Matlab 中常微分方程数值解法讲解©

作者：dynamic

时间：2008.12.10

1.ODE 解算器简介

先来认识下常微分方程(ODE)初值问题解算器(solver)

[T,Y,T*E*,*YE*,*IE*] = odesolver(odefun,tspan,y0,*options*)
sxint = deval(sol,xint)

解算器(odesolver)

解算器	问题类型	精确度	说明
ode45	非 刚 性 (nonstiff)	中等	4-5 阶龙格库塔，对以所有问题的首先解算器
ode23	非刚性	低	基于 Bogacki-Shampine 2-3 阶 Runge-Kutta 公式，有时对轻度的刚度方程，它可以比 ode45 有更好的效果，在相同的精度是，要比 ode45 更小的步长
ode113	非刚性	低到高	变阶次 Adams-Bashforth-Moutlon 算法，此算法使用前几次节点上的值来计算当前节点处的解，因此在相同 jingdu 下，比 ode45 和 ode23 更快些，适用用于高阶或者需要大量计算的问题，但不适合不连续的系统
ode15s	刚性(stiff)	低到中	刚性系统的变阶次多步解法，此算法使用最新的数值差分公式，如果使用 ode45 计算比较慢，可以使用它试试
ode23s	刚性	低	刚性系统固定阶次的单步解法，由于它是单步方法，故有时要比 ode15s 速度快些，对具体问题，到底哪个好些，大家可以同时尝试下
ode23t	中等刚性	低	
ode23tb	刚性	低	

相关优化选项

参数	ode45	ode23	ode113	ode15s	ode23s	ode23t	ode23tb
relto abstol normcontrol	√	√	√	√	√	√	√
outputfcn outputsel refine stats	√	√	√	√	√	√	√
nonnegative	√	√	√	√*	×	√*	√*
events	√	√	√	√	√	√	√
maxstep inialstep	√	√	√	√	√	√	√
jacobian jpattern vectorized	×	×	×	√	√	√	√
mass mstatedependence	√ √	√ √	√ √	√ √	√ ×	√ √	√ √

mvpattern masssingular	× ×	× ×	× ×	√ √	× ×	√ √	√ ×
initialslope	×	×	×	√	×	√	×
maxorder bdf	×	×	×	√	×	×	×

注：√*表示只能应用于没有质量矩阵的问题

输入参数：

odefun：微分方程的句柄，必须写成 Matlab 规范格式，这个具体在后面讲解

tspab=[t0 tf]或者[t0,t1,…tf]：微分变量的范围，两者都是根据 t0 和 tf 的值自动选择步长，后只是前者返回所有计算点的微分值，而后者只返回指定的点的微分值，一定要注意对于后者 tspan 必须严格单调，还有就是两者数据存储时使用的内存不同(明显前者多)，其它没有任何本质的区别

y0=[y(0),y’(0),y’’(0)…]：微分方程初值，依次输入所有状态变量的初值

options：微分优化参数，是一个结构体，使用 odeset 可以设置具体参数，详细内容查看帮助

输出参数：

T：时间列向量

Y：二维数组，第 i 列表示第 i 个状态变量的值， 行数与 T 一致

在求解 ODE 时，我们还会用到 deval()函数，deval 的作用就是通过结构体 solution 计算 t 对应 x 值，和 polyval 之类的很相似

输入参数：

sol：就是上次调用 ode**函数得道的结构体解

xint：需要计算的点，可以是标量或者向量，但是必须在 tspan 范围内

该函数的好处就是如果我想知道 t=t0 时的 y 值，不需要重新使用 ode 计算，而直接使用上次计算的得道 solution 就可以

2.微分方程转换

好，上面我们把 Matlab 中的常微分方程(ODE)的解算器讲解的差不多了，下面我们就具体开始介绍如何使用上面的知识吧！

现实总是残酷的，要得到就必须先付出，不可能所有的 ODE 一拿来就可以直接使用，因此，在使用 ODE 解算器之前，我们需要做的第一步，也是最重要的一步就是将微分方程组化成 Matlab 可接受的标准形式！

如果 ODEs 由一个或多个高阶微分方程给出，则我们应先将其变换成**一阶显式常微分方程组**！

下面我们以两个高阶微分方程构成的 ODEs 为例介绍如何将之变换成一个**一阶显式常微分方程组**。

step1.将微分方程的最高阶变量移到等式的左边，其他移到右边，并按阶次从低到高排列，假如说两个高阶微分方程最后能够显式的表达成如下所示：

$$\begin{cases} x^{(m)} = f(t, x, x', x'', \cdots, x^{(m-1)}, y, y', \cdots, y^{(n-1)}) \\ y^{(n)} = g(t, x, x', x'', \cdots, x^{(m-1)}, y, y', \cdots, y^{(n-1)}) \end{cases}$$

然而现实总是残酷的，有时方程偏偏是隐式的，没法写成上面的样子，不用担心 Matlab 早就为我们想到了，这个留在后面的隐式微分方程数值求解中再详细讲解！

step2.为每一阶微分式选择状态变量，最高阶除外

$$x_1 = x,\ x_2 = x',\ x_3 = x'', \cdots,\ x_m = x^{(m-1)}$$
$$x_{m+1} = y,\ x_{m+1} = y',\ x_{m+2} = y'', \cdots,\ x_{m+n} = y^{(n-1)}$$

从上面的变换，我们注意到，ODEs 中所有因变量的最高阶次之和就是需要的状态变量的个数，最高阶的微分式(比如上面的 $x^{-(m)}$和 $y^{(n)}$)不需要给它一个状态变量

step3.根据上面选用的状态变量，写出所有状态变量的一阶微分的表达式

$$
\begin{aligned}
x'_1 &= x_2 \\
x'_2 &= x_3 \\
x'_3 &= x_4 \\
&\vdots \\
x'_m &= f(t, x_1, x_2, x_3, \cdots, x_{m+n}) \\
x'_{m+1} &= x_{m+2} \\
&\vdots \\
x'_{m+n} &= g(t, x_1, x_2, x_3, \cdots, x_{m+n})
\end{aligned}
$$

注意到，最高阶次的微分式的表达式直接就是 step1 中的微分方程

好，到此为止，一阶显式常微分方程组，变化顺利结束，接下来就是 Matlab 编程了。**请记住上面的变化很重要，Matla 中所有微分方程的求解都需要上面的变换**

下面通过一个实例演示 ODEs 的转换和求解

已知Apoll卫星的运动轨迹(x,y)满足下面的微分方程组

$$
\begin{cases}
x'' = 2y' + x - \dfrac{m^*(x+m)}{r_1^3} - \dfrac{m(x-m^*)}{r_2^3} \\
y'' = -2x' + y - \dfrac{m^* y}{r_1^3} - \dfrac{m y}{r_2^3}
\end{cases}
$$

其中：

$$
r_1 = \sqrt{(x+m)^2 + y^2}, r_2 = \sqrt{(x-m^*)^2 + y^2}, m^* = 1 - m, m = 1/82.45
$$

$$
x(0) = 1.2, x'(0) = 0, y(0) = 0, y'(0) = -1.04935751
$$

【解】 真是万幸，该 ODEs 已经帮我们完成了 step1，我们只需要完成 step2 和 step3 了

(1)选择一组状态变量

$$
x_1 = x,\ \ x_2 = x',\ \ x_3 = y,\ \ x_4 = y'
$$

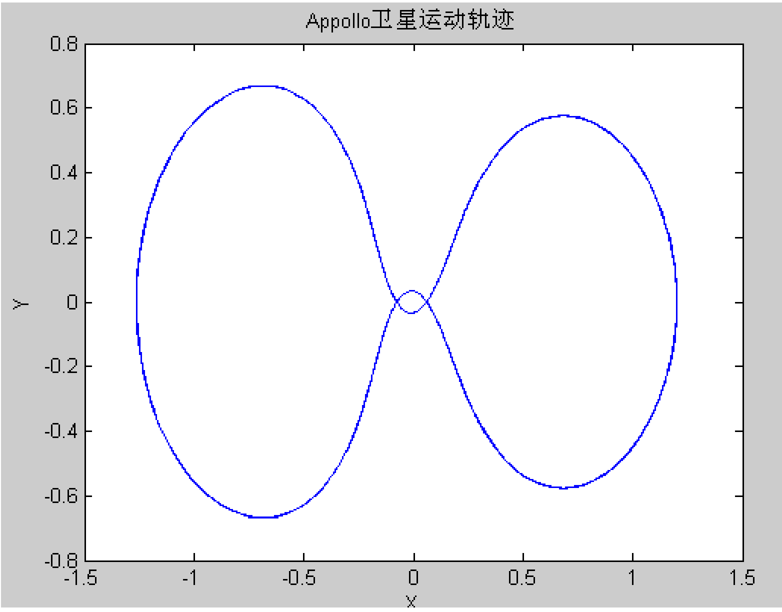
(2)写出所有状态变量的一阶微分表达式

$$x_1' = x_2$$
$$x_2' = 2x_4 + x_1 - m^*(x_1 + m)/r_1^3 - m(x_1 - m^*)/r_2^3$$
$$x_3' = x_4$$
$$x_4' = -2x_2 + x_3 - m^* x_3 / r_1^3 - m x_3 / r_2^3$$

(4)有了数学模型描述，则可以立即写出相应的 Matlab 代码了(这里我们优先选择 ode45)

```
x0=[1.2;0;0;-1.04935751];%x0(i)对应与 xi 的初值
options=odeset('reltol',1e-8);%该命令的另一种写法是 options=odeset;options.reltol=1e-8;
tic
[t,y]=ode45(@appollo,[0,20],x0,options);%t 是时间点，y 的第 i 列对应 xi 的值，t 和 y 的行数相同
toc
plot(y(:,1),y(:,3))%绘制 x1 和 x3，也就是 x 和 y 的图形
title('Appollo 卫星运动轨迹')
xlabel('X')
ylabel('Y')

function dx=appollo(t,x)
mu=1/82.45;
mustar=1-mu;
r1=sqrt((x(1)+mu)^2+x(3)^2);
r2=sqrt((x(1)-mustar)^2+x(3)^2);
dx=[x(2)
    2*x(4)+x(1)-mustar*(x(1)+mu)/r1^3-mu*(x(1)-mustar)/r2^3
    x(4)
-2*x(2)+x(3)-mustar*x(3)/r1^3-mu*x(3)/r2^3];
```



3.刚性/非刚性问题

在工程实践中，我们经常遇到一些 ODEs，其中某些解变换缓慢，另一些变化很快，且相差悬殊的微分方程，这就是所谓的刚性问题(Stiff)，对于所有解的变化相当我们则称为非刚性问题(Nonstiff)。
对于刚性问题一般不适合使用 ode45 这类函数求解。

由于非刚性问题我们使用的多比较多，我们就不多说，下面主要讲解下 Stiff

看一个例子，考虑下面的微分方程

$$\begin{cases} y_1' = 0.04(1-y_1)-(1-y_2)y_1+0.0001(1-y_2)^2 \\ y_2' = -10000y_1+3000(1-y_2)^2 \end{cases}$$

其中

$$y_2(0)=y_1(0)=1 \quad t\in[0,100]$$

【解】题目中已经帮我们完成了方程组转换，下面我们就直接编程了

(1)编写微分方程函数

```
odefun=@(t,x)[0.04*(1-x(1))-(1-x(2))*x(1)+0.0001*(1-x(2))^2
    -1e4*x(1)+3000*(1-x(2))^2];
x0=[0 1];
tspan=[0 100];
options=odeset('reltol',1e-6,'abstol',1e-8);
```

(2)使用 ode45 函数计算(反正我们是没有信心等待它，太慢了)

```
tic;[t,y]=ode45(odefun,tspan,x0,options);toc
disp(['ode45 计算的点数' num2str(length(t))])
```

(3)使用 ode15s 函数计算

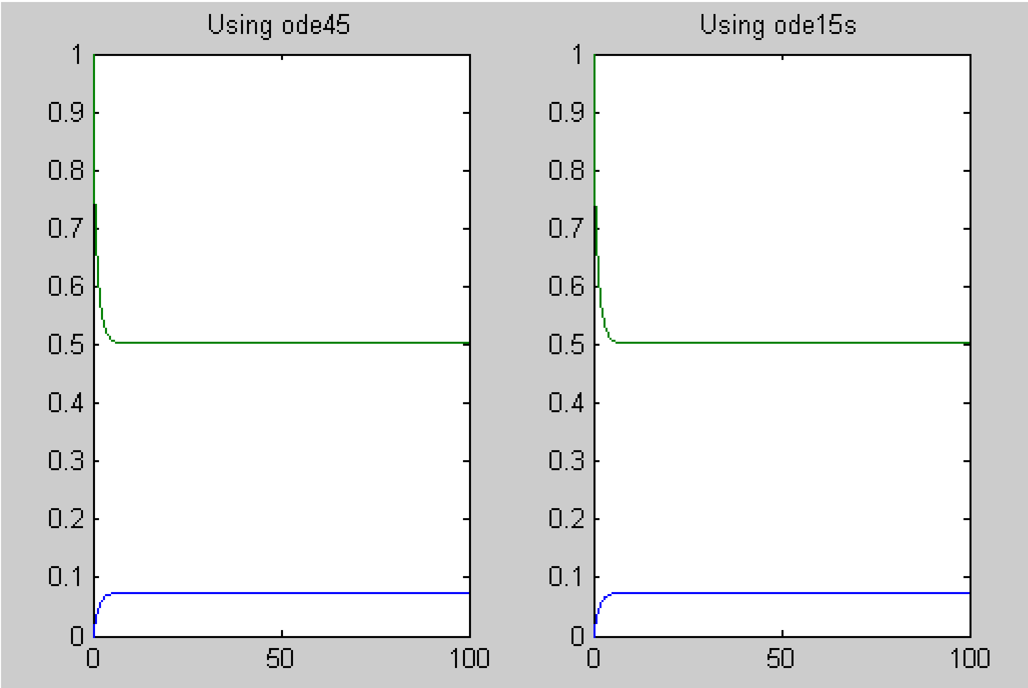
```
tic;[t2,y2]=ode15s(odefun,tspan,x0,options);toc
disp(['ode15s 计算的点数' num2str(length(t2))])
```

(4)绘图看看

```
figure('numbertitle','off','name','Stiff ODEs Demo—by Matlabsky')
subplot(121)
title('Using ode45')
plot(t,y)
subplot(122)
plot(t2,y2)
title('Using ode15s')
```

下面是运行结果：

Elapsed time is 171.005688 seconds.
ode45 计算的点数 356981
Elapsed time is 0.397287 seconds.
ode15s 计算的点数 188



4.隐式微分方程(IDE)

上帝不会总是那么仁慈的，不是所有的 ODEs 都是可以直接显式的表达成下面的样子

$$\begin{cases} x^{(m)} = f(t, x, x', x'', \mathbf{L}, x^{(m-1)}, y, y', \mathbf{L}, y^{(n-1)}) \\ y^{(n)} = g(t, x, x', x'', \mathbf{L}, x^{(m-1)}, y, y', \mathbf{L}, y^{(n-1)}) \end{cases}$$

比如，下面的隐式微分方程组

$$\begin{cases} x''+2y'x=2y'' \\ x''y'+3x'y''+xy'-y=5 \end{cases}$$

那该如何办呢？

我们这里有三种解决方法：

方法一

使用符号计算函数 solve()求解出 x”和 y”的表达式，不就可以了吗？那好，我们下面试试看看

【解】

(1)求解表达式

```
>> [d2x,d2y]=solve('d2x+2*dy*x-2*d2y','d2x*dy+3*dx*d2y+x*dy-y-5','d2x','d2y')%d2x 表示 x 的二阶导数，其它同理

d2x =

-2*(3*dy*x*dx-5+dy*x-y)/(3*dx+2*dy)

d2y =

(2*dy^2*x+5-dy*x+y)/(3*dx+2*dy)
```

也就是说原来的 ODEs 可以重新写成

$$\begin{cases} x''=-2*\dfrac{(3y'xx'-5+y'-y)}{3x'+2y'} \\ y''=\dfrac{2(y')^2x+5-y'x+y}{3x'+2y'} \end{cases}$$

也就是说 step1 完成了，接下来的就比较简单了

(2).选取状态变量

$x_1 = x,\ x_2 = x',\ x_3 = y,\ x_4 = y'$

(3).写出状态变量的一阶微分表达式

$$\begin{cases} x_1' = x' = x_2 \\ x_2' = x'' = -2 * \dfrac{(3x_4 x_1 x_2 - 5 + x_4 - x_3)}{3x_2 + 2x_4} \\ x_3' = y' = x_4 \\ y_4' = y'' = \dfrac{2(x_4)^2 x_1 + 5 - x_4 x_1 + x_3}{3x_2 + 2x_4} \end{cases}$$

(4).写 Matlab 程序进行数值求解

```
myode=@(t,x)[x(2)
    -2*(3*x(1)*x(2)*x(4)-5+x(4)-x(3))/(3*x(2)+2*x(4))
    x(4)
    (2*x(4)^2*x(1)+5-x(4)*x(1)+x(3))/(3*x(2)+2*x(4))]
[t,y]=ode45(myode,[0,20],x0);
```

方法二

但是 solve()函数也不是万能的，对有些完全隐式，solve()可能压根没法求解出它们的具体表达式，那此时该怎么办呢？天无绝人之路，车到山前必有路，下面我们分析下数值解法的本质。

其实 Matlab 要求我们先将 ODEs 转化为显式的一阶微分方程组，就是为了便于计算状态变量一阶微分函数值(注意只需要它的值)！虽然现在我们没有解析的得到各个状态变量一阶微分的表达式，但是我们只要求出每个点的对应的所有状态变量一阶微分值即可。

这个思想就是在 Matlab 没有提供 ode15i()函数之前，大家唯一能使用的方法，下面我们以一个例子说明下

$$\begin{cases} x''\sin(y') + (y'')^2 = -2xy + xx''y' \\ xx''y'' + \cos(y'') = 3yx' \end{cases}$$

其中

$x0 = [1;0;0;1]$

对于这样的 ODEs 难道你还想将它化为那个一阶显式微分方程组吗，别费劲了！

【解】

(1)好，我们同样的需要一组状态变量

$x_1 = x,\ x_2 = x',\ x_3 = y,\ x_4 = y'$

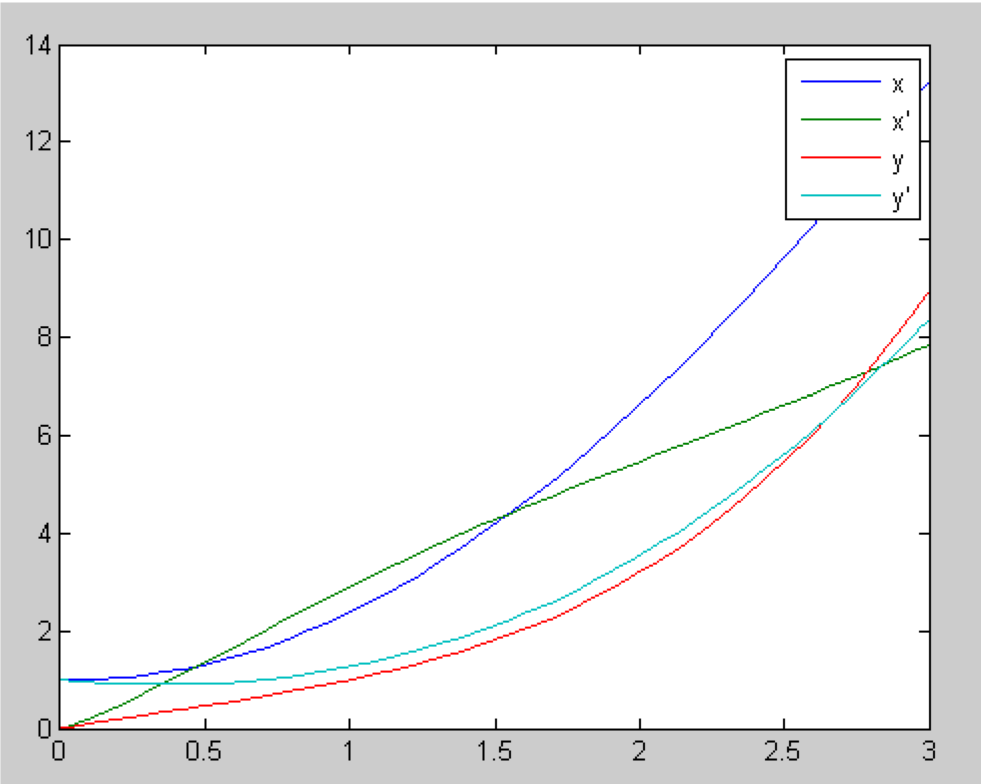
(2).写出状态变量的一阶微分。只不过此时的 x”和 y”不再是显式的，我们使用 fsolve 进行数值求解

$$\begin{cases} x_1' = x_2 \\ x_2'\sin(x_4) + (x_4')^2 = -2x_1x_3 + x_1x_2'x_4 \\ x_3' = x_4 \\ x_1x_2'x_4' + \cos(x_4') = 3x_3x_2 \end{cases}$$

(3)编写 Matlab 代码

```
[t,y]=ode45(@odefun,[0 3],[1 0 0 1]');
plot(t,y)
legend('x','x''','y','y''')

function dy=odefun(t,x)
fun=@(dxy)[dxy(1)*sin(x(4))+dxy(2)^2+2*x(1)*x(3)-x(1)*dxy(1)*x(4)
    x(1)*dxy(1)*dxy(2)+cos(dxy(2))-3*x(3)*x(2)];
options=optimset('display','off');
y=fsolve(fun,x([1,3]),options);%使用fsolve求解出x''和y''
dy=[x(2);y(1);x(4);y(2)];%状态变量一阶微分值
```



方法三

方法二也有它的缺点，每一个点的 x"和 y"都需要独立计算，假如说数据量很大的话，那个叫难熬呀，并且有时可能是由于方程或者参数配置问题 fsolve 还有会出现罢工现象！

于是 MathWorks 为我们准备了 ode15i 函数，不过这个 ode15i 有点不好用，没有 ode45 等那么直接。

ode15i 调用格式

[y0mod,yp0mod,*resnrm*] = decic(odefun,t0,y0,fixed_y0,yp0,fixed_yp0, *options*...)
[T,Y,TE,*YE*,*IE*] = ode15i(odefun,tspan,y0mod,yp0mod,*options*...)

隐式 ODEs 不同于一般的显式 ODEs，**ode15i 需要给出状态变量及其一阶导数的初值(x0,dx0)**，它们不能任意赋值，只能有 n 个是独立的，其余的需要隐式方程求解(使用 decic 函数)，否则出现矛盾的初值条件

输入参数：

odefun：微分方程，注意此时的函数变量有 t(自变量)，x(状态变量)，dx(状态变量一阶导数)等三个
t0：自变量初值，必须与 tspan(1)相等
y0：状态变量初值，题目给出几个就写几个，没给的自己猜测一个
fixed_y0：与 y0 的尺寸一样，表示对应位置的初值是给定的还是猜测的，如果给定则 1，否则 0
yp0：状态变量一阶导数初值，其它同 y0
fixed_yp0：同理 fixed_y0，不过此时对应的是 yp0 了
options：其它选项，具体查看帮助
tspan：自变量的范围，其中 tspan(1)必须与 t0 一致

输出参数：

大部分同理 ode45

下面我们以实例说明，对于下面的 IDE，求解 x

$$\begin{cases} x''\sin(y')+(y'')^2=-2xy+xx''y' \\ xx''y''+\cos(y'')=3yx' \end{cases}$$

其中

$$x0=[1;0;0;1]$$

【解】

(1)选取状态变量

$$x_1=x,\ x_2=x',\ x_3=y,\ x_4=y'$$

(2)写出状态变量的一阶微分。只不过此时的 x”和 y”不再是显式的

$$\begin{cases} x_1' = x_2 \\ x_2'\sin(x_4) + (x_4')^2 = -2x_1x_3 + x_1x_2'x_4 \\ x_3' = x_4 \\ x_1x_2'x_4' + \cos(x_4') = 3x_3x_2 \end{cases}$$

(3)书写 odefun，注意此时多了一个 dx 变量，就是 x 的一阶导数

```
odefun=@(t,x,dx)[dx(1)-x(2)
    dx(2)*sin(x(4))+dx(4)^2+2*x(1)*x(3)-x(1)*dx(2)*x(4)
    dx(3)-x(4)
    x(1)*dx(2)*dx(4)+cos(dx(4))-3*x(3)*x(2)];
```

(4)求解 dx0

```
t0=0;%自变量初值
%对于 x0 和 dx0 中的题目给出的初值，如实写，没有给出的任意写
x0=[1 0 0 1]';%本题初值 x0 的都给出了，所以必须是这个
%初值 x0 中题目给出的，对应位置填 1，否则为 0
fix_x0=ones(4,1);%本题中 x0 都给出了，故全为 1
dx0=[0 0 1 1]';%本题中初值 dx0 一个都没有给出，那么全部任意写
%初值 dx0 中题目给出的，对应位置填 1，否则为 0
fix_dx0=zeros(4,1);%本题中 dx0 一个没有给出，故全部为 0
[x02,dx02]=decic(odefun,t0,x0,fix_x0,dx0,fix_dx0);
```

(5)求解微分方程

```
solution=ode15i(odefun,[0 2],x02,dx02)%x02 和 dx02 是由 decic 输出参数
```

虽然我们上面的分析是完全正确，但是好像 Matlab 无法计算出结果

其实对一些简单的 ODEs 压根没有必要通过 decic 函数来求解未知的初值，我们可以直接人工算出 x02 和 dx02，根据下面的式子

$$\begin{cases} x_1' = x_2 \\ x_2'\sin(x_4) + (x_4')^2 = -2x_1x_3 + x_1x_2'x_4 \\ x_3' = x_4 \\ x_1x_2'x_4' + \cos(x_4') = 3x_3x_2 \end{cases}$$

我们演示下，根据题目 x0=[x1,x2,x3,x4]=[1 0 0 1]，由第一个式子可以得出 x1’=0，第三个式子有 x3’=1，再联立第二四两个式子后有 x2’=0，x4’=1，故 dx0=[x1’ ,x2’,x3’,x4’]=[0 0 1 1]

看了下，隐式 ODEs 数值求解的确麻烦的些，但是和前面的 ode45 没有本质的区别，decic 函数就是为了求解那些隐式中的状态变量的一阶导数，与我们方法二中的很相似，只不过这里直接调用罢了。

其实不管隐式还是显式 ODEs，甚至 DAE，都是可以使用 ode15i 求解，只不过此时将显式当隐式处理，或者说将代数约束看成一个隐式罢了！当然大部分人都不会傻到这么做，有现成的函数不用，何必自找麻烦呢！但是对于了解和学习，我们可以试试。

5.微分代数方程(DAE)

所谓微分代数方程，是指在微分方程中，某些变量满足某些代数方程的约束。假设微分方程的更一般形式可以写成

$$M(t,x)x' = f(t,x)$$

前面所介绍的 ODEs 数值解法主要针对能够转换为一阶常微分方程组的类型，故 DAE 就无法使用前面介绍的常微分方程解法直接求解，必须借助 DAE 的特殊解法。

其实对于我们使用 Matlab 求解 DAE 时，却没有太大的改变，只需增加一个 Mass 参数即可。

描述 f(t,x)的方法和普通微分方程完全一致，由于对而对真正的微分代数方程来说，M(t,x)矩阵为奇异矩阵，在 DAE 求解程序中我们只需要设置解算器的 Mass 参数为上面的 M(t,x)矩阵即可。

下面我们以实例说明，看下面的例子，求解该方程的数值解

$$\begin{cases} x_1' = -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\ x_2' = 2x_2x_1 - 5x_2x_3 - 2x_2^2 \\ x_1 + x_2 + x_3 = 1 \end{cases}$$

其中

$$x_1(0) = 0.8, x_2(0) = x_3(0) = 0.1$$

【解】

真是万幸，选取状态变量和求状态变量的一阶导数等，微分方程转换工作，题目已经帮我们完成。

可是细心的网友会发现，最后一个方程不是微分方程而是一个代数方程**(这就是为什叫 DAE 的原因)**，其实我们可以将它视为对三个状态变量的约束。

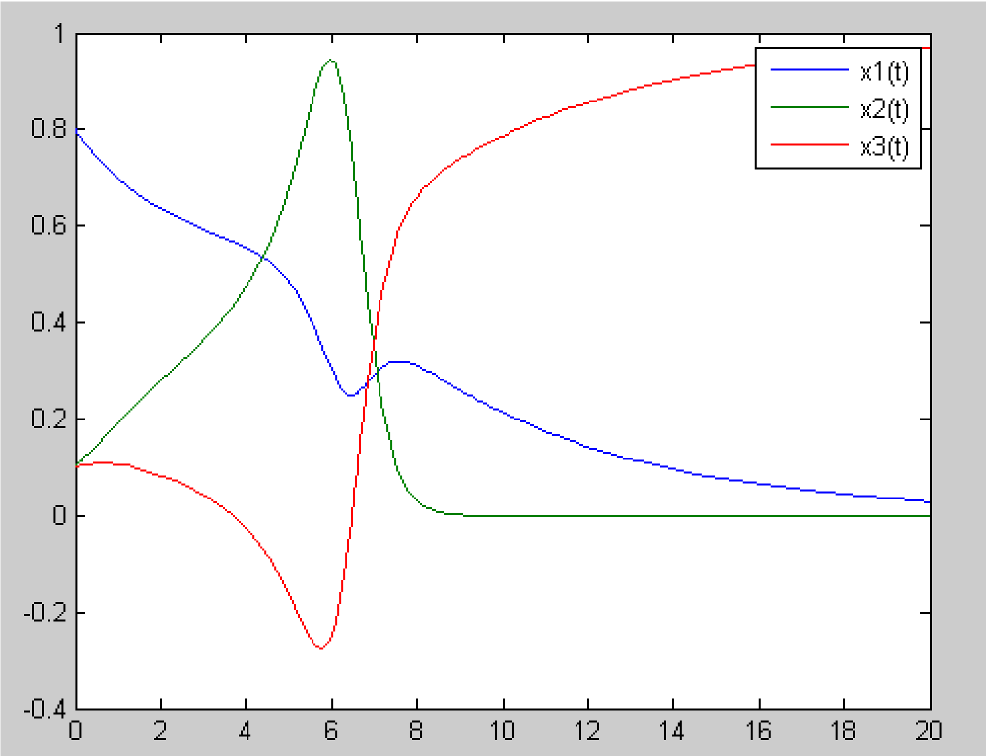
(1)用矩阵形式表示出该 DAEs

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\ 2x_1x_2 - 5x_2x_3 - 2x_2^2 \\ x_1 + x_2 + x_3 - 1 \end{bmatrix}$$

(2)编写 Matlab 代码

```
odefun=@(t,x)[-0.2*x(1)+x(2)*x(3)+0.3*x(1)*x(2);
    2*x(1)*x(2)-5*x(2)*x(3)-2*x(2)*x(2);
    x(1)+x(2)+x(3)-1];%微分方程组
 M=[1 0 0;0 1 0;0 0 0];%质量矩阵
 options=odeset('mass',M);%对以DAE问题，mass属性必须设置
```

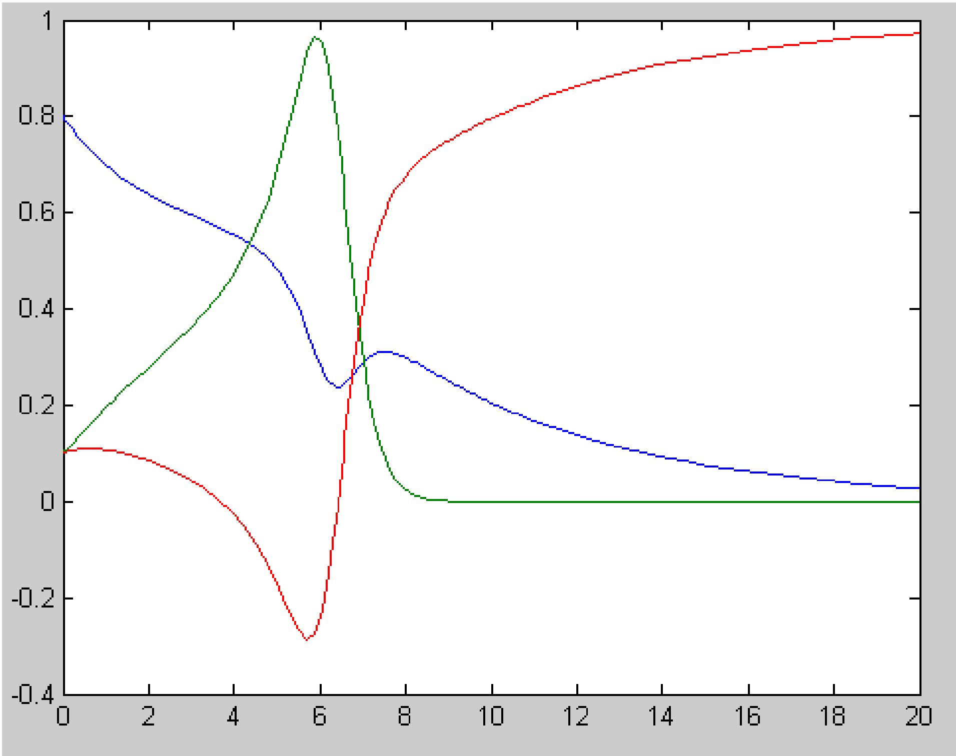
```
x0=[0.8;0.1;0.1];%初值
[t,x]=ode15s(odefun,[0 20],x0,options);%这里好像不能使用ode45
figure('numbertitle','off','name','DAE demo—by Matlabsky')
plot(t,x)
legend('x1(t)','x2(t)','x3(t)')
```



在介绍隐式 ODEs 时，我们说了 ode15i 也可以求解 DAEs，原理就是将约束看成一个隐式，好，下面我们看看

```
odefun=@(t,x,dx)[dx(1)+0.2*x(1)-x(2)*x(3)-0.3*x(1)*x(2);
        dx(2)-2*x(1)*x(2)+5*x(2)*x(3)+2*x(2)^2;
        x(1)+x(2)+x(3)-1];
%状态变量初值，题目中给出
x0=[0.8;0.1;0.1];
%该初值全部给定，按理说应该全部为1，但是记住方程中有一个约束，故其实只有两个独立初值
x0_fix=[1;0;1];%随意一个改为0都可以，比如[0;1;1]或者[1;1;0]
%状态变量一阶微分初值，题目没有给出，可以任意写
dx0=[1;1;1];
%该初值一个都没有给出，故全部为0
dx0_fix=[0;0;0];
%时间变量的初值
t0=0;
%计算相容初值
[x0,dx0]=decic(odefun,0,x0,x0_fix,dx0,dx0_fix);
```

```
[x0,dx0]  % 相容初始条件
res=ode15i(odefun,[0,20],x0,dx0);
figure('numbertitle','off','name','DAE demo—by Matlabsky')
plot(res.x,res.y)
```



其实，有些简单 DAEs 是可以转换为 ODEs 的，由于代数方程只是状态变量的一个约束，假如约束充分的话，我们就可以使用消参的思想，将约束反代回 ODEs 中，这样就将那个约束方程消去了，最后只剩下那些微分方程了

对于本例，我们只要通过第三个方程表示出 $x_3 = 1 - x_1 - x_2$，再将 x3 反代回前两个微分方程中则有

$$\begin{cases} x_1' = -0.2x_1 + x_2(1 - x_1 - x_2) + 0.3x_1x_2 \\ x_2' = 2x_1x_2 - 5x_2(1 - x_1 - x_2) - 2x_2^2 \end{cases}$$

这个就是一个正常的 ODEs 了，好，下面我们就重新使用 Matlab 求解下看

```
x0=[0.8;0.1];
odefun=@(t,x)[-0.2*x(1)+x(2)*(1-x(1)-x(2))+0.3*x(1)*x(2)
   2*x(1)*x(2)-5*x(2)*(1-x(1)-x(2))-2*x(2)*x(2)];
[t,x]=ode45(odefun,[0,20],x0);
figure('numbertitle','off','name','DAE demo—by Matlabsky')
plot(t,x)
legend('x1(t)','x2(t)')
```

6.延迟微分方程(DDE)

延迟微分方程的一般形式为：

$$x'(t)=f(t,x(t-t_1),x(t-t_2),\ldots,x(t-t_n))$$

其中 taui≥0 为状态变量 x(t)的延迟常数。

Matlab 中提供了求解这类方程的隐式 Runge-Katta 算法 dde23()函数，可以直接求解延迟微分方程，该函数的格式为：

sol = dde23(ddefun,tau,history,tspan,*options*)

输入参数：

ddefun：描述延迟微分方程的句柄

tau=[tau1,taou,….taun]：

history：描述 t≤t0 时的状态变量的值的函数，题目一般会直接给出

tspan：同理 ode45

注意返回参数 sol.y 是按行排列，只要与 ode**函数的转置

看下面的例子，进行说明。假设延迟微分方程组为

$$\begin{cases} x'(t)=1-3x(t)-y(t-1)-0.2x^3(t-0.5)-x(t-0.5) \\ y''(t)+3y'(t)+2y(t)=4x(t) \end{cases}$$

其中

在$t\le 0$时, $x(t)=y(t)=y'(t)=0$

【解】

(1)选取状态变量。若想得出该方程的数值解，需要先将其变换成一阶显式微分方程组

$$x_1(t)=x(t),\ \ x_2(t)=y(t),\ \ x_3(t)=y'(t)$$

(2)计算每个状态变量的一阶微分

$$\begin{cases} x_1'(t)=1-3x(t)-x_2(t-1)-0.2x_1^3(t-0.5)-x_1(t-0.5) \\ x_2'(t)=x_3(t) \\ x_3'(t)=4x_1(t)-2x_2(t)-3x_3(t) \end{cases}$$

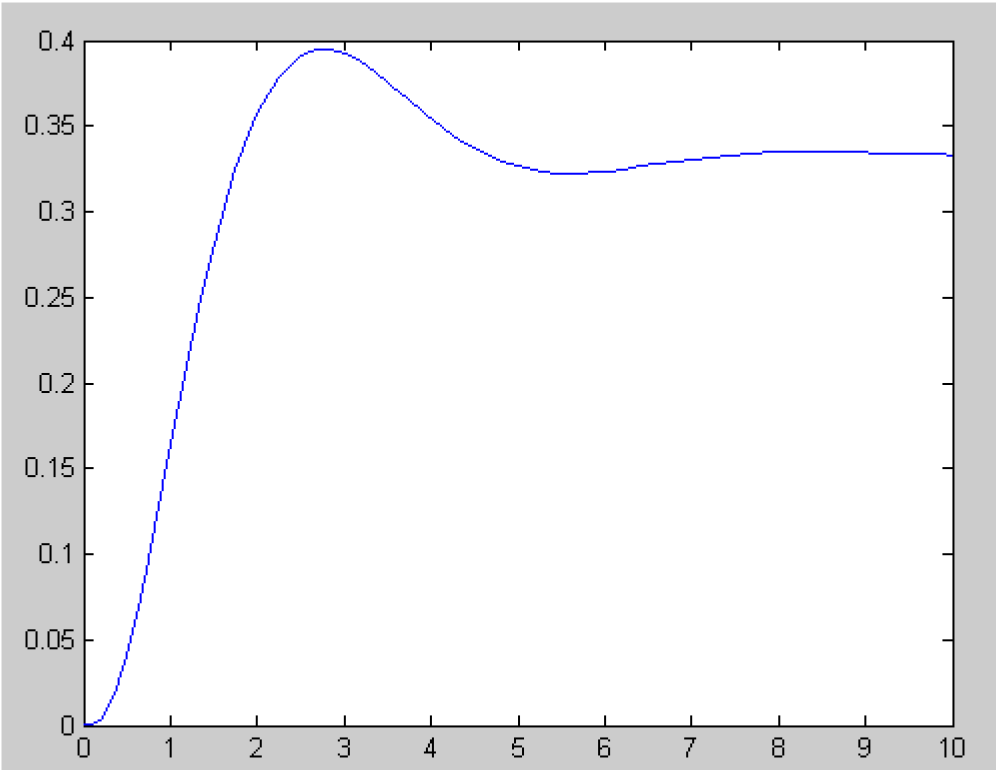
(3)编写延迟函数。从第一个方程可以看出 x2 延迟 1，x1 延迟 0.5，我们取 tau=[1 0.5]

```
function dx=ddefun(t,x,z)
%z(i,j)表示状态变量 xj 延迟了 tau(i)
tau1=z(:,1);%第一列表示延迟了 tau(1)=1 的所有状态变量
```

```
tau2=z(:,2);
%x2 延迟了 1，故表示为 z(2,1)或者 tau1(2)
%x1 延迟了 0.5，故表示为 z(1,2)或者 tau2(1)
dx=[1-3*x(1)-tau1(2)-0.2*tau2(1)^3-tau2(1)
    x(3)
    4*x(1)-2*x(2)-3*x(3)];
```

(4)编写主调函数

```
tau=[1 0.5];
history=[0 0 0]';
tspan=[0 10];
sol=dde23(@ddefun,tau,history,tspan);
figure('numbertitle','off','name','DDE demo—by Matlabsky')
plot(sol.x,sol.y(2,:))
```



7.边值问题(BVP)

前面的讲到现在为止都是初值问题，也就是说 Matlab 的 ode**系列解算器，默认将 tspan(1)作为初值条件时的 t，比如你将初值条件换为 x(2)=x’(2)=0，那么 tspan(1)就必须是 2。

但是工程应用中我们经常遇到边值问题，这些是那些 ode**函数无能为力的，当然我们可以自己编写函数求解(比如 shooting)，但是那个毕竟不是某些人能力所及的，还好 Matlab 中提供了 bvp 解算器。

solinit = bvpinit(x, yinit, *params*)
sol = bvpsolver(odefun,bcfun,solinit,*options*)

由于边值问题可能有多解，为了便于我们确定那个解是我们需要的，所以必须使用 bvpinit 函数对初值进行估计

解算器(bvpsolver)：Matlab 中提供了 bvp4c 和 bvp5c，后者误差控制更好些

输入参数：
x：需要计算的网格点，相当于 ode**的 tspan
yinit：猜测的值，可以是具体值，也可以是函数，类似与 ode**的 x0
params：其它未知参数，也是一个猜测值
odefun：描述边值问题微分方程的函数句柄
bcfun：边值函数，一般是双边值(x 的上下限即认为两个边界)，但也支持多边值(具体看帮助)
solinit：由 bvpinit 生成的初始化网格
options：BVP 解算器优化参数，可以通过 bvpset 设置，具体参数查看帮助

输出参数：
大部分同理 ode45

好下面我们以例子说事，考察包含未知参数λ的二阶微分方程，求解λ，由于λ也是未知的，故需要三个边值条件

$$y''+(l-2q\cos 2x)=0$$

其中

$$q=5, y'(0)=y'(p)=0, y(0)=1$$

【解】

(1)选用状态变量

$$x_1=y \quad x_2=y'$$

(2)求每个状态变量的一阶微分

$$\begin{cases} x_1' = y' = x_2 \\ x_2' = -l + 2q\cos 2x \end{cases}$$

(3)编程实现求解

```
q=5;
lambda = 15;%未知参数猜测值
x=linspace(0,pi,10);%需要计算的点
yinit=@(x)[cos(4*x);-4*sin(4*x)];%使用函数估计目标值
solinit = bvpinit(x,yinit,lambda);%生成计算网格
odefun=@(x,y,lambda)[y(2);-(lambda - 2*q*cos(2*x))*y(1)];%边值微分方程
bcfun=@(ya,yb,lamdba)[ya(2);yb(2);ya(1)-1];%边界条件
sol = bvp4c(odefun,bcfun,solinit);
fprintf('The fourth eigenvalue is approximately %7.3f.\n',sol.parameters)
xint = linspace(0,pi);
Sxint = deval(sol,xint);%计算xint对应的y值
figure('name','BVP Demo—by Matlabsky','numbertitle','off')
plot(xint,Sxint(1,:))%绘制xint和x1=y的图形，即x与y
axis([0 pi -1 1.1])
title('Eigenfunction of Mathieu''s equation.')
xlabel('x')
ylabel('solution y')
```