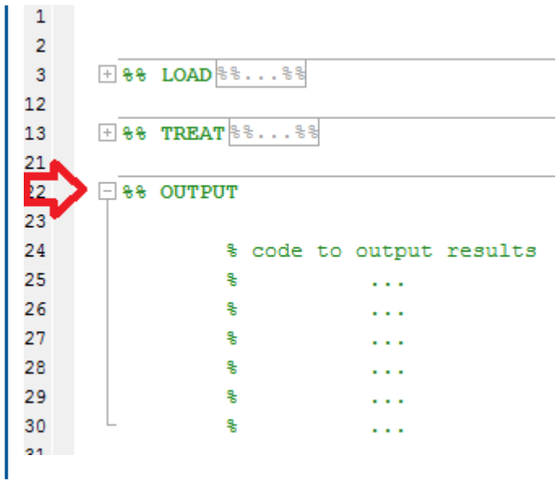
Example: To enable it check the "Sections" box:



Then instead of seeing a long source code similar to :

```
%% LOAD

        %code to load data
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...

%% TREAT
        % code to run the model
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...

%% OUTPUT

        % code to output results
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
        %           ...
```

You will be able to fold sections to have a general overview of your code :



第32.3节：使用匿名函数的函数式编程

匿名函数可用于函数式编程。主要问题是没有原生方式来锚定递归，但这仍然可以用一行代码实现：

```
if_ = @(bool, tf) tf{2-bool}();
```

该函数接受一个布尔值和一个包含两个函数的单元数组。如果布尔值为真，则执行第一个函数；如果布尔值为假，则执行第二个函数。现在我们可以轻松地编写阶乘函数：

```
fac = @(n,f) if_(n>1, {@()n*f(n-1,f), @()1});
```

这里的问题是我们不能直接调用递归，因为在右侧表达式被求值时，函数尚未赋值给变量。
不过我们可以通过编写以下代码来完成这一步：

```
factorial_ = @(n)fac(n,fac);
```

现在@(n)fac(n,fac)递归地计算阶乘函数。另一种在函数式编程中实现的方法是使用y组合子，这也可以很容易地实现：

```
y_ = @(f)@(n)f(n,f);
```

使用这个工具，阶乘函数甚至更简短：

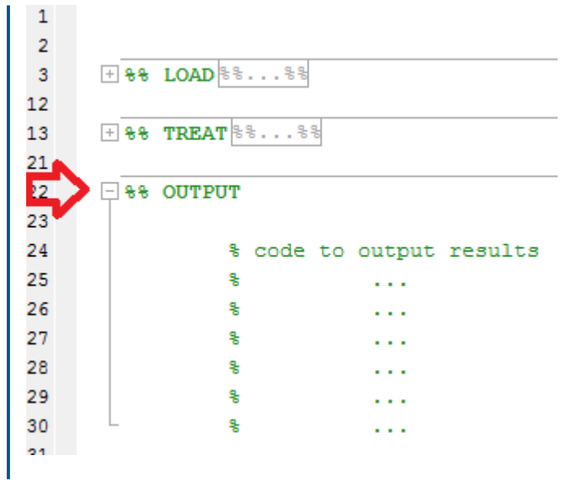
```
factorial_ = y_(fac);
```

或者直接写成：

```
factorial_ = y_(@(n,f) if_(n>1, {@()n*f(n-1,f), @()1}));
```

第32.4节：将多个图形保存到同一个.fig文件中

通过将多个图形句柄放入图形数组，可以将多个图形保存到同一个 .fig 文件中



Section 32.3: Functional Programming using Anonymous Functions

Anonymous functions can be used for functional programming. The main problem to solve is that there is no native way for anchoring a recursion, but this can still be implemented in a single line:

```
if_ = @(bool, tf) tf{2-bool}();
```

This function accepts a boolean value and a cell array of two functions. The first of those functions is evaluated if the boolean value evaluates as true, and the second one if the boolean value evaluates as false. We can easily write the factorial function now:

```
fac = @(n,f) if_(n>1, {@()n*f(n-1,f), @()1});
```

The problem here is that we cannot directly invoke a recursive call, as the function is not yet assigned to a variable when the right hand side is evaluated. We can however complete this step by writing

```
factorial_ = @(n)fac(n,fac);
```

Now `@(n)fac(n,fac)` evaluates the factorial function recursively. Another way to do this in functional programming using a y-combinator, which also can easily be implemented:

```
y_ = @(f)@(n)f(n,f);
```

With this tool, the factorial function is even shorter:

```
factorial_ = y_(fac);
```

Or directly:

```
factorial_ = y_(@(n,f) if_(n>1, {@()n*f(n-1,f), @()1}));
```

Section 32.4: Save multiple figures to the same .fig file

By putting multiple figure handles into a graphics array, multiple figures can be saved to the same .fig file

```
h(1) = figure;
scatter(rand(1,100),rand(1,100));

h(2) = figure;
scatter(rand(1,100),rand(1,100));

h(3) = figure;
scatter(rand(1,100),rand(1,100));

savefig(h,'ThreeRandomScatterplots.fig');
close(h);
```

这会创建3个随机数据的散点图，每个都是图形数组 h 的一部分。然后可以像保存普通图形一样使用 savefig 保存图形数组，但需要将图形数组的句柄作为额外参数传入。

一个有趣的附带说明是，当你打开这些图形时，它们通常会保持保存时的排列方式。

第32.5节：注释块

如果你想注释代码的一部分，注释块可能会很有用。注释块以新行中的 %{ 开始，以另一新行中的 %} 结束：

```
a = 10;
b = 3;
%{
c = a*b;
d = a-b;
%}
```

这允许你折叠你注释的代码段，使代码更加整洁紧凑。

这些代码块也有助于切换代码的部分开关。你只需在开始处再添加一个%即可取消注释该代码块：

```
a = 10;
b = 3;
%%{ <-- 这里再加一个 %
c = a*b;
d = a-b;
%}
```

有时你想注释掉一段代码，但又不影响其缩进：

```
for k = 1:a
    b = b*k;
    c = c-b;
    d = d*c;
     disp(b)
end
```

通常，当你选中一段代码并按下 Ctrl + r 来注释它（即添加%）时自动应用到所有行，然后当你稍后按下 Ctrl + i 进行自动缩进时，代码块会从其正确的层级位置移动，且向右移动过多：

```
for k = 1:a
    b = b*k;
```



```
h(1) = figure;
scatter(rand(1,100),rand(1,100));

h(2) = figure;
scatter(rand(1,100),rand(1,100));

h(3) = figure;
scatter(rand(1,100),rand(1,100));

savefig(h,'ThreeRandomScatterplots.fig');
close(h);
```

This creates 3 scatterplots of random data, each part of graphic array h. Then the graphics array can be saved using savefig like with a normal figure, but with the handle to the graphics array as an additional argument.

An interesting side note is that the figures will tend to stay arranged in the same way that they were saved when you open them.

Section 32.5: Comment blocks

If you want to comment part of your code, then comment blocks may be useful. Comment block starts with a %{ in a new line and ends with %} in another new line:

```
a = 10;
b = 3;
%{
c = a*b;
d = a-b;
%}
```

This allows you to fold the sections that you commented to make the code more clean and compact.

These blocks are also useful for toggling on/off parts of your code. All you have to do to uncomment the block is add another % before it starts:

```
a = 10;
b = 3;
%%{ <-- another % over here
c = a*b;
d = a-b;
%}
```

Sometimes you want to comment out a section of the code, but without affecting its indentation:

```
for k = 1:a
    b = b*k;
    c = c-b;
    d = d*c;
    disp(b)
end
```

Usually, when you mark a block of code and press Ctrl + r for commenting it out (by that adding the % automatically to all lines, then when you press later Ctrl + i for auto indentation, the block of code moves from its correct hierarchical place, and moved too much to the right:

```
for k = 1:a
    b = b*k;
```

```
    %     c = c-b;
    %     d = d*c;
    disp(b)
end
```

解决方法之一是使用注释块，这样块的内部部分可以保持正确的缩进：

```
for k = 1:a
    b = b*k;
    %{
c = c-b;
    d = d*c;
    %}
    disp(b)
end
```

第32.6节：对单元格和数组操作的有用函数

这个简单的例子解释了一些自从我开始使用MATLAB以来发现非常有用的函数：cellfun，arrayfun。其思想是对一个数组或单元格类变量，遍历其所有元素，并对每个元素应用一个专门的函数。所应用的函数可以是匿名函数，这通常是常见情况，或者是定义在*.m文件中的任何常规函数。

让我们从一个简单的问题开始，假设我们需要根据文件夹找到一组 *.mat 文件。以这个例子为例，首先在当前文件夹中创建一些 *.mat 文件：

```
for n=1:10; save(sprintf('mymatfile%d.mat',n)); end
```

执行代码后，应该会有 10 个新的 *.mat 文件。如果我们运行一个命令来列出所有 *.mat 文件，例如：

```
mydir = dir('*.mat');
```

我们应该得到一个 dir 结构体元素的数组；MATLAB 应该会给出类似如下的输出：

```
10x1 struct 数组，包含字段：
name
    date
bytes
    isdir
    datenum
```

如你所见，这个数组的每个元素都是一个包含若干字段的结构体。关于每个文件的所有信息确实都很重要，但在 99% 的情况下，我更关心的是文件名而已。为了从结构体数组中提取信息，我过去通常会创建一个局部函数，涉及创建合适大小的临时变量、for 循环、从每个元素中提取名称并保存到创建的变量中。实现完全相同结果的更简单方法是使用上述函数之一：

```
mydirlist = arrayfun(@(x) x.name, dir('*.mat'), 'UniformOutput', false)
mydirlist =
    'mymatfile1.mat'
    'mymatfile10.mat'
    'mymatfile2.mat'
    'mymatfile3.mat'
    'mymatfile4.mat'
```



```
    %     c = c-b;
    %     d = d*c;
    disp(b)
end
```

A way to solve this is to use comment blocks, so the inner part of the block stays correctly indented:

```
for k = 1:a
    b = b*k;
    %{
    c = c-b;
    d = d*c;
    %}
    disp(b)
end
```

Section 32.6: Useful functions that operate on cells and arrays

This simple example provides an explanation on some functions I found extremely useful since I have started using MATLAB: `cellfun`, `arrayfun`. The idea is to take an array or cell class variable, loop through all its elements and apply a dedicated function on each element. An applied function can either be anonymous, which is usually a case, or any regular function define in a *.m file.

Let's start with a simple problem and say we need to find a list of *.mat files given the folder. For this example, first let's create some *.mat files in a current folder:

```
for n=1:10; save(sprintf('mymatfile%d.mat',n)); end
```

After executing the code, there should be 10 new files with extension *.mat. If we run a command to list all *.mat files, such as:

```
mydir = dir('*.mat');
```

we should get an array of elements of a dir structure; MATLAB should give a similar output to this one:

```
10x1 struct array with fields:
    name
    date
    bytes
    isdir
    datenum
```

As you can see each element of this array is a structure with couple of fields. All information are indeed important regarding each file but in 99% I am rather interested in file names and nothing else. To extract information from a structure array, I used to create a local function that would involve creating temporal variables of a correct size, for loops, extracting a name from each element, and save it to created variable. Much easier way to achieve exactly the same result is to use one of the aforementioned functions:

```
mydirlist = arrayfun(@(x) x.name, dir('*.mat'), 'UniformOutput', false)
mydirlist =
    'mymatfile1.mat'
    'mymatfile10.mat'
    'mymatfile2.mat'
    'mymatfile3.mat'
    'mymatfile4.mat'
```

```
    'mymatfile5.mat'
    'mymatfile6.mat'
    'mymatfile7.mat'
    'mymatfile8.mat'
    'mymatfile9.mat'
```

这个函数是如何工作的？它通常接受两个参数：第一个参数是函数句柄，第二个参数是数组。函数随后会对给定数组的每个元素进行操作。第三和第四个参数是可选的，但很重要。如果我们知道输出不会是规则的，就必须将其保存在单元格中。必须指出的是，将UniformOutput设置为false。

默认情况下，该函数尝试返回规则的输出，例如数字向量。

例如，让我们提取每个文件占用的磁盘空间大小（以字节为单位）：

```
mydirbytes = arrayfun(@(x) x.bytes, dir('*.mat'))
mydirbytes =
       34560
       34560
       34560
       34560
       34560
       34560
       34560
       34560
       34560
       34560
```

或者以千字节为单位：

```
mydirbytes = arrayfun(@(x) x.bytes/1024, dir('*.mat'))
mydirbytes =
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
```

这次输出是一个常规的双精度向量。UniformOutput默认设置为true。

`cellfun`是一个类似的函数。该函数与`arrayfun`的区别在于`cellfun`作用于cell类变量。如果我们希望从一个cell中的文件名列表'mydirlist'中提取仅文件名部分，只需按如下方式运行该函数：

```
mydirnames = cellfun(@(x) x(1:end-4), mydirlist, 'UniformOutput', false)
mydirnames =
    'mymatfile1'
    'mymatfile10'
    'mymatfile2'
    'mymatfile3'
    'mymatfile4'
    'mymatfile5'
    'mymatfile6'
    'mymatfile7'
    'mymatfile8'
    'mymatfile9'
```



```
    'mymatfile5.mat'
    'mymatfile6.mat'
    'mymatfile7.mat'
    'mymatfile8.mat'
    'mymatfile9.mat'
```


How this function works? It usually takes two parameters: a function handle as the first parameter and an array. A function will then operate on each element of a given array. The third and fourth parameters are optional but important. If we know that an output will not be regular, it must be saved in cell. This must be point out setting `false` to `UniformOutput`. By default this function attempts to return a regular output such as a vector of numbers. For instance, let's extract information about how much of disc space is taken by each file in bytes:

```
mydirbytes = arrayfun(@(x) x.bytes, dir('*.mat'))
mydirbytes =
       34560
       34560
       34560
       34560
       34560
       34560
       34560
       34560
       34560
       34560
```

or kilobytes:

```
mydirbytes = arrayfun(@(x) x.bytes/1024, dir('*.mat'))
mydirbytes =
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
   33.7500
```

This time the output is a regular vector of double. `UniformOutput` was set to `true` by default.

`cellfun` is a similar function. The difference between this function and `arrayfun` is that `cellfun` operates on cell class variables. If we wish to extract only names given a list of file names in a cell 'mydirlist', we would just need to run this function as follows:

```
mydirnames = cellfun(@(x) x(1:end-4), mydirlist, 'UniformOutput', false)
mydirnames =
    'mymatfile1'
    'mymatfile10'
    'mymatfile2'
    'mymatfile3'
    'mymatfile4'
    'mymatfile5'
    'mymatfile6'
    'mymatfile7'
    'mymatfile8'
    'mymatfile9'
```

同样，由于输出不是常规的数字向量，输出必须保存在cell变量中。

在下面的示例中，我将两个函数合并为一个，只返回不带扩展名的文件名列表：

```
cellfun(@(x) x(1:end-4), arrayfun(@(x) x.name, dir('*.mat'), 'UniformOutput', false),
'UniformOutput', false)
ans =
    'mymatfile1'
    'mymatfile10'
    'mymatfile2'
    'mymatfile3'
    'mymatfile4'
    'mymatfile5'
    'mymatfile6'
    'mymatfile7'
    'mymatfile8'
    'mymatfile9'
```

这很疯狂但非常可能，因为arrayfun返回一个cell，这正是cellfun所期望的输入；顺便说一句，我们可以通过将UniformOutput显式设置为false，强制任一函数返回cell变量中的结果。我们总能得到cell中的结果，但可能无法得到常规向量中的结果。

还有一个类似的函数作用于结构体的字段：structfun。我个人觉得它没有前两个函数那么实用，但在某些情况下会很有用。例如，如果想知道哪些字段是数值型或非数值型，下面的代码可以给出答案：

```
structfun(@(x) 是 char 类型(x), mydir(1))
```

dir 结构的第一个和第二个字段是字符类型。因此，输出为：

```
1
1
0
0
0
```

此外，输出是一个逻辑向量，包含true / false。因此，它是规则的，可以保存在向量中；无需使用单元格类。

Again, as an output is not a regular vector of numbers, an output must be saved in a cell variable.

In the example below, I combine two functions in one and return only a list of file names without an extension:

```
cellfun(@(x) x(1:end-4), arrayfun(@(x) x.name, dir('*.mat'), 'UniformOutput', false),
'UniformOutput', false)
ans =
    'mymatfile1'
    'mymatfile10'
    'mymatfile2'
    'mymatfile3'
    'mymatfile4'
    'mymatfile5'
    'mymatfile6'
    'mymatfile7'
    'mymatfile8'
    'mymatfile9'
```

It is crazy but very possible because `arrayfun` returns a cell which is expected input of `cellfun`; a side note to this is that we can force any of those functions to return results in a cell variable by setting `UniformOutput` to false, explicitly. We can always get results in a cell. We may not be able to get results in a regular vector.

There is one more similar function that operates on fields a structure: `structfun`. I have not particularly found it as useful as the other two but it would shine in some situations. If for instance one would like to know which fields are numeric or non-numeric, the following code can give the answer:

```
structfun(@(x) ischar(x), mydir(1))
```

The first and the second field of a dir structure is of a char type. Therefore, the output is:

```
1
1
0
0
0
```

Also, the output is a logical vector of `true` / `false`. Consequently, it is regular and can be saved in a vector; no need to use a cell class.

第33章：常见错误和失误

第33.1节：转置运算符

- .' 是在 MATLAB 中转置向量或矩阵的正确方式。
- ' 是在 MATLAB 中取向量或矩阵的共轭转置（又称厄米共轭）的正确方式。

注意，对于转置运算符 .'，撇号前有一个点。这符合 MATLAB 中其他元素逐个操作的语法：*用于矩阵乘法，.*用于矩阵元素逐个相乘。这两个命令非常相似，但概念上截然不同。像其他 MATLAB 命令一样，这些运算符是运行时转换成“正式”函数调用的“语法糖”。就像 == 会被转换为 eq 函数的调用一样，.' 可以看作是 transpose 的简写。如果你只写 '（没有点），实际上你使用的是 ctranspose 命令，它计算的是共轭转置，也称为厄米共轭，物理学中常用。只要转置的向量或矩阵是实值的，这两个运算符产生相同结果。但一旦涉及复数，如果不使用“正确”的简写，必然会遇到问题。“正确”的定义取决于你的应用场景。

考虑以下包含复数的矩阵 C 的示例：

```
>> C = [1i, 2; 3*1i, 4]
C =
   0.0000 + 1.0000i   2.0000 + 0.0000i
   0.0000 + 3.0000i   4.0000 + 0.0000i
```

让我们使用简写.'（带点）来进行转置。输出符合预期，是矩阵

```
>> C.'
ans =
   0.0000 + 1.0000i   0.0000 + 3.0000i
   2.0000 + 0.0000i   4.0000 + 0.0000i
```

现在，让我们使用'（不带点）。我们看到，除了转置之外，复数值也被转换成了它们的*共轭复数*。

```
>> C'
ans =
   0.0000 - 1.0000i   0.0000 - 3.0000i
   2.0000 + 0.0000i   4.0000 + 0.0000i
```

总结一下，如果你想计算厄米共轭，即复共轭转置，使用'（不带点）。如果你只想计算转置而不对值进行复共轭，使用.'（带点）。

第33.2节：不要用已有函数名来命名变量

已经存在一个函数 sum()。因此，如果我们用相同的名字命名变量

```
sum = 1+3;
```

如果我们尝试在变量仍存在于工作区时使用该函数

Chapter 33: Common mistakes and errors

Section 33.1: The transpose operators

- `.'` is the correct way to **transpose** a vector or matrix in MATLAB.
- `'` is the correct way to take the **complex conjugate transpose** (a.k.a. Hermitian conjugate) of a vector or matrix in MATLAB.

Note that for the transpose `.'`, there is a **period** in front of the apostrophe. This is in keeping with the syntax for the other element-wise operations in MATLAB: `*` multiplies *matrices*, `.*` multiplies *elements of matrices* together. The two commands are very similar, but conceptually very distinct. Like other MATLAB commands, these operators are "syntactical sugar" that gets turned into a "proper" function call at runtime. Just as `==` becomes an evaluation of the eq function, think of `.'` as the shorthand for `transpose`. If you would only write `'` (without the point), you are in fact using the `ctranspose` command instead, which calculates the complex conjugate transpose, which is also known as the Hermitian conjugate, often used in physics. As long as the transposed vector or matrix is real-valued, the two operators produce the same result. But as soon as we deal with complex numbers, we will inevitably run into problems if we do not use the "correct" shorthand. What "correct" is depends on your application.

Consider the following example of a matrix `C` containing complex numbers:

```
>> C = [1i, 2; 3*1i, 4]
C =
   0.0000 + 1.0000i   2.0000 + 0.0000i
   0.0000 + 3.0000i   4.0000 + 0.0000i
```

Let's take the *transpose* using the shorthand `.'` (with the period). The output is as expected, the transposed form of `C`.

```
>> C.'
ans =
   0.0000 + 1.0000i   0.0000 + 3.0000i
   2.0000 + 0.0000i   4.0000 + 0.0000i
```

Now, let's use `'` (without the period). We see, that in addition to the transposition, the complex values have been transformed to their *complex conjugates* as well.

```
>> C'
ans =
   0.0000 - 1.0000i   0.0000 - 3.0000i
   2.0000 + 0.0000i   4.0000 + 0.0000i
```

To sum up, if you intend to calculate the Hermitian conjugate, the complex conjugate transpose, then use `'` (without the period). If you just want to calculate the transpose without complex-conjugating the values, use `.'` (with the period).

Section 33.2: Do not name a variable with an existing function name

There is already a function `sum()`. As a result, if we name a variable with the same name

```
sum = 1+3;
```

and if we try to use the function while the variable still exists in the workspace

```
A = rand(2);
sum(A,1)
```

我们将得到神秘的错误：

```
下标索引必须是实数正整数或逻辑值。
```

先使用clear()清除变量，然后再使用函数

```
clear sum

sum(A,1)
ans =
       1.0826       1.0279
```

我们如何检查函数是否已存在以避免此冲突？

使用which()和-all标志：

```
which sum -all
sum 是一个变量。
内置 (C:\Program Files\MATLAB\R2016aoolbox\matlab\datafun\@double\sum)    % 被变量 double
方法覆盖
...
```

此输出告诉我们，sum 首先是一个变量，以下方法（函数）被它覆盖，即 MATLAB 会优先尝试将我们的语法应用于该变量，而不是使用该方法。

第33.3节：注意浮点数不准确性

浮点数无法表示所有实数。这被称为浮点数不准确性。

浮点数有无限多个，并且它们可以无限长（例如 π），因此完美表示它们需要无限的内存。鉴于这是一个问题，设计了一种用于计算机中"实数"存储的特殊表示方法，即IEEE 754标准。简而言之，它描述了计算机如何存储这类数字，使用指数和尾数，如下所示，

```
floatnum = 符号 * 2^指数 * 尾数
```

由于每部分的位数有限，只能达到有限的精度。数值越小，可能的数值间隙越小（反之亦然！）。你可以在这个在线演示中尝试你的实数。

注意这种行为，尽量避免所有浮点数比较及其作为循环中的停止条件。以下是两个示例：

示例：错误的浮点数比较：

```
>> 0.1 + 0.1 + 0.1  == 0.3

ans =

  logical

   0
```

使用浮点数比较，如前例所示，是不良的做法。你可以通过取它们差值的绝对值并与一个（较小的）容差水平比较来克服这个问题。


```
A = rand(2);
sum(A,1)
```

we will get the cryptic **error**:

```
Subscript indices must either be real positive integers or logicals.
```

clear() the variable first and then use the function

```
clear sum

sum(A,1)
ans =
       1.0826       1.0279
```

How can we check if a function already exists to avoid this conflict?

Use which() with the -all flag:

```
which sum -all
sum is a variable.
built-in (C:\Program Files\MATLAB\R2016a\toolbox\matlab\datafun\@double\sum)    % Shadowed double
method
...
```

This output is telling us that sum is first a variable and that the following methods (functions) are shadowed by it, i.e. MATLAB will first try to apply our syntax to the variable, rather than using the method.

Section 33.3: Be aware of floating point inaccuracy

Floating-point numbers cannot represent all real numbers. This is known as floating point inaccuracy.

There are infinitely many floating points numbers and they can be infinitely long (e.g. π), thus being able to represent them perfectly would require infinitely amount of memory. Seeing this was a problem, a special representation for "real number" storage in computer was designed, the IEEE 754 standard. In short, it describes how computers store this type of numbers, with an exponent and mantissa, as,

```
floatnum = sign * 2^exponent * mantissa
```

With limited amount of bits for each of these, only a finite precision can be achieved. The smaller the number, smaller the gap between possible numbers (and vice versa!). You can try your real numbers in this online demo.

Be aware of this behavior and try to avoid all floating points comparison and their use as stopping conditions in loops. See below two examples:

Examples: Floating point comparison done WRONG:

```
>> 0.1 + 0.1 + 0.1  == 0.3

ans =

  logical

   0
```

It is poor practice to use floating point comparison as shown by the precedent example. You can overcome it by taking the absolute value of their difference and comparing it to a (small) tolerance level.

下面是另一个示例，其中浮点数被用作while循环的停止条件：

```
k = 0.1;
while k <= 0.3
  disp(num2str(k));
  k = k + 0.1;
end

% --- 输出: ---
0.1
0.2
```

它缺少最后一个预期的循环（0.3 <= 0.3）。

示例：正确进行浮点数比较：

```
x = 0.1 + 0.1 + 0.1;
y = 0.3;
tolerance = 1e-10; % 该情况下的“足够好”的容差。

if ( abs( x - y ) <= tolerance )
  disp('x == y');
否则
  disp('x ~= y');
end

% --- 输出: ---
x == y
```

需要注意的几点：

- 如预期，现在 x 和 y 被视为等价。
- 在上述示例中，容差的选择是任意的。因此，所选值可能不适用于所有情况（尤其是在处理更小的数字时）。可以使用 eps 函数智能地选择界限，即 N*eps(max(x,y))，其中 N 是某个特定问题的数值。一个合理且足够宽松的 N 选择是 1E2（尽管在上述问题中 N=1 也足够）。

进一步阅读：

有关浮点数不准确性的更多信息，请参见这些问题：

- 为什么在 MATLAB 中 24.0000 不等于 24.0000？
- 浮点运算出错了吗？

第 33.4 节：你看到的并不等于你得到的：命令窗口中的 char 与 cellstring

这是一个针对新用户的基本示例。它不着重解释 `char` 和 `cellstring` 之间的区别。

你可能会遇到想要去除字符串中的 `'`，但实际上你从未添加过它们。事实上，那些是*命令窗口*用来区分某些类型的**伪影**。

一个 字符串 会打印

```
s = 'dsadasd'
s =
```


Below is another example, where a floating point number is used as a stopping condition in a while loop:**

```
k = 0.1;
while k <= 0.3
  disp(num2str(k));
  k = k + 0.1;
end

% --- Output: ---
0.1
0.2
```

It misses the last expected loop (`0.3 <= 0.3`).

Example: Floating point comparison done RIGHT:

```
x = 0.1 + 0.1 + 0.1;
y = 0.3;
tolerance = 1e-10; % A "good enough" tolerance for this case.

if ( abs( x - y ) <= tolerance )
  disp('x == y');
else
  disp('x ~= y');
end

% --- Output: ---
x == y
```

Several things to note:

- As expected, now x and y are treated as equivalent.
- In the example above, the choice of tolerance was done arbitrarily. Thus, the chosen value might not be suitable for all cases (especially when working with much smaller numbers). Choosing the bound *intelligently* can be done using the eps function, i.e. `N*eps(max(x,y))`, where `N` is some problem-specific number. A reasonable choice for `N`, which is also permissive enough, is `1E2` (even though, in the above problem `N=1` would also suffice).

Further reading:

See these questions for more information about floating point inaccuracy:

- Why is 24.0000 not equal to 24.0000 in MATLAB?
- Is floating point math broken?

Section 33.4: What you see is NOT what you get: char vs cellstring in the command window

This a basic example aimed at new users. It does not focus on explaining the difference between `char` and `cellstring`.

It might happen that you want to get rid of the `'` in your strings, although you never added them. In fact, those are *artifacts* that the **command window** uses to distinguish between some types.

A string will print

```
s = 'dsadasd'
s =
```

```
dsadasd
```

一个 cellstring 会打印

```
c = {'dsadasd'};
c =
    'dsadasd'
```

注意，单引号和缩进是用来提醒我们c是一个cellstring而不是一个char的标记。字符串实际上包含在单元格中，即

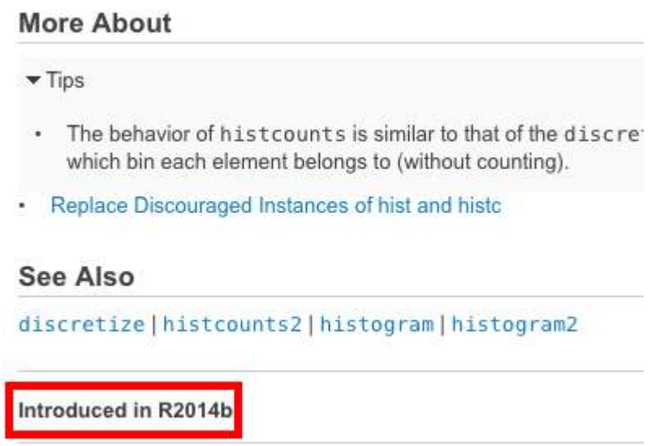
```
c{1}
ans =
dsadasd
```

第33.5节：针对类型为Y的输入参数，未定义函数或方法X

这是MATLAB委婉地表示它找不到你试图调用的函数。出现此错误的原因有很多：

该函数是在你当前版本的MATLAB之后引入的

MATLAB在线文档提供了一个非常好的功能，允许你确定某个函数是在哪个版本引入的。该功能位于文档每页的左下角：



将此版本与你当前的版本（ver）进行比较，以确定该函数是否在你所使用的版本中可用。如果不可用，尝试搜索文档的归档版本，寻找适合你版本的替代方案。

你没有那个工具箱！

基础MATLAB安装包含大量函数；然而，更专业的功能被打包在工具箱中，由MathWorks单独销售。无论你是否拥有该工具箱，所有工具箱的文档都是可见的，因此请务必检查你是否拥有相应的工具箱。

要检查某个函数属于哪个工具箱，请查看在线文档左上角是否提及了特定的工具箱。


```
dsadasd
```

A cellstring will print

```
c = {'dsadasd'};
c =
    'dsadasd'
```

Note how the **single quotes** and the **indentation** are artifacts to notify us that `c` is a `cellstring` rather than a `char`. The string is in fact contained in the cell, i.e.

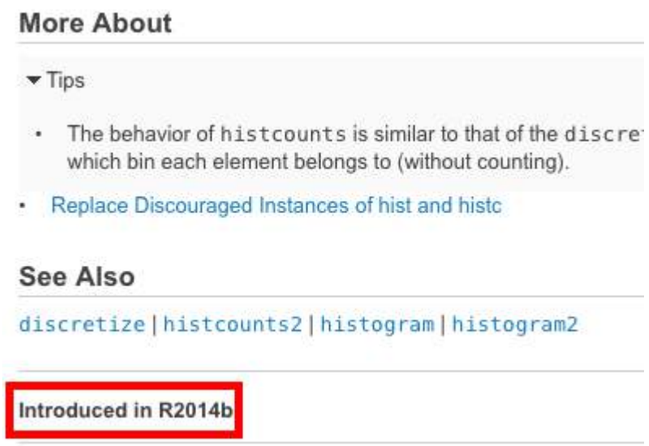
```
c{1}
ans =
dsadasd
```

Section 33.5: Undefined Function or Method X for Input Arguments of Type Y

This is MATLAB's long-winded way of saying that it cannot find the function that you're trying to call. There are a number of reasons you could get this error:

That function was introduced *after* your current version of MATLAB

The MATLAB online documentation provides a very nice feature which allows you to determine in what version a given function was introduced. It is located in the bottom left of every page of the documentation:

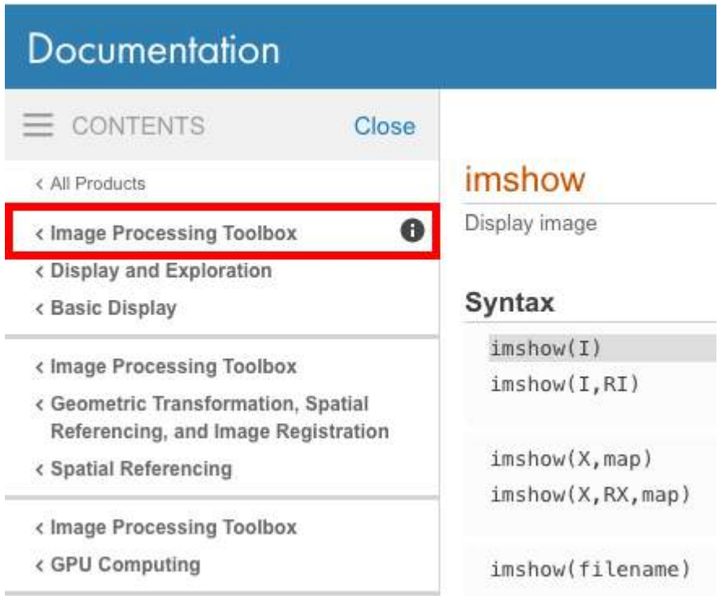


Compare this version with your own current version (`ver`) to determine if this function is available in your particular version. If it's not, try searching the archived versions of the documentation to find a suitable alternative in your version.

You don't have that toolbox!

The base MATLAB installation has a large number of functions; however, more specialized functionality is packaged within toolboxes and sold separately by MathWorks. The documentation for *all* toolboxes is visible whether you have the toolbox or not so be sure to check and see if you have the appropriate toolbox.

To check which toolbox a given function belongs to, look to the top left of the online documentation to see if a specific toolbox is mentioned.



您可以通过执行ver命令来确定您的MATLAB版本安装了哪些工具箱，该命令将打印所有已安装工具箱的列表。

如果您没有安装该工具箱但想使用该函数，则需要从MathWorks购买该特定工具箱的许可证。

MATLAB无法找到该函数

如果MATLAB仍然找不到您的函数，那么它一定是用户自定义函数。该函数可能位于另一个目录中，您需要将该目录添加到搜索路径中，才能运行您的代码。您可以使用which命令检查MATLAB是否能找到您的函数，该命令应返回源文件的路径。

第33.6节：“i”或“j”作为虚数单位、循环索引或常用变量的使用

建议

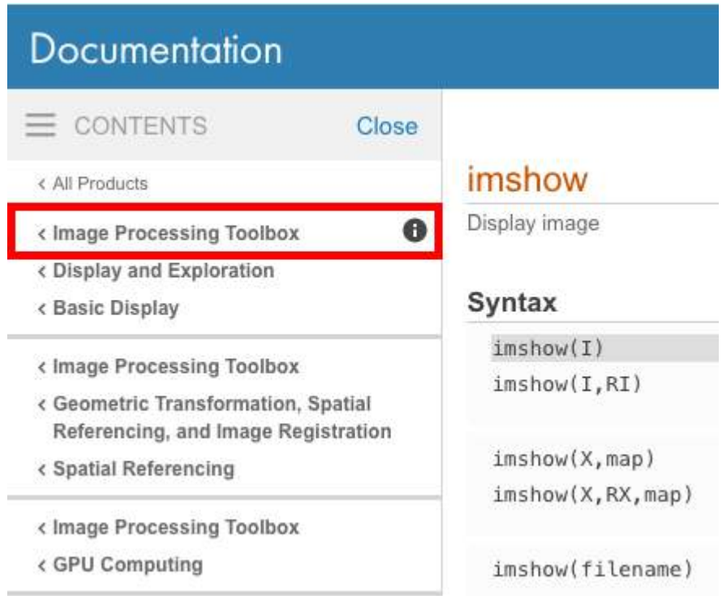
由于符号i和j在MATLAB中可能代表截然不同的含义，它们作为循环索引的使用长期以来一直在MATLAB用户社区中存在分歧。虽然一些历史上的性能原因可能使得偏向某一方，但现在情况已不同，选择完全取决于您和您选择遵循的编码规范。

MathWorks目前的官方建议是：

> - 由于i是一个函数，它可以被覆盖并用作变量。然而，如果您打算在复数运算中使用，最好避免使用i和j作为变量名。
> - 为了提高复数运算的速度和鲁棒性，请使用1i和1j代替i和j。

默认

在 MATLAB 中，默认情况下，字母 i 和 j 是内置的函数名称，它们都指代复数域中的虚数单位。



You can then determine which toolboxes your version of MATLAB has installed by issuing the ver command which will print a list of all installed toolboxes.

If you do not have that toolbox installed and want to use the function, you will need to purchase a license for that particular toolbox from MathWorks.

MATLAB cannot locate the function

If MATLAB still can't find your function, then it must be a user-defined function. It is possible that it lives in another directory and that directory should be added to the search path for your code to run. You can check whether MATLAB can locate your function by using which which should return the path to the source file.

Section 33.6: The use of "i" or "j" as imaginary unit, loop indices or common variable

Recommendation

Because the symbols `i` and `j` can represent significantly different things in MATLAB, their use as loop indices has split the MATLAB user community since ages. While some historic performance reasons could help the balance lean to one side, this is no longer the case and now the choice rest entirely on you and the coding practices you choose to follow.

The current official recommendations from MathWorks are:

> - Since `i` is a function, it can be overridden and used as a variable. However, it is best to avoid using `i` and `j` for variable names if you intend to use them in complex arithmetic.
> - For speed and improved robustness in complex arithmetic, use `1i` and `1j` instead of `i` and `j`.

Default

In MATLAB, by default, the letters `i` and `j` are built-in `function` names, which both refer to the imaginary unit in the complex domain.

因此默认情况下 `i = j = sqrt(-1)`。

```
>> i
ans =
   0.0000 + 1.0000i
>> j
ans =
   0.0000 + 1.0000i
```

正如你所预期的：

```
>> i^2
ans =
-1
```

将它们用作变量（如循环索引或其他变量）

MATLAB 允许将内置函数名用作标准变量。在这种情况下，所使用的符号将不再指向内置函数，而是指向您自己定义的变量。然而，这种做法通常不推荐，因为它可能导致混淆、调试和维护困难（参见另一个示例 do-not-name-a-variable-with-an-existing-function-name）。

如果您非常严格地遵守约定和最佳实践，您会避免在此语言中将它们用作循环索引。然而，编译器允许这样做且完全可行，因此您也可以选择保持旧习惯，将它们用作循环迭代器。

```
>> A = nan(2,3);
>> for i=1:2        % 完全合法的循环结构
       for j = 1:3
A(i, j) = 10 * i + j;
       end
   end
```

注意，循环索引在循环结束后不会超出作用域，因此它们保持新的值。

```
>> [ i ; j ]
ans =
     2
     3
```

如果您将它们用作变量，请确保它们已初始化后再使用。在上面的循环中，MATLAB 在准备循环时会自动初始化它们，但如果未正确初始化，您很快会发现可能无意中在结果中引入了复数。

如果之后您需要撤销对内置函数的遮蔽（例如，您希望 `i` 和 `j` 再次表示虚数单位），可以clear这些变量：

```
>> clear i j
```

你现在应该理解了MathWorks关于将它们用作循环索引的保留意见如果你打算在复杂算术中使用它们。你的代码将充满变量初始化和clear命令，这最容易让最严肃的程序员（是的，就是你！...）感到困惑，也容易导致程序意外发生。

如果不涉及复杂的算术运算，使用 `i` 和 `j` 完全可行，并且不会有性能损失。

So by default `i = j = sqrt(-1)`.

```
>> i
ans =
   0.0000 + 1.0000i
>> j
ans =
   0.0000 + 1.0000i
```

and as you should expect:

```
>> i^2
ans =
    -1
```

Using them as a variable (for loop indices or other variable)

MATLAB allows using built-in function name as a standard variable. In this case the symbol used will not point to the built-in function any more but to your own user defined variable. This practice, however, is not generally recommended as it can lead to confusion, difficult debugging and maintenance (*see other example do-not-name-a-variable-with-an-existing-function-name*).

If you are ultra pedantic about respecting conventions and best practices, you will avoid using them as loop indices in this language. However, it is allowed by the compiler and perfectly functional so you may also choose to keep old habits and use them as loop iterators.

```
>> A = nan(2,3);
>> for i=1:2        % perfectly legal loop construction
       for j = 1:3
        A(i, j) = 10 * i + j;
       end
   end
```

Note that loop indices do not go out of scope at the end of the loop, so they keep their new value.

```
>> [ i ; j ]
ans =
     2
     3
```

In the case you use them as variable, make sure **they are initialised** before they are used. In the loop above MATLAB initialise them automatically when it prepare the loop, but if not initialised properly you can quickly see that you may inadvertently introduce `complex` numbers in your result.

If later on, you need to undo the shadowing of the built-in function (=e.g. you want `i` and `j` to represent the imaginary unit again), you can `clear` the variables:

```
>> clear i j
```

You understand now the MathWorks reservation about using them as loop indices *if you intend to use them in complex arithmetic*. Your code would be riddled with variable initialisations and `clear` commands, best way to confuse the most serious programmer (*yes you there!...*) and program accidents waiting to happen.

If no complex arithmetic is expected, the use of `i` and `j` is perfectly functional and there is no performance penalty.

将它们用作虚数单位：

如果你的代码需要处理复数，那么 i 和 j 肯定会很有用。然而，为了避免歧义，甚至为了性能，建议使用完整形式而非简写语法。完整形式是 1i （或 1j ）。

```
>> [ i ; j ; 1i ; 1j ]
ans =
   0.0000 + 1.0000i
   0.0000 + 1.0000i
   0.0000 + 1.0000i
   0.0000 + 1.0000i
```

它们表示相同的值 sqrt (- 1)，但后者形式：

- 在语义上更明确。
- 更易维护（以后查看你代码的人不必查阅代码来判断 i 或 j 是变量还是虚数单位）。
- 更快（来源：MathWorks）。

注意，完整语法 1i 在符号前可以有任意数字：

```
>> a = 3 + 7.8j
a =
   3.0000 + 7.8000i
```

这是唯一一个可以在数字之间不加运算符直接连接的函数。

注意事项

虽然将其用作虚数单位或变量是完全合法的，下面是一个小例子，说明如果两种用法混用会有多么令人困惑：

让我们重写 i，将其作为一个变量：

```
>> i=3
i =
     3
```

现在 i是一个变量（保存值为3），但我们只重写了虚数单位的简写表示，完整形式仍然被正确解释：

```
>> 3i
ans =
   0.0000 + 3.0000i
```

这现在让我们能够构建最晦涩的表达式。我让你评估以下所有结构的可读性：

```
>> [ i ; 3i ; 3*i ; i+3i ; i+3*i ]
ans =
   3.0000 + 0.0000i
   0.0000 + 3.0000i
   9.0000 + 0.0000i
   3.0000 + 3.0000i
```


Using them as imaginary unit:

If your code has to deal with `complex` numbers, then `i` and `j` will certainly come in handy. However, for the sake of disambiguation and even for performances, it is recommended to use the full form instead of the shorthand syntax. The full form is `1i` (or `1j`).

```
>> [ i ; j ; 1i ; 1j]
ans =
   0.0000 + 1.0000i
   0.0000 + 1.0000i
   0.0000 + 1.0000i
   0.0000 + 1.0000i
```

They do represent the same value `sqrt(-1)`, but the later form:

- is more explicit, in a semantic way.
- is more maintainable (someone looking at your code later will not have to read up the code to find whether `i` or `j` was a variable or the imaginary unit).
- is faster (source: MathWorks).

Note that the full syntax `1i` is valid with any number preceding the symbol:

```
>> a = 3 + 7.8j
a =
   3.0000 + 7.8000i
```

This is the only function which you can stick with a number without an operator between them.

Pitfalls

While their use as *imaginary unit* **OR** *variable* is perfectly legal, here is just a small example of how confusing it could get if both usages get mixed:

Let's override `i` and make it a variable:

```
>> i=3
i =
     3
```

Now `i` is a *variable* (holding the value `3`), but we only override the *shorthand* notation of the imaginary unit, the full form is still interpreted correctly:

```
>> 3i
ans =
   0.0000 + 3.0000i
```

Which now lets us build the most obscure formulations. I let you assess the readability of all the following constructs:

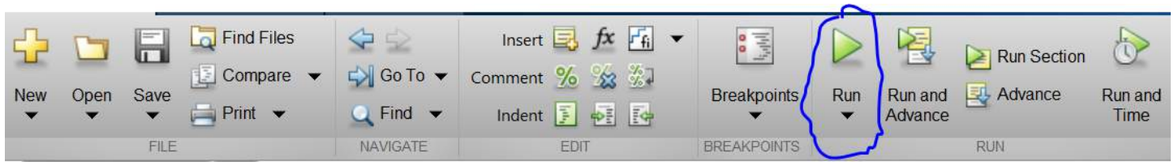
```
>> [ i ; 3i ; 3*i ; i+3i ; i+3*i ]
ans =
   3.0000 + 0.0000i
   0.0000 + 3.0000i
   9.0000 + 0.0000i
   3.0000 + 3.0000i
```

```
12.0000 + 0.0000i
```

如您所见，上述数组中的每个值都会返回不同的结果。虽然每个结果都是有效的（前提是这是最初的意图），但大多数人都会承认，阅读充满此类结构的代码将是一场真正的噩梦。

第33.7节：输入参数不足

初学MATLAB的开发者通常会使用MATLAB的编辑器来编写和编辑代码，特别是带有输入和输出的自定义函数。在MATLAB的较新版本中，顶部有一个*运行*按钮：



一旦开发者完成代码，他们通常会忍不住点击运行按钮。对于某些函数，这样做是可以正常工作的，但对于其他函数，他们会收到输入参数不足的错误，并且会对错误发生的原因感到困惑。

该错误可能不会发生的原因是您编写了一个不带输入参数的 MATLAB 脚本或函数。使用运行按钮将运行测试脚本或假设无输入参数的函数。如果您的函数需要输入参数，则会出现输入参数不足错误，因为您编写的函数期望输入参数传入函数内部。因此，您不能指望仅通过按运行按钮来运行该函数。

为了演示这个问题，假设我们有一个函数mult，它只是将两个矩阵相乘：

```
function C = mult(A, B)
   C = A * B;
end
```

在较新版本的MATLAB中，如果你编写了这个函数并点击Run按钮，它会给出我们预期的错误：

```
>> mult
输入参数不足。

错误 在 mult (第2行)
    C = A * B;
```

有两种方法可以解决这个问题：

方法一 - 通过命令提示符

只需在命令提示符中创建所需的输入，然后使用你创建的这些输入运行函数：

```
A = rand(5,5);
B = rand(5,5);
C = mult(A,B);
```

方法二 - 通过编辑器交互式操作

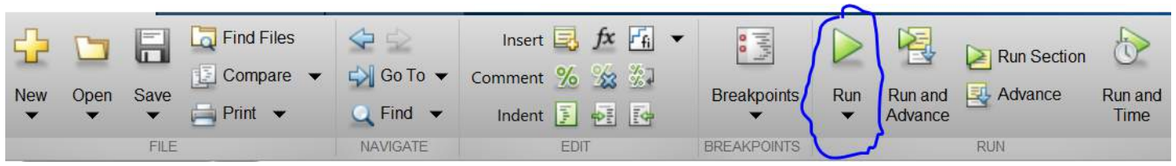
在运行按钮下方，有一个深黑色的箭头。如果点击该箭头，您可以指定变量


```
12.0000 + 0.0000i
```

As you can see, each value in the array above return a different result. While each result is valid (provided that was the initial intent), most of you will admit that it would be a proper nightmare to read a code riddled with such constructs.

Section 33.7: Not enough input arguments

Often beginning MATLAB developers will use MATLAB's editor to write and edit code, in particular custom functions with inputs and outputs. There is a *Run* button at the top that is available in recent versions of MATLAB:



Once the developer finishes with the code, they are often tempted to push the *Run* button. For some functions this will work fine, but for others they will receive a `Not enough input arguments` error and be puzzled about why the error occurs.

The reason why this error may not happen is because you wrote a MATLAB script or a function that takes in no input arguments. Using the *Run* button will run a test script or run a function assuming no input arguments. If your function requires input arguments, the `Not enough input arguments` error will occur as you have written a functions that expects inputs to go inside the function. Therefore, you cannot expect the function to run by simply pushing the *Run* button.

To demonstrate this issue, suppose we have a function `mult` that simply multiplies two matrices together:

```
function C = mult(A, B)
    C = A * B;
end
```

In recent versions of MATLAB, if you wrote this function and pushed the *Run* button, it will give you the error we expect:

```
>> mult
Not enough input arguments.

Error in mult (line 2)
    C = A * B;
```

There are two ways to resolve this issue:

Method #1 - Through the Command Prompt

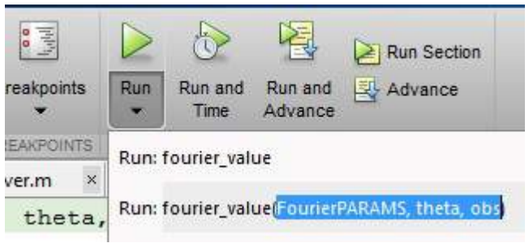
Simply create the inputs you need in the Command Prompt, then run the function using those inputs you have created:

```
A = rand(5,5);
B = rand(5,5);
C = mult(A,B);
```

Method #2 - Interactively through the Editor

Underneath the *Run* button, there is a dark black arrow. If you click on that arrow, you can specify the variables you

您希望通过输入调用函数的方式，从 MATLAB 工作区获取数据，调用方式应与方法#1中看到的完全一致。请确保您在函数中指定的变量存在于 MATLAB 工作区中：



第33.8节：多维数组中使用`length`

MATLAB 编程者常犯的一个错误是将length函数用于矩阵（而非其设计用途的向量）。如其文档中所述，length函数“返回输入的最大数组维度的长度”。

对于向量，length 的返回值有两种不同的含义：

1. 向量中元素的总数。
2. 向量的最大维度。

与向量不同，上述值对于多个非单例（即大小大于1）的维度的数组不会相等。这就是为什么对矩阵使用length存在歧义的原因。相反，即使在处理向量时，也建议使用以下函数之一，以使代码的意图完全清晰：

1. size(A) - 返回一个行向量，其元素包含对应维度上的元素数量 A 的维度。
2. numel(A) - 返回A中的元素数量。等同于prod(size(A))。
3. ndims(A) - 返回数组A的维数。等同于numel(size(A))。

这在编写“面向未来”的矢量化库函数时尤其重要，这些函数的输入事先未知，且可能具有各种大小和形状。

第33.9节：注意数组大小的变化

MATLAB中的一些常见操作，如微分或积分，输出的结果元素数量可能与输入数据不同。这个事实很容易被忽视，通常会导致诸如矩阵维度必须一致的错误。考虑以下示例：

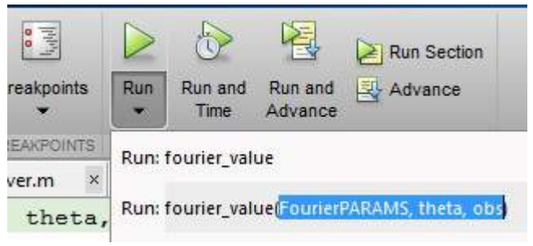
```
t = 0:0.1:10;         % 声明时间向量
y = sin(t);           % 声明函数

dy_dt = diff(y);      % 计算 y = sin(t) 的导数 dy/dt
```

假设我们想绘制这些结果。我们查看数组大小，发现：

```
size(y) 是 1x101
size(t) 是 1x101
```

但是：

would like to get from the MATLAB workspace by typing the way you want to call the function exactly as how you have seen in method #1. Be sure that the variables you are specifying inside the function exist in the MATLAB workspace:



Section 33.8: Using `length` for multidimensional arrays

A common mistake MATLAB coders have, is using the `length` function for matrices (as opposed to **vectors**, for which it is intended). The `length` function, as mentioned in its documentation, "*returns the length of the largest array dimension*" of the input.

For vectors, the return value of `length` has two different meanings:

1. The total number of elements in the vector.
2. The largest dimension of the vector.

Unlike in vectors, the above values would not be equal for arrays of more than one non-singleton (i.e. whose size is larger than 1) dimension. This is why using `length` for matrices is ambiguous. Instead, using one of the following functions is encouraged, even when working with vectors, to make the intention of the code perfectly clear:

1. `size(A)` - returns a row vector whose elements contain the amount of elements along the corresponding dimension of `A`.
2. `numel(A)` - returns the number of elements in `A`. Equivalent to `prod(size(A))`.
3. `ndims(A)` - returns the number of dimensions in the array `A`. Equivalent to `numel(size(A))`.

This is especially important when writing "future-proof", vectorized library functions, whose inputs are not known in advance, and can have various sizes and shapes.

Section 33.9: Watch out for array size changes

Some common operations in MATLAB, like **differentiation** or **integration**, output results that have a different amount of elements than the input data has. This fact can easily be overlooked, which would usually cause errors like `Matrix dimensions must agree`. Consider the following example:

```
t = 0:0.1:10;         % Declaring a time vector
y = sin(t);           % Declaring a function

dy_dt = diff(y);      % calculates dy/dt for y = sin(t)
```

Let's say we want to plot these results. We take a look at the array sizes and see:

```
size(y) is 1x101
size(t) is 1x101
```

But:

```
size(dy_dt) is 1x100
```

数组少了一个元素！

现在假设你有随时间变化的位置测量数据，想要计算jerk(t)，你将得到一个比时间数组少3个元素的数组（因为jerk是位置的三阶导数）。

```
vel = diff(y);       % 计算速度 vel=dy/dt, y = sin(t)  size(vel)=1x100
acc = diff(vel);     % 计算加速度 acc=d(vel)/dt         size(acc)=1x99
jerk = diff(acc);    % 计算jerk jerk=d(acc)/dt               size(jerk)=1x98
```

然后进行如下操作：

```
x = jerk .* t;          % 对应元素相乘 jerk 和 t
```

会返回错误，因为矩阵维度不匹配。

要计算上述操作，必须调整较大数组的大小以匹配较小数组。你也可以对数据运行回归（polyfit）来获得数据的多项式表达式。

维度不匹配错误

维度不匹配错误通常出现在：

- 未注意函数/方法调用返回变量的形状。在许多内置MATLAB函数中，矩阵会被转换为向量以加快计算速度，返回的变量可能仍是向量而非我们预期的矩阵。当涉及逻辑掩码时，这也是常见情况。

- 在调用隐式数组扩展时使用不兼容的数组大小。


```
size(dy_dt) is 1x100
```

The array is one element shorter!

Now imagine you have measurement data of positions over time and want to calculate *jerk(t)*, you will get an array 3 elements less than the time array (because the jerk is the position differentiated 3 times).

```
vel = diff(y);       % calculates velocity vel=dy/dt for y = sin(t)  size(vel)=1x100
acc = diff(vel);     % calculates acceleration acc=d(vel)/dt         size(acc)=1x99
jerk = diff(acc);    % calculates jerk jerk=d(acc)/dt                size(jerk)=1x98
```

And then operations like:

```
x = jerk .* t;          % multiplies jerk and t element wise
```

return errors, because the matrix dimensions do not agree.

To calculate operations like above you have to adjust the bigger array size to fit the smaller one. You could also run a regression (`polyfit`) with your data to get a polynomial for your data.

Dimension Mismatch Errors

Dimension mismatch errors typically appear when:

- Not paying attention to the shape of returned variables from function/method calls. In many inbuilt MATLAB functions, matrices are converted into vectors to speed up the calculations, and the returned variable might still be a vector rather than the matrix we expected. This is also a common scenario when logical masking is involved.
- Using incompatible array sizes while invoking implicit array expansion.

致谢

非常感谢所有来自Stack Overflow文档的人员提供此内容，
更多更改可发送至web@petercv.com以发布或更新新内容

adjpayot	第1章
阿德里安	第27章
agent_C.Hdj	第8章
亚历山大·科罗文	第10章
alexforrence	第6章和第9章
阿姆罗	第1章、第12章和第17章
安德·比古里	第6章、第15章、第24章、第28章和第33章
安扬武	第3章
巴茨	第4章
开普科德	第15章
ceiltechbladhm	第6章
塞尔多尔	第9、12和32章
chrisb2244	第1章
克里斯托弗·克罗伊茨格	第1章
codeaviator	第29章
daleonpz	第26章
丹	第1章和第10章
达仁山	第9章和第27章
德夫	第6、9、10、23、25、26、29、31和33章
德哈根	第8章
DVarga	第1章和第25章
EBH	第1、3、8、10和32章
edwinksl	第33章
埃里克	第6章
埃里克	第1、29和32章
excaza	第1章
flawr	第1章
弗兰克·德尔农库尔	第17和27章
fyrepenguin	第1章和第32章
GameOfThrows	第1章和第18章
girish_m	第15章
Hardik_Jain	第27章
Hoki	第28章、第31章和第33章
il_raffa	第16章、第26章和第30章
itzik Ben Shabat	第13章
jenszvs	第25章
吉姆	第27章
jkazan	第22章
贾斯汀	第25章
肯·塞贝斯塔	第28章
兰达克	第1章、第7章、第18章和第33章
利奥尔	第1章
马利克	第30、32和33章
matlabgui	第29章
马特	第1、10、12和33章
MayeulC	第30章
麦克莱蒙	第30章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

adjpayot	Chapter 1
Adriaan	Chapter 27
agent_C.Hdj	Chapter 8
Alexander Korovin	Chapter 10
alexforrence	Chapters 6 and 9
Amro	Chapters 1, 12 and 17
Ander Biguri	Chapters 6, 15, 24, 28 and 33
anyanwu	Chapter 3
Batsu	Chapter 4
Cape Code	Chapter 15
ceiltechbladhm	Chapter 6
Celdor	Chapters 9, 12 and 32
chrisb2244	Chapter 1
Christopher Creutzig	Chapter 1
codeaviator	Chapter 29
daleonpz	Chapter 26
Dan	Chapters 1 and 10
daren shan	Chapters 9 and 27
Dev	Chapters 6, 9, 10, 23, 25, 26, 29, 31 and 33
drhagen	Chapter 8
DVarga	Chapters 1 and 25
EBH	Chapters 1, 3, 8, 10 and 32
edwinksl	Chapter 33
Eric	Chapter 6
Erik	Chapters 1, 29 and 32
excaza	Chapter 1
flawr	Chapter 1
Franck Dernoncourt	Chapters 17 and 27
fyrepenguin	Chapters 1 and 32
GameOfThrows	Chapters 1 and 18
girish_m	Chapter 15
Hardik_Jain	Chapter 27
Hoki	Chapters 28, 31 and 33
il_raffa	Chapters 16, 26 and 30
itzik Ben Shabat	Chapter 13
jenszvs	Chapter 25
Jim	Chapter 27
jkazan	Chapter 22
Justin	Chapter 25
Kenn Sebesta	Chapter 28
Landak	Chapters 1, 7, 18 and 33
Lior	Chapter 1
Malick	Chapters 30, 32 and 33
matlabgui	Chapter 29
Matt	Chapters 1, 10, 12 and 33
MayeulC	Chapter 30
McLemon	Chapter 30

mhopeng	第28章
mike	第24章
Mikhail_Sam	第1章和第7章
莫森·诺斯拉蒂尼亚	第7章和第9章
nahomyaja	第33章
nitsua60	第16章
NKN	第16、30和33章
诺亚·雷格夫	第14章
奥列格	第10、12、17、28和33章
pseudoDust	第26章
R. Joiny	第33章
rajah9	第2章
rayryeng	第33章
罗伊	第23章
S. 拉德夫	第7章
萨姆·罗伯茨	第1章
萨达尔·乌萨马	第30章
沙伊	第1、5、10和15章
斯特凡M	第5、11、19、20和21章
苏弗	第31章和第33章
thewaywewalk	第16章、第26章和第29章
蒂姆	第33章
Trilarion	第15章
Trogdor	第9章
泰勒	第1章、第8章和第10章
乌玛尔	第33章
泽普	第16章和第31章

mhopeng	Chapter 28
mike	Chapter 24
Mikhail_Sam	Chapters 1 and 7
Mohsen Nosratinia	Chapters 7 and 9
nahomyaja	Chapter 33
nitsua60	Chapter 16
NKN	Chapters 16, 30 and 33
Noa Regev	Chapter 14
Oleg	Chapters 10, 12, 17, 28 and 33
pseudoDust	Chapter 26
R. Joiny	Chapter 33
rajah9	Chapter 2
rayryeng	Chapter 33
Royi	Chapter 23
S. Radev	Chapter 7
Sam Roberts	Chapter 1
Sardar Usama	Chapter 30
Shai	Chapters 1, 5, 10 and 15
StefanM	Chapters 5, 11, 19, 20 and 21
Suever	Chapters 31 and 33
thewaywewalk	Chapters 16, 26 and 29
Tim	Chapter 33
Trilarion	Chapter 15
Trogdor	Chapter 9
Tyler	Chapters 1, 8 and 10
Umar	Chapter 33
Zep	Chapters 16 and 31

你可能也喜欢

You may also like