《Syslab计算方法教程》——基础版

第一章 预备知识

1.1 初识Julia

Julia程序语言是一种计算机编程语言， 就像C、C++、Fortran、Java、R、Python、Matlab等程序语言一样。因此，我们可以使用符合 Julia 语言规范的代码来编写程序。这些程序可以用于纯粹的数学和科学计算、存取本地文件、通过网络收发数据，等等。

Julia语言历史比较短，发布于2012年， 是MIT的几位作者（Jeff Bezanson, Stefan Karpinski, Viral Shah, Alan Edelman）和全世界的参与者共同制作的。 Julia与R、Python等一样是动态类型语言， 程序与R、Python一样简单， 但是它先进的设计使得Julia程序的效率基本达到和C、Fortran等强类型语言同样的高效率。 尤其适用于数值计算，在现今的大数据应用中也是特别合适的语言， 排在Python、R之后，已经获得广泛的关注。

Julia语言的目标是成为一门易用、强大和高效的编程语言。类似 Python、R 的脚本语言是面向解释器的，用户写完代码，可以直接解释执行，并返回结果；而类似 C、Fortran 等语言必须先编译成机器语言才能执行。后者速度更快但使用起来没有前者方便，因此想要快速实现复杂的程序逻辑时，开发者可能会选择用脚本语言写算法，之后再翻译到 C、C++ 等语言，这是一个花精力又容易出错的过程 。

Julia 的诞生就是为了将以上不同语言的性能结合成一门语言的特性，Julia 的编程过程是交互式的、动态的，同时也拥有静态编译语言的性能 。官网介绍的 Julia 特性如下：

- **快速：** 接近编译型语言的性能，2017 年甚至加入了 Petaflop 俱乐部（峰值超过每秒 1 petaflop 的语言，另几个语言分别是 C、C++ 和 Fortran），Julia 开发的应用 Celeste 在 NERSC 的超级电脑上使用 1300 万线程，达到了1.54 petaflops 的峰值。
- **动态：** 是动态类型系统，并很好地支持了交互编程，易于开发者学习使用。
- **环境可复用：** 支持通过 BinaryBuilder 在不同平台上快速重现开发环境。
- **可组合性：** 提供了基于多重派发和类型系统的编程范式，可称为面向方法的（method-oriented）。
- **通用：** 提供了文件读写、元编程、调试、日志记录、性能分析等标准库，还提供了一个包管理工具，可用于实现完整的应用和微服务。
- **开源：** 代码托管在 GitHub，采用 MIT 许可证，当前已有超过 1000 的贡献者。

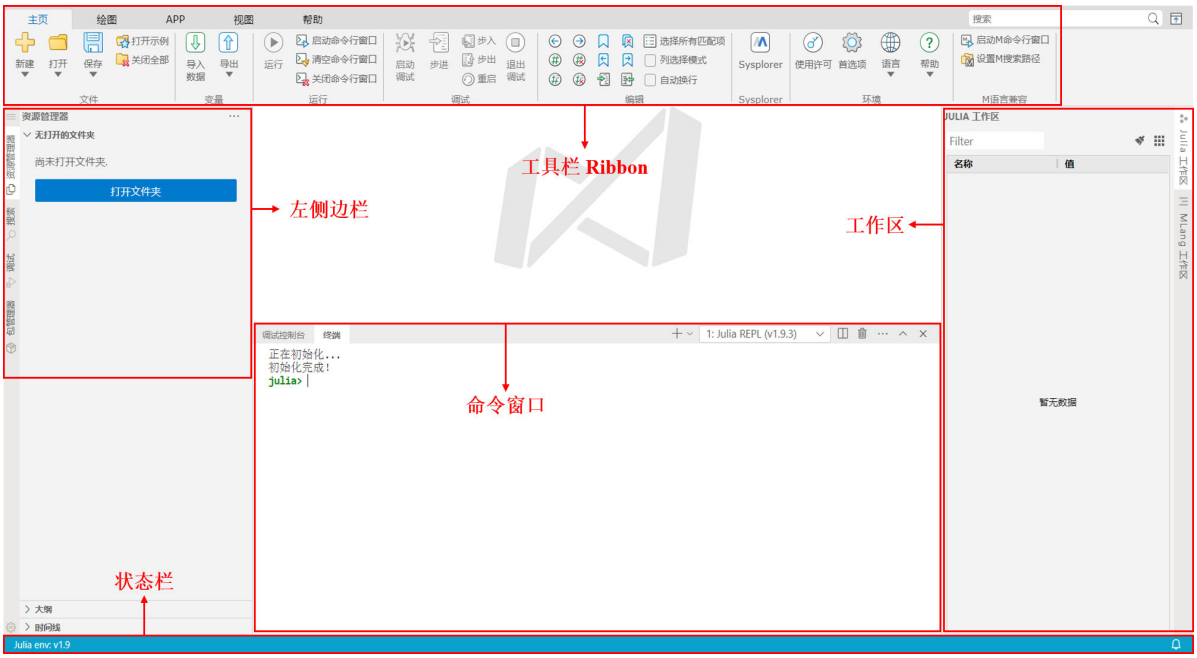
Julia 的速度一定程度上是语言本身的设计带来的，可以根据需求引入静态类型，牺牲部分不那么重要的动态性换来性能的大幅提升。可以说 Julia 是在尝试寻找两种编程语言性能和动态的平衡点。

1.2 Syslab介绍

在现代科学和工程技术中，经常会遇到大量复杂的数学计算问题，这些问题用一般的计算工具来解决非常困难，而用计算机来处理却非常容易。于是，同元软控推出了科学计算环境MWORKS.Syslab，基于高性能科学计算语言Julia提供交互式编程环境的完备功能。MWORKS.Syslab能有效应对科学与工程中遇到的矩阵运算、微分方程求解、数据分析、信号处理、控制设计与优化等问题。MWORKS.Syslab也是一个支持脚本开发和调试的环境，通过脚本驱动系统建模仿真环境，实现科学计算与系统建模仿真过程的自动化运行。

1.2.1 Syslab基本操作

Syslab的界面布局主要由五大块构成，如图1.2.1所示：



[图1.2.1] Syslab界面布局

- 工具栏Ribbon：用于提供平台快捷操作按钮。
- 左侧边栏：用于提供不同的功能部件，点击可以展开功能面板。
- 命令窗口：用于终端交互，可以输入脚本命令，并回显执行结果。
- 工作区：用于提供全局变量列表的显示与管理。
- 状态栏：用于提示状态信息。

Syslab编辑代码的方式有两种：

（1）直接在命令窗口中输入命令

在Syslab的命令窗口中直接输入命令，即可得到结果。

[例1.2.1] 计算 3+5

```
julia> x=3+5     #在Syslab的命令窗口中输入x=3+5，敲击回车；
8                #在命令窗口中将显示：
```

```
julia> x=3+5;    #此命令行以分号结尾，敲击回车后，命令窗口不会直接显示计算结果。

julia> x         #再输入x，并敲击回车，即可显示计算结果
8
```

[例1.2.2] 在命令窗口计算函数 $f(x_1,x_2)=2{x_1}^2+7{x_2}^2-3x_1x_2-5x_1+3x_2-12$

在Syslab的命令窗口中，编辑并输入函数，调用函数，输出相应的函数值。

在命令窗口输入如下代码：

```
julia> function f(x1,x2)
           return 2*x1^2+7*x2^2-3*x1*x2-5*x1+3*x2-12;
       end  #输入完成，敲击回车。
f (generic function with 1 method) #命令窗口会提示
```

接着，在命令窗口中调用函数：

```
julia> f(2,3)   #命令窗口中输入并回车
40              #显示
julia> f(3,-5)  #命令窗口中输入并回车
196             #显示
```

（2）新建脚本文件编辑代码

点击Syslab中新建脚本，在代码编辑器中编辑命令，保存为.jl文件，运行成功后，即可在命令窗口执行该文件，得到所需的结果，具体步骤流程如下：

1. 打开文件夹。点击侧边栏上的资源管理器按钮，资源管理器面板展开，当工作区不存在内容时，可以点击 [打开文件夹] 按钮，弹出文件选择对话框，选择文件夹并确认添加后，即可在资源管理器中打开文件夹。
2. 新建文件。工作区目录树的顶层是一个标题栏，把鼠标悬停在文件目录树上，点击 [新建脚本]，会在选中目录位置处新建一个Julia类型的Unnamed文件，并进入编辑状态。
3. 编写代码，保存为后缀为.jl文件。
4. 运行文件。Julia代码编辑完成后，点击工具栏Ribbon中的[运行]，系统将启动REPL并执行该文件脚本。

[例1.2.3] 使用脚本文件计算函数 $f(x_1,x_2)=2{x_1}^2+7{x_2}^2-3x_1x_2-5x_1+3x_2-12$

1. 新建脚本文件，在代码编辑器中输入该函数的julia代码

```
#  "#"号用于注释，这里注意
function f(x1,x2)
        #第一种输出方式，默认输出y
        y=2*x1^2+7*x2^2-3*x1*x2-5*x1+3*x2-12
end
```

```
function f(x1,x2)
        #第二种输出方式，指定输出的数据
         y=2*x1^2+7*x2^2-3*x1*x2-5*x1+3*x2-12
         return y
end
```

输入完毕后保存在工作目录里面。

2. 在命令窗口中调用该函数

```
julia> f(2,3)   #在窗口中输入
40              #输出
julia> f(3,-5)  #在窗口中再次输入
196             #输出
```

1.2.2 编写第一个程序

现在就在Syslab中编写我们的第一个 Julia 程序。

[例1.2.4] 计算一组数据xi(i=1,2,...,n)的S1和S2

其中

$$S_1 = \frac{\sum_{i=1}^{n} x_i}{n} ; S_2 = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}$$

1. 在代码编辑器中输入该函数的文件

```
# 输入一个数组x，返回两个值，分别是S1和S2
function demofunc(x)
        n=length(x) #返回数组x的长度
        s1=sum(x)/n #sum函数对输入的数组x求和
        s1 = round(s1,digits=2)
        #四舍五入，可以调用round()函数，指定里面小数点后面的位数2位
        s2=sqrt(sum(x.^2)/n)
        s2 = round(s2,digits=2)
        return (s1,s2)
end
```

2. 在命令窗口调用函数

```
julia> x=[1,2,3,4,5];              #在命令行输入数组x
julia> (s1,s2) = demofunc(x)       #在命令行调用函数
(3.0, 3.32)                        #命令行显示结果
```

第二章 Julia语言基础

2.1 基本数据与运算

2.1.1 整数与浮点数

Syslab中整数值可以直接写成如123或者-123这样。虽然整数有多种类型，一般程序中不必特别关心整数常量的具体类型。Syslab允许使用特别长的整数，这时其类型为BigInt。Syslab的浮点数可以写成带点的形式如123.0, 1.23, 也可以写成带有10的幂次如1.23e3(表示1.23×10^{3}), 1.23e-3(表示1.23×10^{-3})。这些写法都属于Float64类型的浮点数。Julia还有其他类型的浮点数，但是科学计算中主要使用Float64类型，在别的语言中这称为双精度浮点数。

2.1.2 字符串

单个字符在两边用英文单撇号界定，如`'A'`，`'算'`。 字符都是用Unicode编码存储，具体使用UTF-8编码。 每个字符可能使用1到4个字节表示。 字符的类型为Char， 自定义函数中的字符参数可声明为AbstractChar。

零到多个字符组成字符串， 程序中的字符串在两边用双撇号界定，如`"A cat"`，`"华中大"`。 字符串数据类型名称为String， 自定义函数中的字符串参数可声明为AbstractString。

对于占据多行的字符串， 可以在两侧分别用三个双撇号界定。如

```
"""
这是第一行
这是第二行
三个双撇号界定的字符串中间的单个双撇号"不需要转义
"""
```

注意多行内容的首行不需要紧挨着写在开头的三个双撇号后面同一行内。

字符串属于不可修改类型(immutable)， 即不能直接修改字符串的内容， 但可以给保存了字符串的变量赋值为一个新的字符串。

用星号“`*`”连接两个字符串，也可以将字符连接成字符串，如

```
julia> '#' * "这是" * "美好的一天" * "。"
"#这是美好的一天。"
```

2.1.3 算术运算

表示加、减、乘、除、乘方的运算符分别为：

加	减	乘	除	乘方
+	-	*	/	^

（1）浮点数的四则运算

遵循传统的算数运算规则和优先级规定。 用圆括号改变优先级，如

```
julia> (1.3 + 2.5)*2.0 - 3.6/1.2 + 1.2^2
6.039999999999999
```

表示$(1.3+2.5)*2.0-3.6/1.2+1.2^2$，注意浮点运算会造成数值计算误差。

(2) 整数的四则运算

整数加法、减法、乘法结果仍为整数， 如

```
julia> 10 + 2*3 - 3*4
4
```

整数用"/"作的除法总是返回浮点数，即使结果是整数也是一样，如

```
julia> 10/2
5.0
```

整数用`a % b`表示a整除b的余数，结果符号总是取a的符号。如

```
julia> 10 % 3
1
```

整数与浮点数的混合运算会将整数转换成浮点数再计算。

2.1.4 比较和逻辑运算

(1) 比较运算

两个数值之间用如下的比较运算符进行比较：

大于	小于	大于等于	小于等于	等于	不等于
>	<	>=	<=	==	!=

要特别注意"等于"比较用两个等号表示。比较的结果是true(真值)或者false(假值)。 结果类型为**布尔型**（Bool）。

```
julia> 1 == 1.0
true
julia> 2 != 2.0001
true
julia> 3.5 > -1
true
julia> 3.5 < -1.5
false
julia> -3.5 >= 1.2
false
julia> -3.5 <= 1.2
true
```

两个字符串之间也可以比较， 比较时从左至右依次按字典序比较每个字符， 两个字符的次序按照其Unicode编码值比较，如：

```
julia> "abc" == "ABC"
false
julia> "ab" < "abc"
true
julia> "陕西省" == "山西省"
false
```

(2) 逻辑运算

比较通常有变量参与，如：

```
julia> age = 35; sex="F";
julia> age < 18
false
julia> sex == "F"
true
```

有时需要构造复合的条件， 如“年龄不足18岁且性别为女”， “年龄在18岁以上或者性别为男”等。

用`&&`表示要求两个条件同时成立， 用`||`表示只要两个条件之一成立则结果为真， 用`!cond`表示`cond`的反面，如：

```
julia> age = 35; sex="F";
julia> age < 18 && sex == "F"
false
julia> age >= 18 || sex == "M"
true
```

2.1.5 变量

变量名是一个标识符， 用来指向某个值在计算机内存中的存储位置。 变量名可以用英文大小写字母、下划线、数字、允许的Unicode字符， 区分大小写。 变量名不允许使用空格、句点以及其它标点符号和井号之类的特殊字符。

为兼容性起见，尽可能不要用汉字作为变量名。 变量名主要使用小写字母、数字和下划线构成， 两个英文单词之间可以用下划线连接， 如`total_number`， 也可以在单词之间使用开头字母大写来区分单词， 如`totalNumber`， 第一个单词仍全部用小写。

给变量**赋值**，即将变量名与一个内存中的内容联系起来，也称为绑定（binding）， 使用等号“=”，等号左边写变量名，右边写要保存到变量中的值，如：

```
julia> x = 123
123
julia> y = 1+3/2
2.5
julia> addr10086 = "北京市海淀区颐和园路5号"
"北京市海淀区颐和园路5号"
```

变量的类型是由它保存的（指向的内存中的）值的类型决定的， 不需要说明变量类型（Julia允许说明变量类型，但一般不需要）。

变量赋值后，就可以参与运算，如：

```
julia> x = 123;
julia> y = 1+3/2;
julia> x + y*2
128.0
```

变量名前面紧挨着数字表示相乘，如

```
julia> x + 2y
128.0
```

赋值还有一种计算修改简写方式， 即将变量值进行四则运算后保存回原变量， 格式为`x op= expr`， 其中`op`是某种四则运算， 这等价于`x = x op expr`， 如：

```
julia> x = 123;
julia> x += 100;
julia> x
223
```

2.2 向量

向量运算中通常会涉及到多种函数，其中数学函数为分析数据、开发算法和创建模型提供了一系列数值计算方法。核心函数使用经过处理器优化的库，可以快速进行向量和矩阵计算。在使用相关函数前需先加载数学函数库 TyMath (using TyMath)。

Julia的向量实际是一维数组。 在程序中直接定义一个向量，只要用方括号内写多个逗号分隔的数值，如在命令行输入：

```
julia> v1 = [1, 3, 4, 9, 13]
5-element Vector{Int64}:
  1
  3
  4
  9
 13
#v1是整数型的向量
```

```
julia> v2 = [1.5, 3, 4, 9.12]
4-element Vector{Float64}:
 1.5
 3.0
 4.0
 9.12
#v2是浮点型Float64的向量
```

用`length(x)`求向量`x`的元素个数，如

```
julia> length(v1)
5
```

可以用`1:5`定义一个范围， 在仅使用其中的元素值而不改写时作用与`[1, 2, 3, 4, 5]`类似。`1:2:9`定义带有步长的范围，表示的值与`[1, 3, 5, 7, 9]`类似。 范围只需要存储必要的开始、结束、步长信息， 所以更节省空间， 但是不能对其元素进行修改。

```
julia> 1:5
1:5          #1:5是一个范围
```

范围不是向量，用`collect()`函数可以将范围转换成向量，如：

```
julia> collect(1:5)
5-element Vector{Int64}:
 1
 2
 3
 4
 5
```

2.2.1 向量下标索引

若`x`是向量，`i`是正整数，`x[i]`表示向量的第`i`个元素。 第一个元素的下标为1，这种规定与R、FORTRAN语言相同， 但不同于Python、C、C++、JAVA语言。 如

```
julia> v1[2]    # v1 = [1, 3, 4, 9, 13]见2.2节
3
```

用`end`表示最后一个元素位置，如

```
julia> v1[end]
13
```

对元素赋值将在原地修改元素的值，如

```
julia> v1[2] = -999;
julia> v1
5-element Vector{Int64}:
    1
 -999
    4
    9
   13
```

（1）用范围作为下标

下标可以是一个范围，如

```
julia> v1[2:4]
3-element Vector{Int64}:
 -999
    4
    9
```

在这种范围中，用`end`表示最后一个下标，如

```
julia> v1[4:end]
2-element Vector{Int64}:
  9
 13
```

(2) 数组作为下标

向量的下标也可以是一个下标数组，如

```
julia> v1[[1, 3, 5]]
3-element Vector{Int64}:
  1
  4
 13
```

取出的多个元素可以修改，可以用`.=`运算符赋值为同一个标量，如：

```
julia> v1[1:3] .= 0;
julia> v1
5-element Vector{Int64}:
  0
  0
  0
  9
 13
```

也可以分别赋值，如

```
julia> v1[[1, 3, 5]] = [101, 303, 505];
julia> v1
5-element Vector{Int64}:
 101
   0
 303
   9
 505
```

2.2.2 向量与标量的运算

向量与一个标量作四则运算， 将运算符前面加句点“.”：

```
.+   .-   .*   ./   .^
```

表示向量的每个元素分别与该标量作四则运算， 结果仍是向量。如

```
julia> v1 = [1, 3, 4, 9, 13];
julia> v1 .+ 100
5-element Vector{Int64}:
 101
 103
 104
 109
 113
```

2.2.3 向量与向量的四则运算

两个等长的向量之间作加点的四则运算，表示对应元素作相应的运算。如

```
julia> v1 = [1, 3, 4, 9, 13];
julia> v3 = [2, 5, 6, 7, 10];
julia> v1 .+ v3
5-element Vector{Int64}:
  3
  8
 10
 16
 23
```

2.2.4 向量的比较运算

两个标量之间可以进行如下的比较运算：

```
==  !==  <  <=  >  >=
```

向量每个元素与标量之间、两个向量对应元素之间比较， 只要在前面的比较运算符前面增加句点：

```
.==  .!=  .<  .<=  .>  .>=
```

标量比较结果之间可以用 `&&` 表示“同时成立”，`||` 表示“至少其中之一成立”。 布尔型标量与向量、向量之间可以用 `.&` 表示元素间“与”， `.|` 表示元素间“或”。

2.2.5 向量初始化

用 `zeros(n)` 可以生成元素类型默认为Float64、元素值为0、长度为n的向量，如

```
julia> zeros(3)
3-element Vector{Float64}:
 0.0
 0.0
 0.0
```

用 `zeros(Int64, 3)` 可以生成指定类型的（这里是Int64）初始化向量，如

```
julia> zeros(Int64, 3)
3-element Vector{Int64}:
 0
 0
 0
```

2.2.6 向量的循环遍历

可以用 `for` 循环和 `eachindex()` 对向量的每个元素下标遍历访问。 使用 `in` 关键字，格式如

```
julia> for i in eachindex(v1)
          println("v1[", i, "] = ", v1[i])
       end
#显示
v1[1] = 1
v1[2] = 3
v1[3] = 4
v1[4] = 9
v1[5] = 13
```

这里`i`是循环产生的向量下标。也可以直接写下标范围，使用“=”或者`in`，如

```
julia> for i = 1:length(v1)
          println("v1[", i, "] = ", v1[i])
       end
v1[1] = 1
v1[2] = 3
v1[3] = 4
v1[4] = 9
v1[5] = 13
```

也可以不利用下标而是直接对元素遍历，如

```
julia> for xi in v1
    println(xi)
end
1
3
4
9
13
```

2.2.7 向量的输出

在命令窗口中，为了显示某个向量，可以用`show()`函数，比如，设`v2 = [1.5, 3, 4, 9.12]`：

```
julia> v2 = [1.5, 3, 4, 9.12];
julia> show(v2)
[1.5, 3.0, 4.0, 9.12]
```

为了将向量v2按文本格式保存到文件“tmp1.txt”中，可用：

```
julia> using DelimitedFiles
julia> writedlm("tmp1.txt", v2, ' ')
```

结果文件中每个数占一行。

2.2.8 向量的输入

假设文件“vecstore.txt”中包含如下的内容：

```
1.2 -5  3.6
7.8 9.12 4.11
```

可以用如下代码将文件中的数据读入到一个向量v4中：

```
julia> using DelimitedFiles
julia> v4 = readdlm("vecstore.txt")[:];
```

2.2.9 向量类型的变量

由于Julia的变量仅仅是向实际存储空间的绑定， 所以两个变量可以绑定到同一个向量的存储空间，修改了其中一个变量的元素值，则另一个变量的元素也被修改了。 如

```
julia> x1 = [1,2,3];
julia> x2 = x1;
julia> x2[2] = 100;
julia> x1
3-element Vector{Int64}:
   1
 100
   3
```

用`===`可以比较两个变量是否同一对象， 如：

```
julia> x2 === x1
true
```

允许两个变量指向同一个对象是有用的， 尤其在函数自变量传递时， 但是在一般程序中这种作法容易引起混淆。 向量（或者数组）作为函数自变量时， 调用函数时传递的是引用， 在函数内可以修改传递进来的向量的元素值。

2.2.10 向量的有关函数

若`x`是向量， `sum(x)`求各个元素的和， `prod(x)`求各个元素的乘积。

```
julia> x = [1,2,3];
julia> sum(x)
6
julia> prod(x)
6
```

为了判断元素`x`是否属于数组`v`，可以用表达式`x in v`判断， 结果为布尔值。

```
julia> a = 2;
julia> a in x
true
```

对于向量 V = [1, 3, 6, -7]，其（正负）无穷范数、1-范数、2-范数计算方法如下：

```
julia> p=norm(V,Inf)
7.0
julia> p=norm(V,-Inf)
1.0
julia> p=norm(V,1)
17.0
julia> p=norm(V,2)
9.7468
```

2.2.11 向量化函数

许多现代的数据分析语言， 如Python, Matlab, R等都存在循环的效率比编译代码低一两个数量级的问题， 在这些语言中， 如果将对向量和矩阵元素的操作向量化， 即以向量和矩阵整体来执行计算， 就可以利用语言内建的向量化计算获得与编译代码相近的执行效率。

Julia语言依靠其LLVM动态编译功能， 对向量和矩阵元素循环时不损失效率， 用显式循环处理向量、矩阵与向量化做法效率相近， 有时显式循环效率更高。 但是，向量化计算的程序代码更简洁， 比如上面的两个向量之间的四则运算的加点格式。Julia中的函数， 包括自定义函数， 如果可以对单个标量执行， 将函数名加后缀句点后， 就可以变成向量化版本， 对向量和矩阵执行。 如

```
julia> sqrt.([1,2,3])
3-element Vector{Float64}:
 1.0
 1.4142135623730951
 1.7320508075688772
```

2.3 矩阵

前面讲的向量当元素类型相同时可以看作一维数组，不区分行向量还是列向量， 在参与矩阵运算时看作列向量。**矩阵是二维数组，有两个下标：行下标和列下标。**在矩阵运算中同样会涉及到多种函数，在使用相关函数前需先加载数学函数库 TyMath (using TyMath)。

数组(Array)是Julia中的数据类型， 有一维、二维、多维等， 区别在于引用一个元素时所用下标个数，数组中的元素属于相同的基本类型， 比如， 元素类型都是Int64， 都是Float64， 都是String， 等等。为了在程序中直接输入一个矩阵， 可以在方括号内两个同行的元素之间用空格分隔， 两行之间用分号分隔，如

```
julia> A = [1 2 3; 4 5 6]
2×3 Matrix{Int64}:
 1  2  3
 4  5  6
```

注意结果显示这是一个`2×3 Matrix{Int64}`， 即一个两行三列的元素都是Int64类型的矩阵。

2.3.1 矩阵的输入

（1）在命令窗口内直接输入

对于$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}$，可采用如下输入：

```
julia> a = [1 2 3;4 5 6;7 8 9]
3×3 Matrix{Int64}:
 1  2  3
 4  5  6
 7  8  9
```

(2) 创建Julia文件进行输入

对于大型矩阵直接输入不方便，可以采用创建文件的方式，即在编辑窗口中输入该矩阵，然后保存在当前目录，每次进入syslab时该矩阵已自动调入内容。下面假设a是一个大规模矩阵，可以创建一个文件保存该矩阵。

```
# 相对路径：demo\bigm.jl
a = [1 2 3;4 5 6; 1 2 4];
```

```
# demo\test.jl
# 调用方法就是include("文件名")
include("bigm.jl")
a = a*a;
a
```

(3) 生成特殊矩阵的命令

1. zeros 默认生成浮点数0阵 需要指定行数与列数

   具体用法：zeros(m,n)。例如，a=zeros(3,3) 生成 3×3 的 0.0 阵

```
julia> zeros(3,3)
3×3 Matrix{Float64}:
 0.0  0.0  0.0
 0.0  0.0  0.0
 0.0  0.0  0.0
```

如果需要生成全整数0的矩阵，需要在参数里面指定数据类型！如果只输入zeros(m)一个参数，默认是一个长度为m行的列向量。

2. ones 生成全是1的矩阵，格式同 zeros。
3. eye 生成单位阵，格式同 zeros。

2.3.2 矩阵的运算

(1) 矩阵的加减运算

```
julia> c=a+b;
julia> c=a-b;
julia> c=a.+3;
```

矩阵亦可与数量运算，需要注意的是运算符前面加点，表示遍历整个矩阵元素，进行该数量运算。

(2) 矩阵的乘法

```
julia> c=a*b;
julia> c=3*a;
```

(3) 矩阵的逆

```
julia> c=inv(a);
```

(4) 矩阵的除法

```
 julia> c=a\b;
 julia> c=a/b;
```

(5) 行列式的值

```
julia> n =det(a);
```

(6) 矩阵的转置

```
julia> c=a';
```

2.3.3 矩阵的范数

[例2.3.1]对于$A=\begin{bmatrix}3 & -7 & 5\\ 6 & 8 & 4\\ 9 & 7 & 3\end{bmatrix}$

(1) 无穷范数

定义：$\|A\|_\infty=\max\limits_{1\le i\le n}\sum_{j=1}^{n}|a_{ij}|$即矩阵元素的绝对值按行相加的最大值

```
julia> p=opnorm(A,Inf)
19.0
```

(2) 1-范数

定义：$\|A\|_1=\max\limits_{1\le j\le n}\sum_{i=1}^{n}|a_{ij}|$即矩阵元素的绝对值按列相加的最大值

```
julia> p=opnorm(A,1)
22.0
```

(3) 2-范数

定义：$\|A\|_2=\sqrt{\lambda_{max}(A^TA)}$，$\lambda_{max}(A^TA)$ 表示为A^TA的最大特征值

```
 julia> p=opnorm(A,2);
 15.8687
```

2.3.4 矩阵的条件数

定义：A 为非奇异矩阵，$\mathrm{cond(A)}_r=||A^{-1}||_r||A||_r$为矩阵A的条件数。其中，r=∞，1，2分别称为无穷范数条件数、1-范数条件数、2-范数条件数。

对于矩阵$A=\begin{bmatrix}3 & 2 & 4\\ 2 & 4 & 1\\ 3 & 1 & 7\end{bmatrix}$

(1) 无穷范数条件数

```
julia> p=cond(A,Inf)
29.5263
```

(2) 1-范数条件数

```
julia> p=cond(A,1)
30.3158
```

(3) 2-范数条件数

```
julia> p=cond(A,2)
19.0575
```

2.3.5 矩阵的特征值和特征向量

定义：设 A 是 n×n 的矩阵，满足关系式$AX_i = \lambda_i X_i$，则$\lambda_i (i = 1, 2, \ldots, n)$为$\lambda_i$对应的特征向量。

对于矩阵$A = \begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix}$

(1) 矩阵的特征值

```
julia> d=eigvals(A)
1.0
4.0 （即λ1 =1.0、λ2 =4.0）
```

(2) 矩阵的特征向量

```
julia> v=eigvecs(A)
-0.707107  0.894427
 0.707107  0.447214
```

2.4 基本语句

2.4.1 输出语句

在Syslab命令行，键入变量名或者计算表达式直接在下面显示结果。 可以用`println()`函数显示指定的变量和结果。

```
julia> x = 1; y = 2; println(x + y*2)
5
```

在`println()`中多个要输出的项用逗号分开。 两项之间没有默认的分隔， 如果需要分隔可以自己写在输出项中。

```
julia> x = 1; y = 2;println("x=", x, " y=", y, " x + y*2 =", x+y*2)
x=1 y=2 x + y*2 =5
```

`println()`函数输出会将后续输出设置到下一行， 而`print()`函数与`println()`类似但是将后续输出设置在当前行。

```
#在脚本文件中运行：
print("Hello, world!")
println("Hello, IncludeHelp")
print(10+20)

Hello, world!Hello, IncludeHelp
30
```

```
#在命令窗口中运行：
julia> print("Hello, world!"); println("Hello, IncludeHelp"); print(10+20)
Hello, world!Hello, IncludeHelp
30
```

字符串中可以用`$变量名`或`$(表达式)`的格式插入变量或表达式的值。例如：

```
julia> name="John"; "My name is $name"
"My name is John"
```

再举个例子：

```
julia> "100$(name)999"
"100John999"
```

如果想要输出圆周率pai小数点后面两位，应该如何操作呢？

在Syslab里面提供了控制输出格式的printf函数，类似于C语言的printf，具体用法如下：

```
julia> using Printf      #加载Printf库
julia> @printf("pi = %0.2f", float(pi))
3.14                     #控制小数的输出精度为小数点后面两位
```

2.4.2 注释语句

Syslab中的注释有单行注释和多行注释。程序执行时会自动忽略注释行。

Syslab中单行注释以 # 开头，例如：

```
# 这是一行注释
# 这是另外一行注释
println("Hello World!")
```

多行注释用 #= 与 =# 将注释括起来，例如:

```
#=
1、这是一行注释
2、这是另外一行注释
=#
println("Hello World!")
```

2.4.3 if 语句

当条件表达式为 true 时执行语句，一般格式为：

```
if 条件表达式1
   #执行语句
elseif 条件表达式2
       #执行语句
elseif 条件表达式3
       #执行语句
else
   #执行语句
end
```

`if expression, statements, end` 计算表达式并在表达式为 true 时执行一组语句。表达式的结果非空并且仅包含非零元素（逻辑值或实数值）时，该表达式为 true。否则，表达式为 false。

`elseif` 和 `else` 模块是可选的。这些语句仅在 `if...end` 块中前面的表达式为 false 时才会执行。`if` 块可以包含多个 `elseif` 块。

注意：

- 条件表达式中if是必须的，且只能出现一次；
- else可以不出现但若出现则只能一次；
- elseif可多次使用；
- 条件表达式与关键词之间用空格隔开；
- 条件表达式为布尔量。

```
#新建脚本，输入代码并保存
if 3>0
    3
elseif 2>0
    2
end
#点击运行，输出结果为3
3
```

```
#新建脚本，输入代码并保存
if 1                #条件表达式只能为布尔量，不支持将0或1作为true或false使用
    println("true")
end
#点击运行，输出ERROR
ERROR: TypeError: non-boolean (Int64) used in boolean context
```

```
z = if 3 < 0        #if语句也属于复合表达式,可以将其赋给一个变量
       10
    else
         -10
    end
-10                  #点击运行、命令窗口显示-10
```

2.4.4 while语句

当条件表达式为 true 时重复执行的 `while` 循环，一般格式为：

```
while 条件表达式
    #执行语句
end
```

`while expression, statements, end` 计算一个表达式，并在该表达式为 true 时在一个循环中重复执行一组语句。表达式的结果非空并且仅包含非零元素（逻辑值或实数值）时，该表达式为 true。否则，表达式为 false。

```
i=1
while i <= 5
      println(i)
      global i += 1
end
#点击运行、命令窗口显示
1
2
3
4
5
#while循环结束后，命令窗口测试i的当前值
julia> i
6
```

2.4.5 for语句

用来重复指定次数的 `for` 循环，一般格式为：

- 格式1：

```
for 迭代变量=可迭代数集
   #语句块
end
```

- 格式2：

```
for 迭代变量 in 可迭代数集
   #语句块
end
```

- 格式3：

```
for 迭代变量∈可迭代数集
   #语句块
end
```

注意：

- 三个遍历操作符=、in及∈是等价的，可以任选一个；
- ∈输入：\in + tab键
- 可迭代的内容可以是Syslab任意类型

```
for i = 1:5
    print(i)
end
12345   #点击运行、命令窗口显示

for i in [1 2 3 4 5]
    print(i)
end
12345   #点击运行、命令窗口显示

for i ∈ (1, 2, 3, 4, 5)
    print(i)
end
12345   #点击运行、命令窗口显示
```

2.5 函数

在数学中，函数是两个非空集合之间的一种对应关系。数学中对函数的定义是非常严谨的，只要有输入，就一定会有输出，并且，同一个或同一组输入不可能产生不同的输出。这保证了每一个函数的有效性与稳定性。相比之下，程序设计领域对函数的定义更为丰富，这主要是因为各种各样的编程语言都有各自的特点，函数就有着不同的地位、用途与功能。

Syslab中使用的Julia语言中的函数具有极为重要的地位。许多功能的实现都依托各种函数，例如经常使用到的+、-、*、/，==等，这些都是函数。本章节我们介绍常用的函数。大致上，我们将函数分为如下两类：

2.5.1 系统函数

（1）常用数学函数

1. 三角函数：sin(x)、cos(x)、tan(x)、asin(x)、acos(x)、atan(x).
2. 初等函数：abs(x)、sqrt(x)、round(x)(四舍五入取整)、floor(x)(去尾数取整)、mod(x,y)(取余数)、exp(x)(以e为底的指数)、log(x)(以e为底的对数)、log10(x)(以10为底的对数).

（2）常用功能函数

1. sum(x) 求向量/数组元素之和
2. max(x) 求向量/数组的最大值
3. min(x) 求向量/数组的最小值
4. rand(n) 生成一个(0,1)的均匀分布的随机数向量数
5. map(f_name,x) 执行以x为参数，名为f_name的函数

2.5.2 自定义函数

Syslab支持自定义函数。在计算机程序中，函数是代码模块化和复用的基础；将常用的操作写成函数，如果在后续的使用过程中需要用到相同的操作，将自定义函数存盘并运行一遍后即可调用函数。

（1）自定义函数的单行形式

可以用一行代码将简单的函数进行表示，也可以将其称为函数的简洁形式。

[例2.5.1] $f(x)=x^3+3x+1$

```
julia> f(x)=x^3+3*x+1                          #用一行代码即可表示函数
f (generic function with 2 methods)            #输入函数后，敲击回车，提示内容

julia> f(5)                                    #调用函数
141                                            #显示计算结果

julia> f(3.2)                                  #调用函数
43.36800000000001                              #显示计算结果
```

(2) 自定义函数的多行形式

自定义函数的多行格式也可以称作标准形式。在Julia语言中，使用的标准形式编写的函数需要以关键字function和函数的签名为首行，格式如：

```
function funcname(x,y,z)
    ...
end
```

上面格式中的`function`是一个关键字；`funcname`是函数的名称，`x,y,z`是自变量名,其中的`...`是函数内的语句或表达式，以`end`关键字结尾，也可以用`return`关键字指定返回值。

[例2.5.2] 自定义函数，计算样本标准差$s=\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}$，x以向量形式输入。

```
#新建脚本文件，输入函数代码
function mysd(x)                               #定义函数名与自变量
    n = length(x)                              #函数体语句与表达式的开始
    mx = sum(x) / n
    s = 0.0
    for z in x                                 #函数中循环结果的开始
        s += (z - mx)^2
    end                                        #函数中循环结果的结束
    return sqrt(s / (n-1))
end                                            #函数结束，点击运行

mysd (generic function with 1 method)          #输出提示

julia> mysd([1, 2, 3, 4, 5])                   #函数调用，输入一个向量
1.5811388300841898                             #返回计算结果
```

(3) 多返回值情况

函数的最后一个表达式为函数的返回值， 也可以用`return y`这样的方法返回值。

如果需要返回多个值， 可以将多个值组成一个元组(tuple)返回， 通过这样的方式就可以返回多个结果。

```
function summ(x)                               #定义函数
  xm = sum(x) / length(x)
  xs = sum(x .^2) / length(x)
  return (xm, xs)                              #(xm, xs)即是一个元组，作为返回类型
end
summ (generic function with 1 method)          #输出提示信息

julia>res1, res2 = summ([1, 2, 3, 4, 5])       #输入信息
(3.0, 11.0)                                    #输出结果
```

2.6 Syslab绘图

图形函数包括二维和三维绘图函数，在绘图前需提前加载图形函数库 TyPlot (using TyPlot)，该函数库用于以可视化形式呈现数据和通信的结果，并以交互方式或编程方式自定义绘图。

2.6.1 二维图绘制（plot）

语法	说明
plot(X,Y)	创建Y数据与X对应值的二维线图。
plot(X,Y,Fmt)	创建二维线图，并设置线型、标记符号和颜色。
plot(X1,Y1,...,Xn,Yn)	绘制多组X、Y的图，所有线条使用相同的坐标区。
plot(X1,Y1,Fmt1,...,Xn,Yn,Fmtn)	绘制多组X、Y的图，所有线条使用相同的坐标区，并设置每条线的线型、标记符号和颜色。
plot(Y)	创建Y中数据对每个值索引的二维线图。
plot(Y,Fmt)	创建Y中数据对每个值索引的二维线图，并设置每条线的线型、标记符号和颜色。

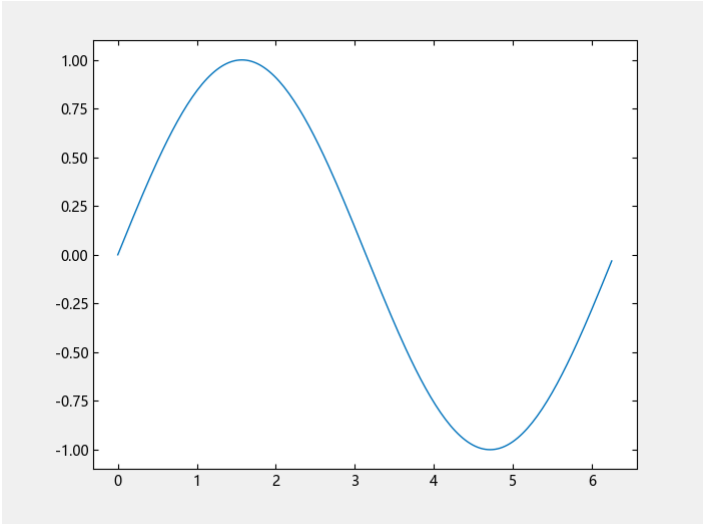
（1）创建X, Y 线图

创建横坐标为x, 纵坐标为y的线图。

新建脚本，输入下列命令，保存并运行：

```
x = 0:pi/100:2*pi
y = sin.(x)
plot(x,y)
```

输出图像：



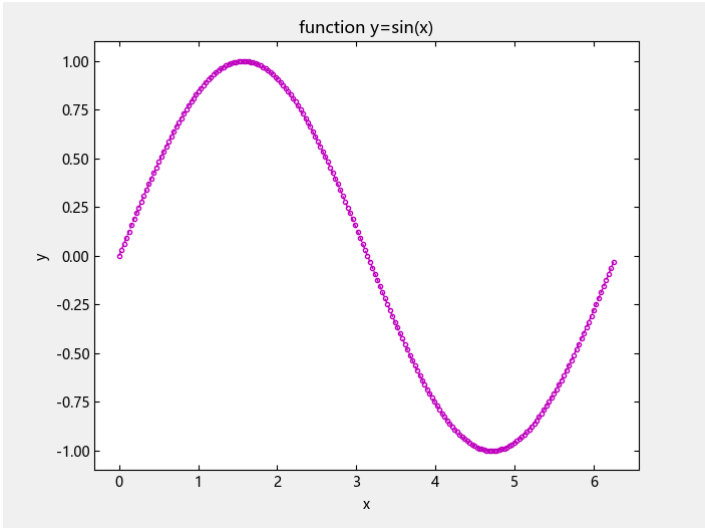
（2）指定线型、颜色及标记

将曲线指定为紫色，标记圆点且设置为虚线样式，添加横纵坐标、标题。

新建脚本，输入下列命令，保存并运行：

```
x = 0:pi/100:2*pi
y = sin.(x)
plot(x,y,"m.--")
title("function y=sin(x)")
xlabel("x")
ylabel("y")
```

输出图像：



指定线型信息补充：

线型	说明	表示的线条
“-”	实线	————
“--”	虚线	--------
“:”	点线	.....
“-.”	点划线	-·-·-·-·-

颜色名称	短名称	RGB三元组	十六进制颜色代码
red	r	[1,0,0]	“#FF0000”
green	g	[0,1,0]	“#00FF00”
blue	b	[0,0,1]	“#0000FF”
magenta	m	[1,0,1]	“#FF00FF”
black	k	[0,0,0]	“#000000”

(3) 创建多条线图

创建y1, y2与x关系的线图，并在同一张图中显示。

新建脚本，输入下列命令，保存并运行：

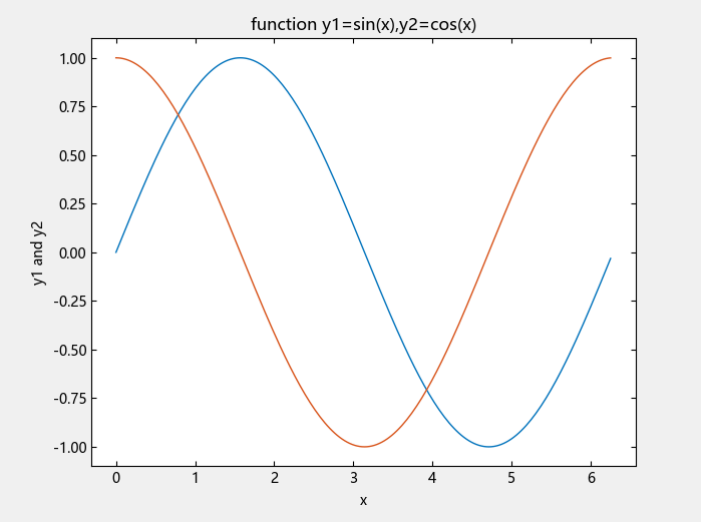
1.采用 plot(X1,Y1,...,Xn,Yn) 方式

```
x = 0:pi/100:2*pi
y1 = sin.(x)
y2 = cos.(x)
plot(x,y1,x,y2)
title("function y1=sin(x),y2=cos(x)")
xlabel("x")
ylabel("y1 and y2")
```

2.采用hold("on")的方式

```
x = 0:pi/100:2*pi
y1 = sin.(x)
y2 = cos.(x)
plot(x,y1)
hold("on")
plot(x,y2)
title("function y1=sin(x),y2=cos(x)")
xlabel("x")
ylabel("y1 and y2")
```

两种方式输出一样的图像：



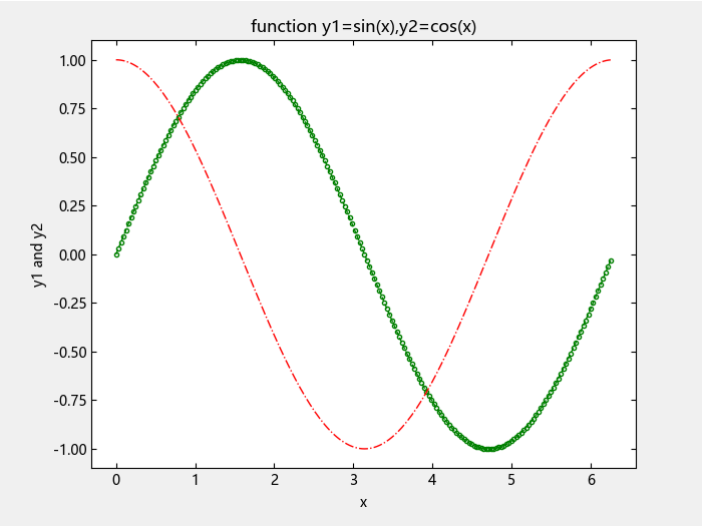
（4）指定多条线图的线型、颜色及标记

将曲线y1指定为绿色、虚线、圆点样式，曲线y2指定为红色、点划线、无标记样式。

新建脚本，输入下列命令，保存并运行：

```
x = 0:pi/100:2*pi
y1 = sin.(x)
y2 = cos.(x)
plot(x, y1, "g.--", x, y2, "r-.")
title("function y1=sin(x),y2=cos(x)")
xlabel("x")
ylabel("y1 and y2")
```

输出图像：



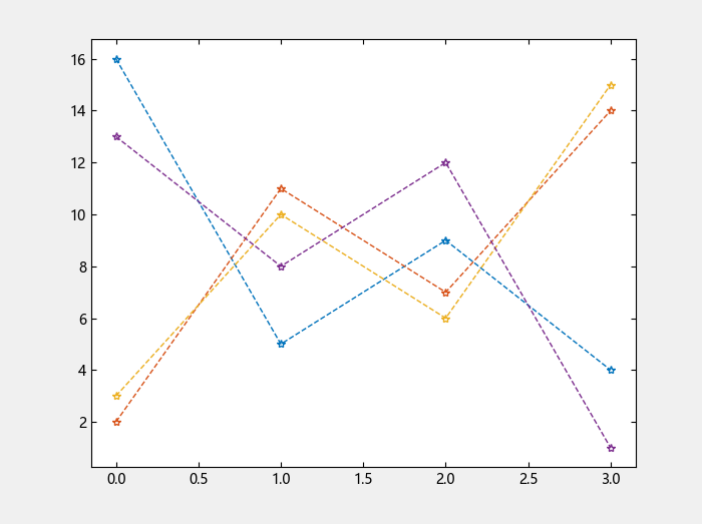
（5）根据矩阵创建线图

创建 Y 的二维线图：横坐标为行数，纵坐标为每一行对应的值。将矩阵的每一列绘制为单独的线条，将曲线指定为虚线，标记为星号。

新建脚本，输入下列命令，保存并运行：

```
Y=[16   2   3   13;
    5   11  10  8;
    9   7   6   12;
    4   14  15  1]
plot(Y,"*--")
```

输出图像：



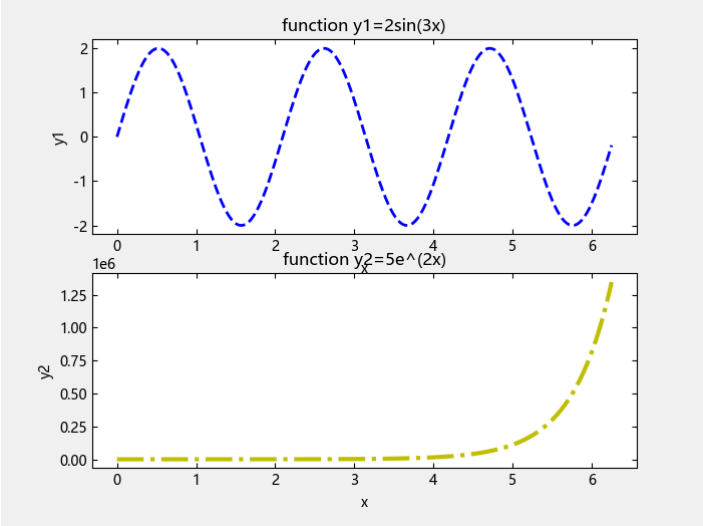
（6）指定绘图坐标区并添加标题及标签

使用方法：subplot（m,n,p）。subplot是将多个图画到一个平面上的工具。其中，m表示是图排成m行，n表示图排成n列，也就是整个figure中有n个图是排成一行的，一共m行，如果m=2就是表示2行图。p表示图所在的位置，p=2表示从左到右从上到下的第2个位置。linewidth用于指定线宽。

新建脚本，输入下列命令，保存并运行：

```
x = 0:pi/100:2*pi
y1 = 2 .* sin.(3 .* x)
y2 = 5 .* exp.(2 .* x)
ax1 = subplot(2, 1, 1)
plot(ax1, x, y1, "b--", linewidth = 2)
title(ax1,"function y1=2sin(3x)")
xlabel("x")
ylabel("y1")
ax2 = subplot(2, 1, 2)
plot(ax2, x, y2, "y-.", linewidth = 3)
title(ax2,"function y2=5e^(2x)")
xlabel("x")
ylabel("y2")
```

输出图像：



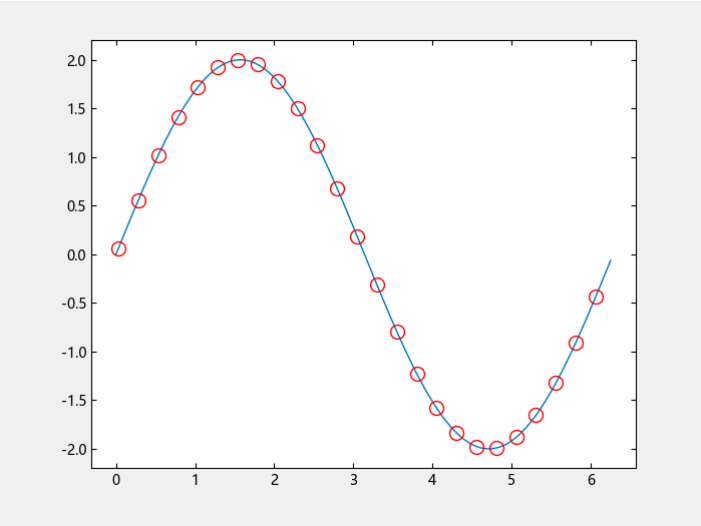
（7）特定数据点显示标记

通过指定标记符号并将 markevery 属性设置为 名称-值 对组，创建一个线图并每隔四个数据点显示一个标记。

新建脚本，输入下列命令，保存并运行：

```
x = 0:pi/100:2*pi
y = 2 .* sin.(x)
plot(x, y, "-o", markevery = 1:8:length(y), markeredgecolor = "r", markersize
= 10)
```

输出图像：



数据点索引信息设置补充：

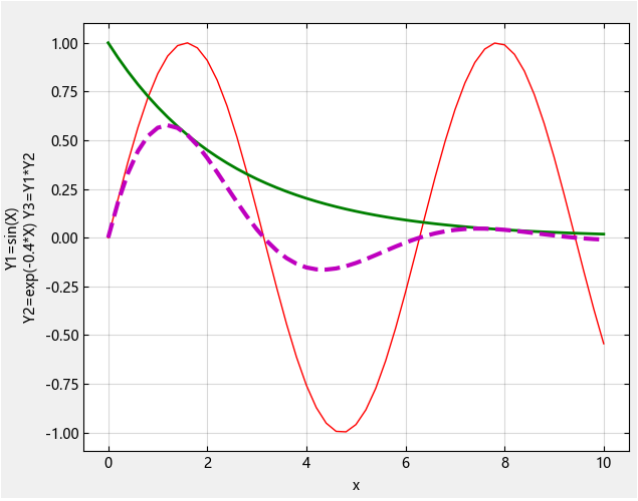
类型	用法	说明
markevery	markevery=nothing	每个点都绘制标记
markevery	markevery=N	从标记0开始，每N个点绘制一个标记
markevery	markevery=start:N:end	从开始点开始(包括开始点)，到结束点结束(不包括结束点)每N个点绘制一个标记
markeredgecolor	“r”，”g”，”b”，”m”，”y”，”k”，”w”	标记轮廓颜色为红、绿、蓝、紫、黄、黑、白色
markerfacecolor	“r”，”g”，”b”，”m”，”y”，”k”，”w”	标记填充颜色为红、绿、蓝、紫、黄、黑、白色
markersize	markersize=value	标记大小为任意值

(8) 其他示例

1. 新建脚本，输入下列命令，保存并运行：

```
x = 0:0.2:10
y1 = sin.(x)
y2 = exp.(-0.4 .* x)
y3 = y1 .* y2
plot(x, y1, "r")
hold("on")
plot(x, y2, "g", linewidth = 2)
plot(x, y3, "m--", linewidth = 3)
grid("on")
xlabel("x")
ylabel("Y1=sin(X)
Y2=exp(-0.4*X) Y3=Y1*Y2")
```

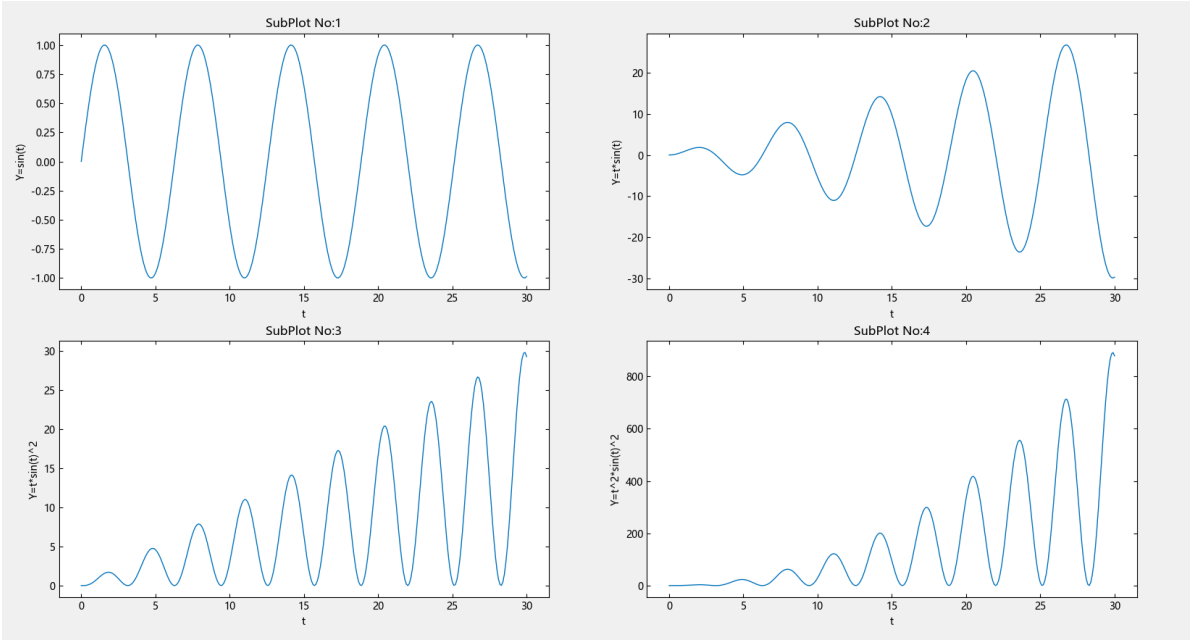
输出图像：



2. 新建脚本，输入下列命令，保存并运行：

```
t=0:0.1:30
ax1 = subplot(2,2,1);
plot(ax1,t,sin.(t))
title(ax1,"SubPlot No:1")
xlabel(ax1,"t")
ylabel(ax1,"Y=sin(t)")
ax2 = subplot(2,2,2)
plot(ax2,t,t.*sin.(t))
title(ax2,"SubPlot No:2")
xlabel(ax2,"t")
ylabel(ax2,"Y=t*sin(t)")
ax3 = subplot(2,2,3)
plot(ax3,t,t.*sin.(t).^2)
title(ax3,"SubPlot No:3")
xlabel(ax3,"t")
ylabel(ax3,"Y=t*sin(t)^2")
ax4 = subplot(2,2,4)
plot(ax4,t,(t.^2).*sin.(t).^2)
title(ax4,"SubPlot No:4")
xlabel(ax4,"t")
ylabel(ax4,"Y=t^2*sin(t)^2")
```

输出图像：



2.6.2 三维图的绘制（plot3）

语法	说明
plot3(X,Y,Z)	绘制三维空间中的坐标
plot3(X,Y,Z,Fmt)	创建三维线图，并设置线型、标记符号和颜色
plot3(X1,Y1,Z1,...,Xn,Yn,Zn)	绘制多组三维线图，所有线条使用相同的坐标区
plot3(X1,Y1,Z1,Fmt1,...,Xn,Yn,Zn,Fmtn)	绘制多组三维线图，所有线条使用相同的坐标区，并设置线型、标记符号和颜色

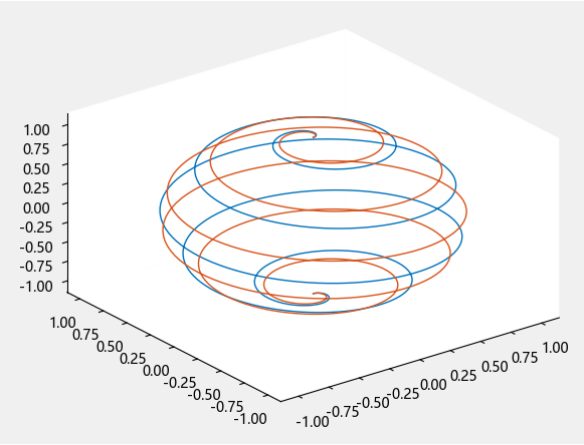
（1）创建多条线图

创建两组 x、y 和 z 坐标。

新建脚本，输入下列命令，保存并运行：

```
t = 0:pi/500:pi
xt1 = sin.(t) .* cos.(10 .* t)
yt1 = sin.(t) .* sin.(10 .* t)
zt1 = cos.(t)
xt2 = sin.(t) .* cos.(12 .* t)
yt2 = sin.(t) .* sin.(12 .* t)
zt2 = cos.(t)
plot3(xt1, yt1, zt1, xt2, yt2, zt2)
```

输出图像：



（2）指定线型、颜色及标记

创建向量 t、xt 和 yt，并将这些向量中的点绘制为带 10 磅圆形标记的蓝线。使用十六进制颜色代码指定标记的填充颜色为浅蓝色。

新建脚本，输入下列命令，保存并运行：

```
t = 0:pi/20:10*pi;
xt = sin.(t);
yt = cos.(t);
plot3(xt,yt,t,"-o",color = "b",
      markersize=10,
      markerfacecolor= "#D9FFFF")
```

输出图像：