MATLAB 语言及应用

MATLAB Language and Its Applications

第四章 数值计算
Ch.4 Numerical Calculations

刘世东
Shidong Liu

School of Physics and Physical Engineering
Qufu Normal University

June 18, 2020

本章目录—MATLAB 以数值见长

数值极限

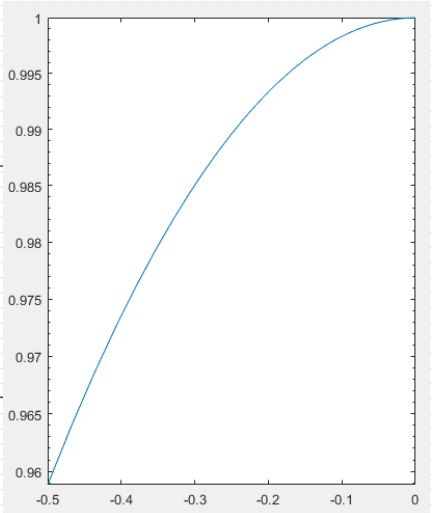
MATLAB 的数值计算中, 没有专门的求极限的指令. 因为 MATLAB 中的数值精度有限, 无法描述无穷量, 故尽量不要使用数值方法计算函数的极限.

某些情况下, 可以适当构造递减自变量序列查看数值极限.

$f(x) = \dfrac{1-\cos 2x}{2x \sin x}$ 在 x 趋于 0 时的极限.

根据洛必达法则可知, 极限值为 1.

```
N=20; % N不能太大
n=1:0.1:N;
dx=-0.5.^n; % 随着N增大, dx趋于0
x0=0; %极限点
x=x0+dx; % 产生x趋于x0的序列
fx=(1-cos(2*x))./(2*x.*sin(x));
semilogy(x,fx)
```



数值求极限的一般方法

1. 产生趋于 x_0 的序列. 上例中使用的是 $x_n = x_0 - 0.5^n$, n 变大时, x_n 趋于 x_0.
2. N 取值一般不能太大[1].
3. 计算函数值, 通过绘图或者数值表格几个找到极限值 (或者极限范围).

通过上述方法计算以下函数的极限

$$\frac{x^2 - 2x - 3}{x - 3} \quad \text{at } x \to 3$$

$$\frac{1 + \sin(x)}{x} - \frac{1}{\sin(x)} \quad \text{at } x \to 0$$

$$\frac{1 - \cos(2x)}{x^2} \quad \text{at } x \to 0$$

[1]实际取值大小与具体函数表达式有关

符号法求极限 limit

limit 函数用来求符号表达式的极限. 基本语法 limit(expr,x,a)

上例中的函数, 用符号法求极限如下

```
syms x % 创建符号变量x
fx=(1-cos(2*x))/(2*x*sin(x)); % 符号函数表达式fx
limit(fx,x,0)
%运行结果
ans = 1
```

limit 求导

```
syms x h % 定义两个符号变量, 变量之间只能用空格
limit((sin(x + h) - sin(x))/h, h, 0)
%运行结果
ans = cos(x)
```

符号计算比较耗资源, 多数的科研工作中都是使用数值计算.

数值差分 diff

`Y = diff(X,n)` 计算向量 X 相邻元素的差分, n 表示阶数, 默认为 1.

若 $n = 1$, 则 $Y = [X(2) - X(1), X(3) - X(2), ..., X(m) - X(m-1)]$, 故 Y 的元素个数与 X 的元素个数相差 1.

```
X = [1 1 2 3 5 8 13 21]; % 8个元素
Ydiff1 = diff(X)
%运行结果
Ydiff1 =
0     1     1     2     3     5     8 % Ydiff1只有7个元素

Ydiff2 = diff(X,2)
%运行结果
Ydiff2 =
1     0     1     1     2     3 % Ydiff1只有6个元素
```

数值梯度 gradient

`Y = gradient (F)` 计算 F 的梯度.

Y 的非端点元素由 $(F_{n+1} - F_{n-1})/2$ 计算得到, 第一个元素 $F_2 - F_1$, 最后一个元素 $F_{end} - F_{end-1}$. 注意 `gradient` 返回的 Y 元素个数与 X 相同.

```
X = [1 1 2 3 5 8 13 21]; % 8个元素
Ydiff1 = diff(X)
Ygrad = gradient(X)
%运行结果
Ydiff1 =
0     1     1     2     3     5     8 % Ydiff1只有7个元素

Ygrad =
0    0.5000    1.0000    1.5000    2.5000    4.0000    6.5000    8.0000 % 8个元素
```

`[FX,FY] = gradient(F)`, F 是矩阵时, Fx 返回 x 方向梯度, Fy 返回 y 方向的梯度. `doc gradient` 查看例子.

diff 与 gradient 的区别联系

diff 和 gradient 在计算差分的原理不同, diff 相当于是向前差分, 而 gradient 相当于是中心差分. 另外, 两者在返回元素个数上相差 1.

对于随自变量 x 变化的函数 $f(x)$, 当 x 序列相邻元素差值大时, diff 与 gradient 的返回值差异明显, 但是当 x 序列相邻元素差值较小时, diff和gradient的返回值差异很小.

下列代码显示diff与gradient计算函数 $f = 4 - (x-2)^2$ 在 $[0, 3]$ 之间的差分区别.

```
h = .1;%通过改变h的大小查看diff与gradient的差异
x = 0:h:3;
fx = 4-(x-2).^2;
dfx = diff(fx); gfx = gradient(fx);
plot(x,fx,'r-*')
hold on
plot(x(1:end-1),dfx,'b-o')
plot(x,gfx,'k-s')
legend('fx','dfx','gfx')
hold off
```

差分等价于求函数的导数, 将 dfx 与 gfx 分别除以 h 就得到函数 f 的导数值, 即 $-2x + 4$

MATLAB 中的数值积分函数

随着 MATLAB 的版本更新, 好多以前的数值积分函数都不再建议使用. 在使用时请注意一下

integral	Numerical integration
integral2	Numerically evaluate double integral
integral3	Numerically evaluate triple integral
quadgk	Numerically evaluate integral, adaptive Gauss-Kronrod quadrature
quad2d	Numerically evaluate double integral, tiled method
cumtrapz	Cumulative trapezoidal numerical integration
trapz	Trapezoidal numerical integration
polyint	Polynomial integration

`quad`, `quadv`, `quadl` 被 `integral` (2012a 引入) 代替. `dblquad` 被 `integral2` (2012a 引入) 代替. `triplequad` 被 `integral3` (2012a 引入) 代替.

单重积分 integral — 单重积分优先选择的函数

`q = integral(fun,xmin,xmax)` fun 可以是匿名函数 (简单函数) 也可以 M 文件函数 (复杂函数). fun 必须是数组运算, 上下限可以是 `inf` 或 `-inf` 或复数

$$q = \int_0^1 x^2 \,\mathrm{d}x$$

```
f = @(x) x.^2;
q = integral(f,0,1)
%运行结果
q =  0.3333 % 根据微积分精确值为1/3, 可以通过format rat 转换分数格式
```

参数方程积分: $\int_0^1 (x^2 + c)\,\mathrm{d}x$

```
fxc = @(x,c) x.^2+c;% 二元匿名函数
qc = integral(@(x)fxc(x,2),0,1)
%运行结果
qc = 2.3333
```

其他的单重积分函数调用形式与 `integral` 类似, 可以通过 `doc` 查看.

integral 单重积分练习

$$\int_0^{\infty} \frac{1}{x^2+1}\,\mathrm{d}x \qquad 1.5708\ (\pi/2)$$

$$\int_0^{\infty} \exp(-x^2)\cos(2x)\,\mathrm{d}x \qquad 0.3206$$

$$\int_0^{\pi/2} \sin^6(x)\cos^7(x)\,\mathrm{d}x \qquad 0.0053\ (16/3003)$$

$$\int_1^{8} \frac{(1+x^{2/3})^6}{x^{1/3}}\,\mathrm{d}x \qquad 1.6714\times 10^4\ (\frac{3}{14}(5^7-2^7))$$

上式中右边为积分值 (精确值).

NOTE: 数值积分存在误差, MATLAB 提供的积分函数返回结果是有误差的, `integral` 的默认误差为 10^{-10}.

二重积分 integral2

`q = integral2(fun,xmin,xmax,ymin,ymax)`: 求二元函数 fun 在积分区间为 $\mathrm{xmin} \le x \le \mathrm{xmax}$, $\mathrm{ymin}(x) \le y \le \mathrm{ymax}(x)$ 的积分值.

$$\int_1^2 \int_{\sin(x)}^{\cos(x)} xy \,\mathrm{d}y \,\mathrm{d}x$$

```
f = @(x,y) x.*y;
ymin = @(x)sin(x);
ymax = @(x)cos(x);
q = integral2(f,1,2,ymin,ymax) % 积分顺序的内外层有点反人类
% q=integral2(@(x,y) x.*y,1,2,@(x)sin(x),@(x)cos(x)) 可读性不好
% 运行结果
q =
-0.6354
```

可与精确值 $\dfrac{\cos 4 - \cos 2}{8} + \dfrac{2\sin 4 - \sin 2}{4}$ 比较.

`integral2` 的默认精度为 10^{-10}

给定积分区域的二重积分举例

对于常数积分上下限或者给定确定函数关系的上下限积分比较容易. 若给定的积分区域是由 x 和 y 构成的闭合面积, 要稍微复杂一点. 主要有两种方法:

1. 通过函数关系改成上例中的积分上下限形式进行积分
2. 通过逻辑或者关系延拓被积分函数, 使积分区域变成矩形区域.

被积函数 $f(x,y)=\sqrt{100^2-x^2}$, 积分区域 $x^2+y^2\leq 100^2$.

根据积分区域的函数关系, x 的积分区域 $[-100,100]$, y 的积分区域 $[-\sqrt{100^2-x^2},\sqrt{100^2-x^2}]$.

方法一:

```
f = @(x,y) sqrt(10000-x.^2);
ymin = @(x) -sqrt(10000-x.^2);
ymax = @(x) sqrt(10000-x.^2);
q = integral2(f,-100,100,ymin,ymax)
%运行结果
q =
2.6667e+06 % 精确结果 (8/3)e+06
```

方法二:

```
f = @(x,y) sqrt(10000-x.^2).*(x.^2+y.^2<=10000);
q = integral(f,-100,100,-100,100)
% 运行结果
q =
2.6669e+06

% 运行出现warning:
Warning: Reached the maximum number of function evaluations (10000).
The result fails the global error test.
> In integral2Calc>integral2t (line 129)
In integral2Calc (line 9)
In integral2 (line 106)
In Untitled (line 2)
```

wanrning 原因是计算次数超过了默认的计算次数上限. 超过计算次数上限的原因是尺度跨度太大, 这里圆的半径为 100, 导致其平方为 10000, 所以网格数就会增多 (为保证计算准确, 网格不能太大), 从而计算量就会增大. 将 10000 改成 100 后, 不会再出现 warning 警告.

二重积分计算练习

$$\int_{10}^{20}\int_{5x}^{x^2} \exp(\sin x)\ln y \,\mathrm{d}y\,\mathrm{d}x$$

$$\int y\sin x + x\cos y \,\mathrm{d}y\,\mathrm{d}x, \quad x\in[\pi,2\pi] \quad y\in[0,\pi]$$

$$\int_1^2\int_0^1 x^y \,\mathrm{d}x\,\mathrm{d}y \quad \text{Note limits of } x \text{ and } y$$

三重积分 integral3

`q = integral3(fun,xmin,xmax,ymin,ymax,zmin,zmax)`: 语法与 `integral2` 一样, x 是外层积分, 然后 y, 最后 z. 故 z 可以是 x 和 y 的函数, y 可以是 x 的函数, x 只能是数值.

具体示例参看 `doc integral3`

多重积分与单重积分之间的关系

多重积分可以用单重积分进行计算, 只是过程麻烦且不一目了然.

这里以二重积分为例进行说明

$$\int_1^2 \int_{\sin(x)}^{\cos(x)} xy \, \mathrm{d}y \, \mathrm{d}x \quad \text{Result} \ -0.6354$$

```
function q=Use1Cal2
q=integral(@cal1Int,1,2); % 计算外积分
    function q1=cal1Int(x) % 内嵌函数
        for k=1:length(x)
            q1(k)=integral(@(y) x(k).*y, sin(x(k)),cos(x(k))); % 内积分得到x的函数 q1
        end
    end
end
```

利用 arrayfun 函数, 只需一行代码表示上述程序 (NOTE: 可读性稍差)

```
q=integral(@(x)arrayfun(@(xx)integral(@(y)xx.*y,sin(xx),cos(xx)),x),1,2)
```

畸形的积分表达式

利用以上方法, 可以处理一些畸形的积分, 例如

$$\int_{10}^{100} \left(\frac{1}{\int_{x}^{x^2} y \, \mathrm{d}y} \right) \mathrm{d}x$$

arrayfun 函数格式: q=integral(@(x)arrayfun(@(xx)1./integral(@(y)y,xx,xx.^2),x),10,100)

嵌套函数形式:

```
function q=Use1Cal2
q=integral(@cal1Int,10,100);
    function q1=cal1Int(x)
        for k=1:length(x)
            q1(k)=1./integral(@(y) y, x(k),x(k)^2);
        end
    end
end
```

畸形积分练习

$$\int_{0.2}^{2} 2y\exp(-y^2)\left[\int_{-1}^{1}\frac{\exp(-x^2)}{x^2+y^2}\,\mathrm{d}x\right]^2\mathrm{d}y$$

结果为 10.2135

$$\int_0^{40}\int_0^{40}\frac{f^2+g^2}{\sqrt{u^2+v^2}}\,\mathrm{d}u\,\mathrm{d}v$$

其中

$$f=\int_0^1\int_0^1[1-\cos(2\pi x)]\cdot[1-\cos(2\pi y)]\cdot\cos(ux+vy)\,\mathrm{d}x\,\mathrm{d}y$$

$$g=\int_0^1\int_0^1[1-\cos(2\pi x)]\cdot[1-\cos(2\pi y)]\cdot\sin(ux+vy)\,\mathrm{d}x\,\mathrm{d}y$$

```
q=integral2(@(u,v)((arrayfun(@(uu,vv)integral2(@(x,y)(1-cos(2*pi*x))
.*(1-cos(2*pi*y)).*cos(uu*x+vv*y),0,1,0,1),u,v)).^2+
(arrayfun(@(uu,vv)integral2(@(x,y)(1-cos(2*pi*x)).*(1-cos(2*pi*y)).*
sin(uu*x+vv*y),0,1,0,1),u,v)).^2)./sqrt(u.^2+v.^2),0,40,0,40);
```

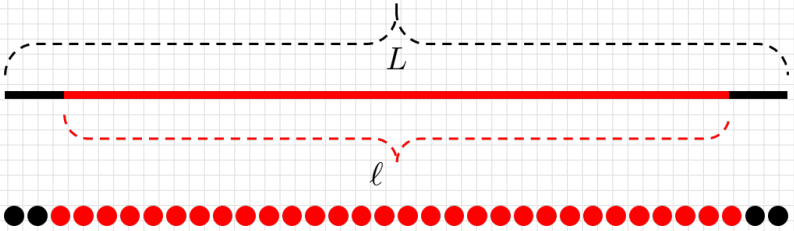
上一页积分 M 函数文件形式 (嵌套函数)

```
function q = ComplexInt
q = integral2(@(u,v) arrayfun(@(uu,vv) tempFun(uu,vv),u,v),0,40,0,40);

% 以下定义中间积分函数(定义的是标量函数, 与前前页的数组函数有区别)
function z = tempFun(u,v)
f = @(x,y)(1-cos(2*pi*x)).*(1-cos(2*pi*y)).*cos(u.*x+v.*y);
g = @(x,y)(1-cos(2*pi*x)).*(1-cos(2*pi*y)).*sin(u.*x+v.*y);
fint = integral2(f,0,1,0,1);
gint = integral2(g,0,1,0,1);
z = (fint.^2+gint.^2)./sqrt(u.^2+v.^2);
end
end
```

n 重积分的蒙特卡洛法

n 重积分可以用前面学过的单重, 二重或者三重积分联合求解, 但是当 n 很大时, 会多次调用 MATLAB 函数, 计算量会增大, 耗费资源严重. 因此可以使用蒙特卡洛方法[2]. 此方法的计算几乎与积分重数无关. 为说明蒙特卡洛积分可以从积分的概念出发:

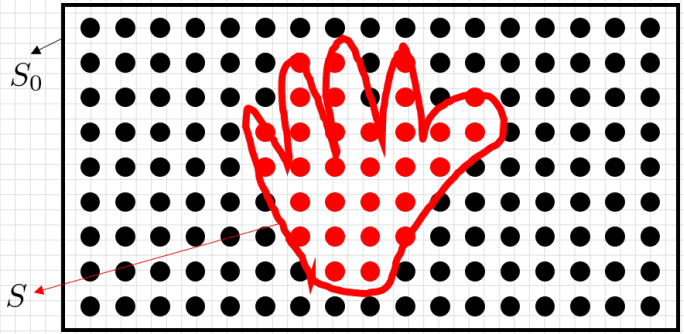
单重积分: $q = \int_a^b f(x)\,\mathrm{d}x = \sum_i^n f(x_i)\Delta x = \sum_i^n f(x_i)\frac{\ell}{n} = \frac{\ell}{n}\sum_i^n f(x_i)$



在直线段 L 上均匀取 m 个点, 落在 ℓ 内的点数为 n, 则知 $\ell = \frac{n}{m}L$, 则积分值为 $q = \frac{L}{M}\sum_i^n f(x_i)$.

[2]随机抽样法.

二重积分: $q = \int_S f(x, y)\,\mathrm{d}x\,\mathrm{d}y = \frac{S}{n}\sum_{i,j} f(x_i, y_j)$



如图 $S = \frac{n}{m}S_0$. 故积分 $q = \frac{S_0}{m}\sum_{i,j} f(x_i, y_j)$.

优点: 面积 S_0 比面积 S 好求的多, 我们要做的是求出 $f(x_i, y_j)$ 的和. m 是已知的, n 可以通过逻辑关系表达.

同理, 三重积分或更高重数的积分, 则是用超立方体的体积与抽样数进行决定.

蒙特卡洛积分例子

$$q = \int_1^2 \left(\int_{x_1}^{2x_1} \left(\int_{x_1 x_2}^{2x_1 x_2} x_1 x_2 x_3 \, \mathrm{d}x_3 \right) \mathrm{d}x_2 \right) \mathrm{d}x_1$$

通过微积分知, 该积分的精确值为 $11475/64 \simeq 179.296875$.

```
% 每次运行结果不一样
f = @(xx1,xx2,xx3) xx1.*xx2.*xx3;
m = 10000;% m 越大计算结果越精确
x1 = unifrnd(1,2,1,m);% 产生m个在1和2之间的连续随机数
x2 = unifrnd(1,4,1,m);
x3 = unifrnd(1,16,1,m);
neiBuDianZuoBiao = (x2>x1)&(x2<2*x1)&(x3>x1.*x2)&(x3<2*x1.*x2); % for所求体积对应函数和
daTiji = 1*3*15;
fxi = f(x1,x2,x3);%求函数值
fxi( neiBuDianZuoBiao) = 0;% 非(波浪号) 没有显示
fxiHe = sum(fxi); % 可以合写为 fxiHe = sum(fxi(neiBuDianZuoBiao));
q = daTiji*fxiHe/m
```

多算几次求平均值即可, 结果非常接近真值, 且计算速度非常快.

一元代数方程解

一般地, 非线性方程 $f(x)=0$ 的解很难解析求解 (若解存在的话). 实际中, 我们有多种方法进行非线性方程的求解.

1. 作图法, 找到 $f(x)$ 与 x 轴的交点;
2. 做表法, 计算一系列 $f(x)$ 的值, 然后...
3. 算法上: 二分法, 迭代法, 切线法等等

MATLAB 中 `fzero` 函数可以求解一元非线性方程 $f(x)=0$ 的解. M 语法

`fzero(fun,x0)`:

计算函数 fun 在 x0 附近或者 x0 定义的零点 (一次只能求一个). x0 通常称之为试探零点, 实际问题中往往通过不断尝试得到或者通过画图得到.

利用 MATLAB 求解的一般步骤

1. `plot` 作图找试探零点
2. `fzero` 求解.

求 $f(x)=x-sin(x)-1=0$ 的解.

作图发现, $f(x) = x - sin(x) - 1 = 0$ 只可能有一个根在 0 附近 (取的画图范围大时, 试探零点往往越偏离真值).

$f(x) = x - sin(x) - 1 = 0$ 在 0 附近的根

```
f = @(x) x-sin(x)-1;
x0 = fzero(f,0) % try x0 = fzero(f,[0,3])
% One M: x0=fzero(@(x) x-sin(x)-1,1.5)
x0 =
1.9346
```

对于有多个零点的方程, 需要计算多次 (使用不同的试探零点)

其他注意事项, fzero 一般只能给出穿过 x 轴的根, 与 x 轴相切的根一般无法求算, 除非试探零点正好设置成此值.

另外, 广义上, 一元线性方程也可以看作一元非线性方程, 因此 fzero 也可以解一元线性方程的根.

一元非线性方程的根练习

计算下列方程 $f(x)=0$ 在 $[-2\,,2]$ 区间内的根.

$$
\begin{aligned}
f(x) &= 2\sin^2 x - x^2 \\
f(x) &= x - \tan x \\
f(x) &= x\exp(x) - 1 \\
f(x) &= x^2 - 1 \\
f(x) &= \sin^2 x - 0.5
\end{aligned}
$$

最后##个有解析解/注意比较/

对于多项式 $x^2-1=0$ 的根, 我们需要求两次才能得到全部的根, MATLAB 提供了可以一次性求多项式全部根的函数, 即 roots.

非线性方程组的根

非线性方程组的根可以通过 fsolve 函数求算, 一般有多个根. M 语法

x = fsolve(fun,x0) 求解非线性方程组 fun 在 x0 附近的根.

$$\begin{cases} \sqrt{x^2+y^2+25} = 7 \\ 2x+4y = 4 \end{cases}$$

```
function x=fsolveTest
x0=[4;-1]; % 也可以是行向量, 但是建议使用列向量; try x0=[-4,2]
x=fsolve(@eqs,x0);
% 嵌套函数, 定义方程组
function f=eqs(xx)
x1=xx(1); % 可以不用赋值, 直接在下面的表达式中使用 xx(1) 和 xx(2)
x2=xx(2);
f1=sqrt(25+x1.^2+x2.^2)-7;
f2=2*x1+4*x2-4;
f=[f1;f2]; % 也可以是行向量, 但是建议使用列向量
end % 嵌套函数结束
end
```

非线性方程组的根练习

$$\begin{cases} x^3 + 2y - 3 = 0 \\ x^2 + 3y^2 - 4 = 0 \end{cases}$$

$$\begin{cases} \sin^3(x^2 + y) + y^2 - x - \dfrac{7}{4} = 0 \\ x^2 + 3y^2 - 31 = 0 \end{cases}$$

$$\begin{cases} \tan x - \sqrt{y} = 0 \\ \sin^2 x - \dfrac{y}{4} - \dfrac{1}{4} = 0 \end{cases}$$

线性方程组的解

线性方程组利用 MATLAB 矩阵除法非常容易求解. 对于线性方程组 $Ax = b$, 其解的 MATLAB 命令为 `x = A\b`. 一般地, 当方程组的个数 m, 未知数的个数 n

1. $m = n$, 一组确定解;
2. $m > n$, 无解, MATLAB 给出最小平方解.
3. $m < n$, 无数解. MATLAB 给出一个基本解, 其中 x 中包含 $n - m$ 个 0.

线性方程组的解

m=n 的情况

$$3x + 5y = 1$$
$$2x + 2y = 3$$

```
A=[3 5;2 2];
b=[1;3];
xy=A\b
%运行结果
xy =
  3.2500
-1.7500
```

m>n 的情况

$$3x + 5y = 1$$
$$2x + 2y = 3$$
$$2x + 5y = 1$$

```
A=[3 5;2 2;2 5];
b=[1;3;1];
xy=A\b
%运行结果
xy =
1.6883
-0.6104
```

m<n 的情况

$$3x + 5y + 3z = 1$$
$$2x + 2y + 3z = 3$$

```
A=[3 5 3;2 2 3];
b=[1;3];
xy=A\b
%运行结果
xy =
0
-0.6667
1.4444
```

当多项式的形式为 $xA = b$ 时, 使用 $x = b/A$.

MATLAB 中如何表示多项式

多项式的阶 (次数) 指多项式中的最高幂次项. 一个 n 阶多项式有 n+1 个系数. 数学上, 我们一般将多项式表示如下两种形式的书写

$$p = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \text{ or}$$
$$p = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

多项式的系数下标与未知量的次数是 '相等' 的.

但是在 MATLAB 中, 我们表示多项式, 使用一个行向量表示, 例如行向量 $p = [p_1, p_2, \cdots, p_n, p_{n+1}]$. 注意的是, MATLAB 中用 p 表示的多项式是降幂排列的, p_1 对应 x^n, p_{n+1} 对应 x^0 (也就是 1), 即

$$p = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + p_{n+1}$$

系数的次序与未知量的幂和为 $n+1$

多项式的根 roots

roots(p): 计算 p 对应的多项式的所有根 (包括复数根).

计算 $x^2 + x - 2 = 0$ 的根.

```
p =[1,1,-2];
r = roots(p)
% 运行结果
r =
-2
1
```

计算 $x^2 - 2x + 1 = 0$ 的根.

```
p =[1,-2,1];
r = roots(p)
% 运行结果
r =
1
1
```

计算 $x^4 - 1 = 0$ 的根.

```
p =[1,0,0,0,-1];
r = roots(p) % 运行结果如下
r =
-1.0000 + 0.0000i
 0.0000 + 1.0000i
 0.0000 - 1.0000i
 1.0000 + 0.0000i
```

roots 返回 n 个根, 即便相等也返回相等的根.

多项式的值计算与多项式的系数计算

一. `y = polyval(p,x)`: 计算多项式在 x 的值. (x 可以是向量, 返回对应个元素的多项式的值)

计算多项式 $x^2 + x - 2$ 在 $x = [0, 1, -2, 3]$ 处的值.

```
p = [1,1,-2];
x = [0,1,-2,3];
y = polyval(p,x)
% 运行结果
y = -2     0     0    10
```

二. `poly(r)` 返回根为 r 的多项式的系数.

```
r = [1,-2];
p = poly(r)
% 运行结果
p = 1     1    -2
```

```
r = [1]; % try r = [1,1]
p = poly(r)
% 运行结果
p = 1    -1
```

返回多项式的阶数与 r 的长度相同.

多项式向量形式输出为字符形式

MATLAB 内 (可能是因为我不知道) 没有专门的函数使多项式的向量形式转化为字符形式, 只能通过自定义函数

```
function s=polyprint (p)
%polyprint(p) print p as string
%2020.4.16
if nargin>1 %nargin预定义变量
error('Too much input arguments')
end
if all(p==0) %防止全零矩阵
p=[];
end
pl=length(p);
if pl==0%空p, 来自全零
s='NOT polynomial';
elseif pl==1
s=num2str(p(1));%常数多项式
elseif pl==2
if p(1)==1
```

多项式的加减乘除

以多项式 $p_1 = 3x^3 + x + 1$ 和 $p_2 = 5x^2 - x + 8$ 为例:

一. 加减

```
p1=[3 0 1 1];
p2=[0 5 -1 8]; % 将小幂次增为高幂次.
pplus=p1+p2; % 结果 pplus = 3 5 0 9
pminus= p1-p2; % 结果 pminus = 3 -5 2 -7
```

二. 乘法多项式的乘法对应数列的卷积, 故乘法使用 conv 函数.

```
p1=[3 0 1 1];  p2=[5 -1 8]; % 乘法不需要补齐幂次
pmultiply = conv(p1,p2) % 结果: pmultiply = 15 -3 29 4 7 8
```

三. 除法: 对应去卷积

```
p1=[3 0 1 1];  p2=[5 -1 8]; % 除法同样不需要补齐
[ps,py] = deconv(p1,p2) % 运行结果如下
ps = 0.6000  0.1200 % % 对应商 0.6x+0.12
py = 0  0  -3.6800  0.0400 % 对应余式 -3.6x+0.04
```

多项式的微分和积分

polyder(p): 微分多项式

polyint(p,k): 积分多项式, k 为积分常数, 缺省为 0

```
p = [3  0  -2  0  1  5];
qder = polyder(p)
qint = polyint(p)
% 运行结果
q =
15      0     -6      0      1
qint =
0.5000         0   -0.5000         0    0.5000    5.0000         0
```

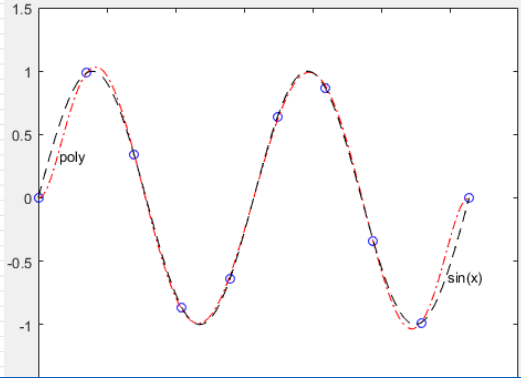
多项式拟合 polyfit

`p = polyfit(x,y,n)`: 用 n 阶多项式拟合数据 x 和 y, 返回多项式系数.

`[p,S] = polyfit(x,y,n)`: 用 n 阶多项式拟合数据 x 和 y, 返回多项式系数 p 和结构数组 S 用于计算拟合误差.

`[p,S,mu] = polyfit(x,y,n)`: 用 n 阶多项式拟合数据 x 和 y, 返回多项式系数 p 和结构数组 S 以及平均值 mu(1) 和标准差 mu(2).

```
x = linspace(0,4*pi,10);
y = sin(x); % 实验数据
plot(x,y,'bo') % 画实验数据
p = polyfit(x,y,7);% 7阶拟合
xx = linspace(0,4*pi);
yy = polyval(p,xx); % 画拟合曲线
yyreal = sin(xx); % 真实曲线
hold on
plot(xx,yy,'r-.',xx,yyreal,'k--')
hold off
gtext('sin(x)') %鼠标输入字符
```



一维插值 interp1

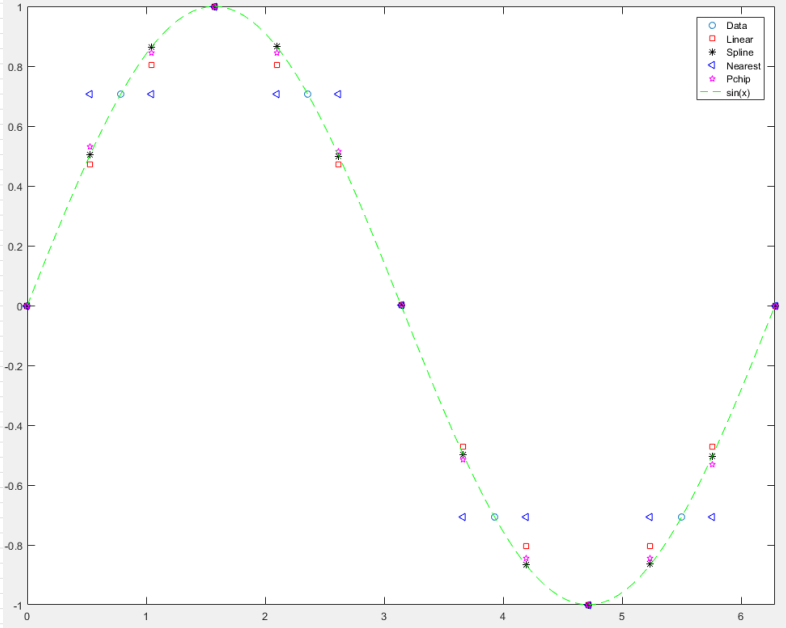
`yq = interp1(x,y,xq,method)`: 根据数据 x 和 y, 使用 method 方法对 xq 插值得到 yq. 方法有:

Method	Description	Continuity	Comments
`'linear'`	线性插值(缺省默认)	光滑连续	2 点. 耗时比 nearest 长
`'nearest'`	就近插值	不光滑连续	2 点. 内存耗费大, 但是计算快
`'next'`	下一个临近插值	不光滑连续	2 点. 与 nearest 一样
`'previous'`	向前插值	不光滑连续	2 点. 与 nearest 一样
`'pchip'`	保形三次插值	比较光滑连续	4 点. 内存和计算大于 linear
`'v5cubic'`	立方卷积插值	比较光滑连续	点均匀. 将来版本被 cubic 代替
`'spline'`	样条插值	很光滑连续	4 个点, 内存和计算大于 pchip
`'cubic'`	与 pchip 一样	比较光滑连续	与 pchip 一样

一般内插是可靠的, 外延插值选项 `'extrap'` 要慎用! `pchip` 和 `spline` 默认可以外延插值 (即可以不使用`'extrap'`). 其他方法若要使用外延必须声明.

interp1 插值方法的区别

```
x = 0:pi/4:2*pi; y = sin(x); %离散数据
plot(x,y,'o','DisplayName','Data');
hold on
xq = 0:pi/6:2*pi; % 欲插值点
% 线性插值
yq1linear = interp1(x,y,xq); plot(xq,yq1linear,'rs','DisplayName','Linear');
% 样条插值
yqspline= interp1(x,y,xq,'spline'); plot(xq,yqspline,'k*','DisplayName','Spline');
% 就近插值
yqnearest= interp1(x,y,xq,'nearest'); plot(xq,yqnearest,'b<','DisplayName','Nearest');
%保形分段三次插值
yqpchip= interp1(x,y,xq,'pchip'); plot(xq,yqpchip,'mp','DisplayName','Pchip');
% 真值
xx=0:pi/50:2*pi; yy=sin(xx);
plot(xx,yy,'g--','DisplayName','sin(x)')
xlim([0 2*pi]); legend('show'), hold off % 图形见下一页...
```


Data
Linear
Spline
Nearest
Pchip
sin(x)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6

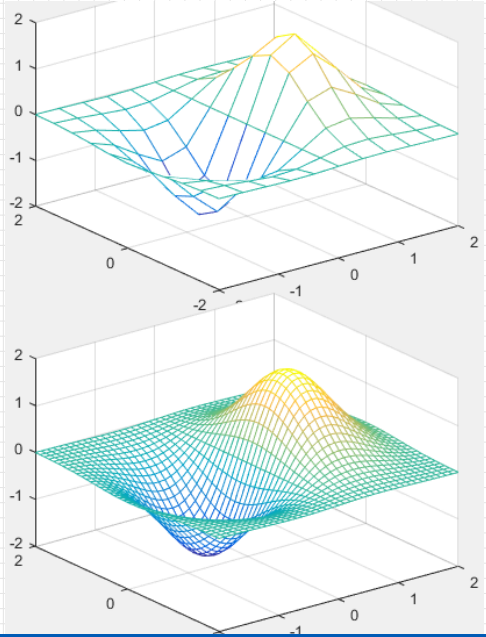
是否默认外延测试代码

```
x = [1 2 3 4 5];
v = [12 16 31 10 6];
plot(x,v,'r-o')

xq = [0 0.5 1.5 5.5 6];
vq1 = interp1(x,v,xq,'pchip');
hold on
plot(xq,vq1,'b-s')

vq2 = interp1(x,v,xq,'linear','extrap');% 带有外延选项
plot(xq,vq2,'g-h')
hold off
```

二维插值 interp2

`Vq = interp2(X,Y,V,Xq,Yq)`: X, Y 为网格数据点, 插值点为 Xq 和 Yq. 二维插值的方法有 `'linear'`, `'nearest'`, `'cubic'` 和 `'spline'`. 默认为 `'linear'`.

```
[x,y] = meshgrid(-2:.4:2);
z=4*x.*exp(-x.^2-y.^2);
subplot(2,1,1)
mesh(x,y,z)
[xq,yq] = meshgrid(-2:.1:2);
zq=interp2(x,y,z,xq,yq,'spline');
subplot(2,1,2)
mesh(xq,yq,zq)
```



三维插值与高维插值

interp3可以用于三维插值

interpn可以用于 n 维插值, n 随已知数据维度而定. 因此, interp 也可以适用于一维, 二维和三维.

NOTE: 插值使用注意事项

1. 内插比外推可靠
2. 数据点越密集, 插值结果越可靠. 当数据密集度达到一定程度时, 每种方法给出的结果几乎没有差别
3. 相对来说 spline, pchip 和 cubic 方法插值更可靠, 因为其插值一个数据需要的数据点要多, 一定程度上更能反映真实物理规律. 但是其可靠度也仅仅是相对而言, 但是其平滑度确实绝对的.
4. 如果对数据背后的物理缺乏基本认识, 线性插值应该是最优也是最可靠的插值方法.

常微分方程的概念

微分方程: 含有未知函数及其导数的关系式. **常微分方程 (Ordinary differential equations, ODE)**: 未知数个数只有一个时, 称之为常微分方程; 多于一个时称之为偏微分方程 (Partial differential equations, PDE).

常微分方程的求解分为**初值问题** (Initial value problem, IVP) 和**边值问题** (Boundary value problem, BVP), 后者较难. MATLAB ODE 解算器:

Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with **crude tolerances**, or in the presence of **moderate stiffness**.
ode113		Low-High	ode113 can be more efficient than ode45 at problems with **stringent error tolerances**, or when the ODE function is expensive to evaluate.
ode15s	Stiff	Low-Medium	Try ode15s when ode45 fails or is inefficient and you suspect that the problem is stiff. Also use ode15s when solving differential algebraic equations (DAEs).
ode23s		Low	ode23s can be more efficient than ode15s at problems with crude error tolerances. It can solve some stiff problems for which ode15s is not effective. ode23s computes the Jacobian in each step, so it is beneficial to provide the Jacobian via odeset to maximize efficiency and accuracy. If there is a mass matrix, it must be constant.
ode23t		Low	Use ode23t if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve differential algebraic equations (DAEs).
ode23tb		Low	Like ode23s, the ode23tb solver might be more efficient than ode15s at problems with crude error tolerances.
ode15i	Fully implicit	Low	Use ode15i for fully implicit problems $f(t,y,y') = 0$ and for differential algebraic equations (DAEs) of index 1.

一阶非刚性微分方程 ode45

ode45 是最常用且首选的解常微分方程的解算器, 其采用的四五阶龙格库塔法. M 语法

[t,y] = ode45(odefun,tspan,y0): 解微分方程, 初值为 y0. tspan 可以是区间, 也可以是向量. 当 tspan 是区间时, 在给定区间内求解; 当是向量时, 计算给定值时的解.

例一: 微分方程 $\frac{\mathrm{d}y}{\mathrm{d}t} = 2t$, 初值 $y(t=0) = 0$. 根据微积分的知识其解为 $y = t^2$.

M 数值解

```
tspan = [0 5]; % 计算区间为 [0,5]
y0 = 0; % 初值为 0
[t,y] = ode45(@(t,y) 2*t, tspan, y0); % 匿名函数, t和y都是列向量.
plot(t,y,'b-o')
```

例二: $\frac{\mathrm{d}y}{\mathrm{d}t} = \frac{y}{100}\left[1+\sin\left(\frac{2\pi}{365}t\right)\right]$, 初值 $y(t=0)=2$, 计算区间为 $[0,365]$;

```
a = 0.01; omega = 2*pi/365;
tspan = [0,365]; y0=2;
funhandle = @(t,y) a*y*(1+sin(omega*t));
[t,y] = ode45(funhandle, tspan, y0); % 函数句柄
plot(t,y,'b.-')
```

例三: $a\frac{\mathrm{d}y}{\mathrm{d}t} + y = \sin(2\pi t)$, 其中 $a = \frac{1}{2\pi}$, 初值 $y(t=0)=0$

```
omega=2*pi;
tspan = [0,6];
y0=0;
funhandle = @(t,y) (sin(omega*t)-y)*omega;
[t,y] = ode45(funhandle, tspan, y0);
plot(t,y,'b.-')
```

一般步骤:

1. 将导数, 即 $\mathrm{d}y/\mathrm{d}t$ 单独放在一侧
2. 定义导数函数
3. ode45 求解.

高阶非刚性常微分方程 ode45

ode45 只能处理一阶微分方程, 因此要解高阶微分方程, 需要将其化成一阶微分方程组. 例如有 n 阶微分方程

$$f(y, y^{(1)}, y^{(2)}, \cdots, y^{(n)}) = 0$$

初值条件需要 n 个.

高阶化一阶微分方程组

1. 设 $y1 = y, y2 = y^{(1)}, y3 = y^{(2)}, \cdots, yn = y^{(n-1)}$.
2. 故 $y^{(n)} = \frac{\mathrm{d}yn}{\mathrm{d}t}$ 一阶微分方程, 以此类推...
3. 写出微分方程组 (共 n 个微分方程)...
4. ode45 求解

高阶非刚性常微分方程 ode45

例一: van der Pol 微分方程:

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} - \mu(1-x^2)\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0$$

设 $\mu = 1$ (μ 很大时为刚性, 后讲), 初始时刻 $x(t=0) = 2\,, x'(t=0) = 0$

其为二阶微分方程, 所以写成两个微分方程.

设 $x = x_1\,, x' = x_2$, 故上述二阶微分方程化为

$$\begin{cases} \dfrac{\mathrm{d}x_1}{\mathrm{d}t} = x_2\,; \\ \dfrac{\mathrm{d}x_2}{\mathrm{d}t} = \mu(1-x_1^2)x_2 - x_1\,. \end{cases}$$

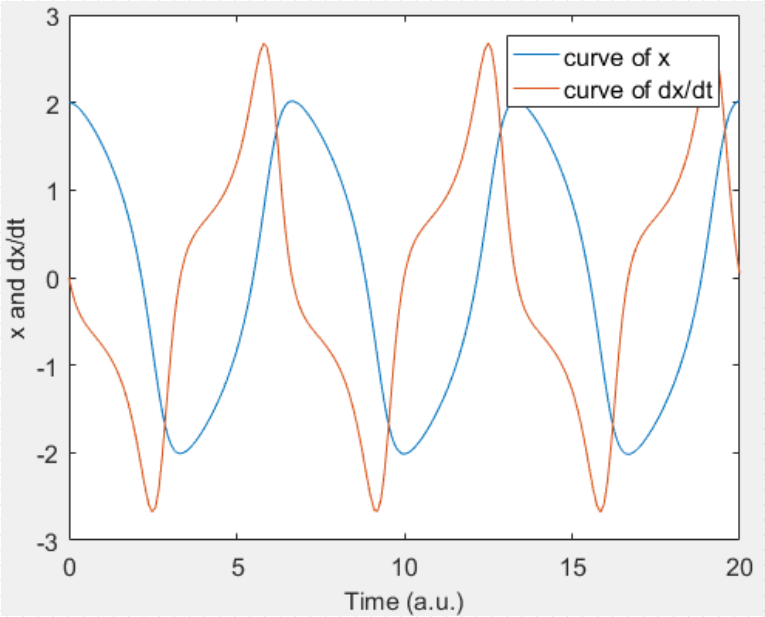
代码下一页

```
function onlyForTest
tspan=[0,20];
x0=[2;0]; % 初始条件
[tt,X] = ode45(@vdp,tspan,x0);
plot(tt,X(:,1)); hold on
plot(tt,X(:,2)); hold off
lgd=legend('curve of x','curve of dx/dt');
lgd.FontSize=12; % 图注相关
set(gca,'FontSize',12) % 坐标轴相关
xlabel('Time (a.u.)','FontSize',12)
ylabel('x and dx/dt','FontSize',12)
    function dxdt = vdp(t,x) %
        x1=x(1);x2=x(2);
        mu=1;
        dx1dt=x2; dx2dt=mu*(1-x1.^2).*x2-x1;
        dxdt=[dx1dt;dx2dt];
    end % 可读性高才这么写，建议如此写
end
```

1. 为了放在一个文件里，才都写成嵌套函数形式... 所以母函数没有输入和输出
2. 可以将微分方程写成一个函数，然后求解过程写成 M 脚本
3. 关于微分方程函数，可以采用别的写法：直接写成向量的形式即

`[x(2);(1-x(1).^2).*x(2)-x(1);]`

当阶数更高时不是很方便且可读性差.


curve of x
curve of dx/dt
x and dx/dt
Time (a.u.)
3
2
1
0
-1
-2
-3
0
5
10
15
20

高阶非刚性常微分方程 ode45

例二: Lotka-Voltera 竞争模型: 生态系统中, 具有相似要求的物种, 为了争夺空间和资源, 而产生的一种直接或间接抑制对方的现象. 以狐狸 (Fox) 和兔子 (Rabbit) 为例:

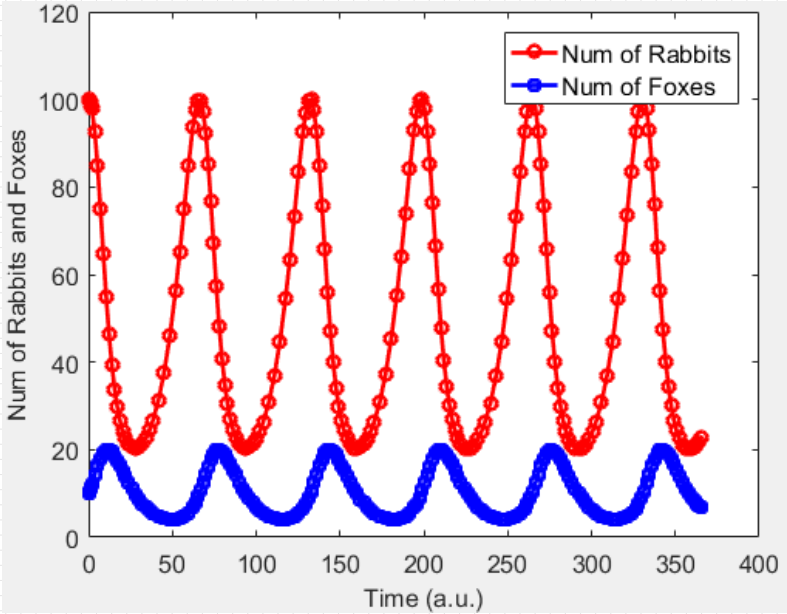
$$\begin{cases} \dfrac{\mathrm{d}R}{\mathrm{d}t} = aR - bRF \\ \dfrac{\mathrm{d}F}{\mathrm{d}t} = ebRF - cF \end{cases}$$

- R, F 分别是兔子和狐狸的数量
- a 表示在没有狐狸的情况下, 兔子的自然生产率
- c 表示没有兔子的情况下, 狐狸的自然死亡率
- b 表示狐狸与兔子作用时的死亡率
- e 表示死亡的兔子转化为狐狸的概率.

设: $a = 0.1\,, \quad b = 0.01\,, \quad c = 0.1\,, \quad e = 0.2$

代码下一页

```
function forTest
tspan=[0,365];
x0=[100;10];
[tt,X] = ode45(@lotka,tspan,x0);
plot(tt,X(:,1),'LineWidth',2);
hold on
plot(tt,X(:,2),'LineWidth',2)
hold off
lgd=legend('Num of Rabbits','Num of Foxes');
lgd.FontSize=12;
set(gca,'FontSize',12)
xlabel('Time (a.u.)','FontSize',12)
ylabel('Num of Rabbits and Foxes','FontSize',12)
    function res = lotka( ,RF) % 不知为何波浪号 不显示
        a=0.1; b=0.01; c=0.1; e=0.2;
        R=RF(1); F=RF(2);
        dRdt=a*R-b*R*F; dFdt=e*b*R*F-c*F;
        res=[dRdt;dFdt];
    end
end
```


Num of Rabbits
Num of Foxes
Num of Rabbits and Foxes
Time (a.u.)
0
20
40
60
80
100
120
0
50
100
150
200
250
300
350
400

刚性微分方程 ode15s

ode15s 作为刚性方程的首选解算器. (有的变化快有的变化慢两者不在一个量级). 调用方式与 ode45 一样:

```
[t,y] = ode15s(odefun,tspan,y0)
```

仍以 van der Pol 微分方程:

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} - \mu(1-x^2)\frac{\mathrm{d}x}{\mathrm{d}t} + x = 0$$

只是这里 $\mu = 1000$, 初始时刻 $x(t=0) = 2\,, x'(t=0) = 0$, 计算区间为 $[0\,, 3000]$

用 ode45 和 ode15s 求解看区别

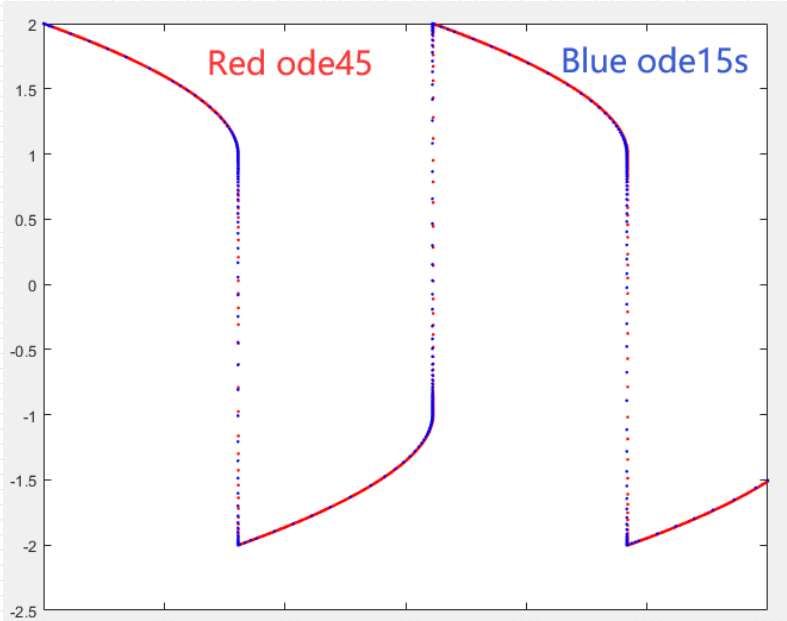
```
function onlyForTest
% Using ode45
tic % 给出计算时间
tspan=[0,3000];
x0=[2;0];
[tt,X] = ode45(@vdp,tspan,x0);
plot(tt,X(:,1),'r.');
toc % 给出计算时间
function dxdt = vdp(t,x) %
x1=x(1);x2=x(2);
mu=1000;
dx1dt=x2;
dx2dt=mu*(1-x1.^2).*x2-x1;
dxdt=[dx1dt;dx2dt];
end
end
```

Elapsed time is 84.388189 seconds.

```
function onlyForTest
% using ode15s
tic % 给出计算时间
tspan=[0,3000];
x0=[2;0];
[tt,X] = ode15s(@vdp,tspan,x0);
plot(tt,X(:,1),'b.');
toc % 给出计算时间
function dxdt = vdp(t,x) %
x1=x(1);x2=x(2);
mu=1000;
dx1dt=x2;
dx2dt=mu*(1-x1.^2).*x2-x1;
dxdt=[dx1dt;dx2dt];
end
end
```

Elapsed time is 0.160132 seconds.

精度上没有太大的差异...


Red ode45
Blue ode15s
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5

导数方程分段

当导数方程比较复杂时, 可以在定义函数时说明. 例如

$$\begin{cases} \dfrac{\mathrm{d}y_1}{\mathrm{d}t} = y_2 - f(t) \\ \dfrac{\mathrm{d}y_2}{\mathrm{d}t} = y_1 g(t) - y_2 \end{cases}$$

$$f(t) = \begin{cases} 2\sin(t)\,, & t < 4\pi \\ 0\,, & t \geq 4\pi \end{cases} \qquad g(t) = \begin{cases} 0\,, & t < 3.5\pi \\ \cos(t)\,, & t \geq 3.5\pi \end{cases}$$

导数函数的定义

```
function res = dydt(t,y)
ft = 0;gt = 0;
if t<4*pi % 采用条件语句定义导数函数
   ft = 2*sin(t);
end
if t>3.5*pi
   gt = cos(t);
end
res = [y(2)-ft;y(1)*gt-y(2)];
end
```

整个文件如下: 并利用 deval 及 fzero 函数求算了 $y_1 + y_2 = 0$ 的零点.

```
function T0 = ComplexODESolve
% 太长放不下, 两页也没意思, 参看: ComplexODESolve.m 文件或<MATLAB 25 个高效...>的第十章
tspan = [0,20];y0 = [1,2];%区间与初值
sol = ode23tb(@dydt,tspan,y0);%结构体sol
subplot(1,2,1); plot(sol.x,sol.y(1,:),'k-','linewidth',2);%NOTE y(1,: NOT y(:,1))
hold on; plot(sol.x, sol.y(2,:),'k--','linewidth',2);%
```

隐函数微分方程

ode45 以及 ode15s 求解微分方程时, 必须将导数函数写成显含数的形式, 即 $\mathrm{d}y/\mathrm{d}t$ 在等号一侧, 另一侧不能存在 $\mathrm{d}y/\mathrm{d}t$. 对于隐函数, 它们无法直接求解, 可以使用专门的求解器 ode15i.

隐函数微分方程:

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \exp\left\{-\left\{\left[y - 0.5 - \exp\left(-t + \frac{\mathrm{d}y}{\mathrm{d}t}\right)\right]^2 + y^2 - \frac{t}{5} + 3\right\}\right\}$$

初值 $y(t=0) = 0.1$, 求解区间为 $[0\,,20]$ (无法写成显含数形式).

ode15i 的基本调用格式: `[t,y] = ode15i(odefun,tspan,y0,yp0)`

- odefun 形式 $f(t\,,y\,,y') = 0$.
- tspan: 原含义 (积分区间或者积分序列).
- y0: y 的初值 (已知条件).
- yp0: $\mathrm{d}y/\mathrm{d}t$ 的(试探) 初值, 尽量满足 $f(t_0\,,y_0\,,yp_0) = 0$ (可以用 decic 函数求算, 简单的也可以用 fzero 函数求解.).

上一页隐函数 M 代码

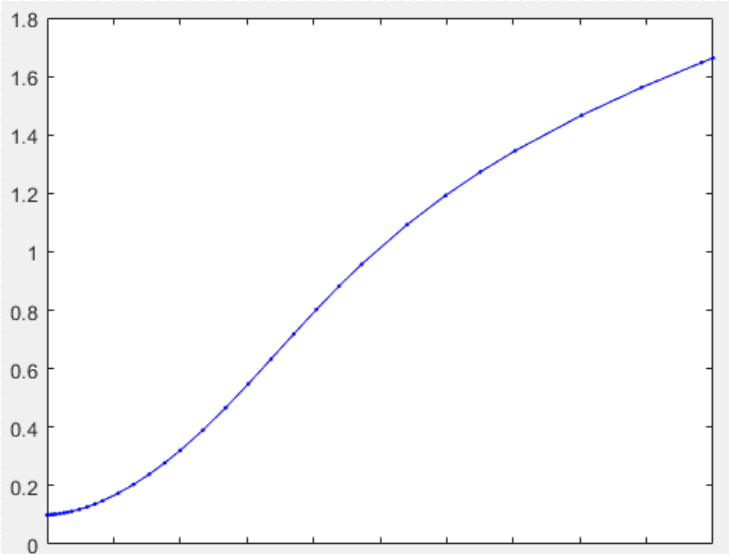
一. fzero 求 yp0

```
ti=0;   tf=20;
y0=0.1; % 初始条件
f=@(t,y,yp) yp-exp(-((y-.5-exp(-t+yp)).^2+y.^2-t/5+3)); % 微分方程改写为 f(t,y,y') = ...
    0 的形式
yp0 = fzero(@(yp)f(t0,y0,yp),1); % fzero 求yp0的初值.
[tt,yy]=ode15i(f,[ti,ff],y0,yp0); % ode15i 求解
plot(tt,yy,'r.-')
```

二. decic 函数求 yp0

```
ti=0; tf=20
y0=0.1; y0fix = y0; yp0guess=1; % 猜测的yp初值
f=@(t,y,yp) yp-exp(-((y-.5-exp(-t+yp)).^2+y.^2-t/5+3));
[y0,yp0] = decic(f,ti,y0,y0fix,yp0guess,[]); % doc decic
%fres=f(t0,y0,0.0333) % 用来确定求得的yp0是否使f接近0
[tt,yy]=ode15i(f,[ti,tf],y0,yp0);
plot(tt,yy,'b.-')
```

上一隐函数微分方程求解结果图示



微分方程求解练习—RLC 电路

RLC 电路中, 电容两端的电压满足

$$LC\frac{\mathrm{d}^2V_c}{\mathrm{d}t^2}+RC\frac{\mathrm{d}V_c}{\mathrm{d}t}+V_c=V_s$$

考虑比较简单的情况: 设 $LC=1\,,RC=3\,,V_s=\sin(t)$. 初始条件 $V_c(t=0)=V_c'(t=0)=0$. (根据微积分的知识, 此时可以解析求解)

M 代码求解 (符号求解和数值求解)

```
ti=0; tf=50; y0=0; yp0=0; % 区间及初始条件
% 符号求解
syms y(t)
Dy1=diff(y,t);Dy2=diff(y,t,2); % 一二阶微商
eqn=Dy2+3*Dy1+y==sin(t);% 符号表达式
yExactRes=dsolve(eqn,y(0)==0,Dy1(0)==0); % 符号解算器 dsolve
yhandle=matlabFunction(yExactRes);% matlabFunction 符号->函数句柄
%数值求解
[tt,yy] = ode45(@dydt,[ti,tf],[y0;yp0]);
plot(tt,yy(:,1),'b.-','DisplayName','Num. Res');% 数值
hold on; tseq=linspace(0,50,1000); yseq=yhandle(tseq);
plot(tseq,yseq,'r.-','DisplayName','Sym. Res')
lgd=legend('show'); lgd.FontSize=12; hold off
% 下面微分函数, 可以直接在脚本中定义...
function res = dydt(t,yn)
y1=yn(1); y2=yn(2); dy1dt=y2; dy2dt=sin(t)-y1-3*y2; res=[dy1dt;dy2dt];
end
```

数值与符号求解结果图示比较