Using Matlab ode45 to solve differential equations

Nasser M. Abbasi

May 30, 2012 page compiled on July 1, 2015 at 11:43am

Contents

1 download examples source code

1. `first_order_ode.m.txt`
2. `second_order_ode.m.txt`
3. `engr80_august_14_2006_2.m.txt`
4. `engr80_august_14_2006.m.txt`
5. `ode45_with_piecwise.m.txt`

2 description

This shows how to use Matlab to solve standard engineering problems which involves solving a standard second order ODE. (constant coefficients with initial conditions and nonhomogeneous).

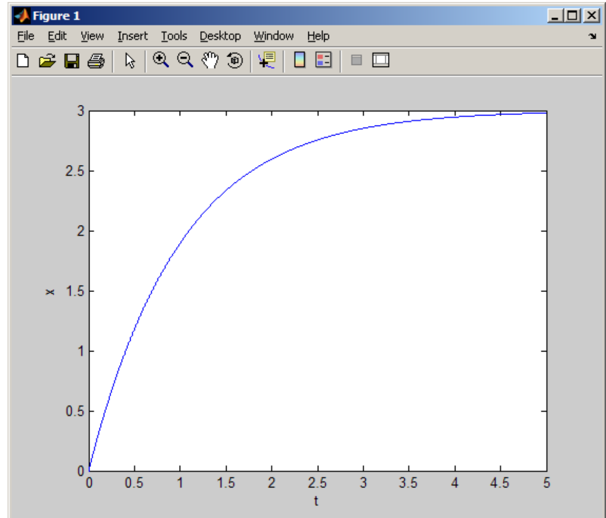
A numerical ODE solver is used as the main tool to solve the ODE's. The matlab function ode45 will be used. The important thing to remember is that ode45 can only solve a first order ODE. Therefore to solve a higher order ODE, the ODE has to be first converted to a set of first order ODE's. This is possible since an n order ODE can be converted to a set of n first order ODE's.

Gives a first order ODE

$$\frac{dx}{dt} = f(x,t)$$

An example of the above is $\frac{dx}{dt} = 3e^{-t}$ with an initial condition $x(0) = 0$. Here is the result of solving this ODE in Matlab. Source code is `first_order_ode.m.txt`

```
function test1

% SOLVE  dx/dt = -3 exp(-t).
% initial conditions: x(0) = 0

t=0:0.001:5;   % time scalex
initial_x=0;

[t,x]=ode45( @rhs, t, initial_x);

plot(t,x);
xlabel('t'); ylabel('x');

    function dxdt=rhs(t,x)
        dxdt = 3*exp(-t);
    end
end
```



To solve a second order ODE, using this as an example.

$$\frac{d^2x}{dt^2} + 5\frac{dx}{dt} - 4x(t) = \sin(10\ t)$$

Since ode45 can only solve a first order ode, the above has to be converted to two first order ODE's as follows. Introduce 2 new state variables x_1, x_2 and carry the following derivation

$$\left.\begin{aligned} x_1 &= x \\ x_2 &= x' \end{aligned}\right\} \overset{\text{take derivative}}{\rightarrow} \left.\begin{aligned} x_1' &= x' \\ x_2' &= x'' \end{aligned}\right\} \overset{\text{do replacement}}{\rightarrow} \left.\begin{aligned} x_1' &= x_2 \\ x_2' &= -5x' + 4x + \sin(10t) \end{aligned}\right\} \rightarrow \left.\begin{aligned} x_1' &= x_2 \\ x_2' &= -5x_2 + 4x_1 + \sin(10t) \end{aligned}\right\}$$

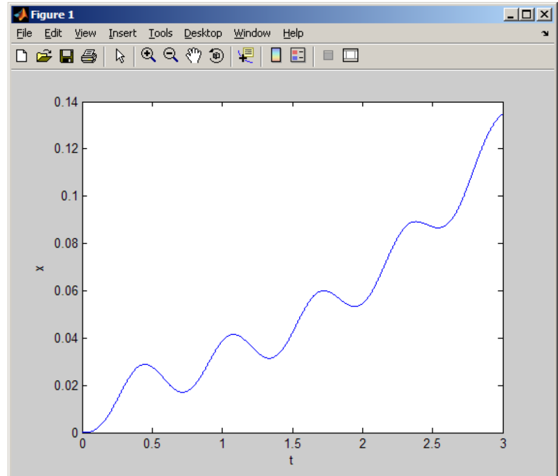
The above gives 2 new first order ODE's. These are

$$\begin{aligned} x_1' &= x_2 \\ x_2' &= -5x_2 + 4x_1 + \sin(10t) \end{aligned}$$

Now ode45 can be used to solve the above in the same way as was done with the first example. The only difference is that now a vector is used instead of a scalar.

This is the result of solving this in Matlab. The source code is `second_order_ode.m.txt`

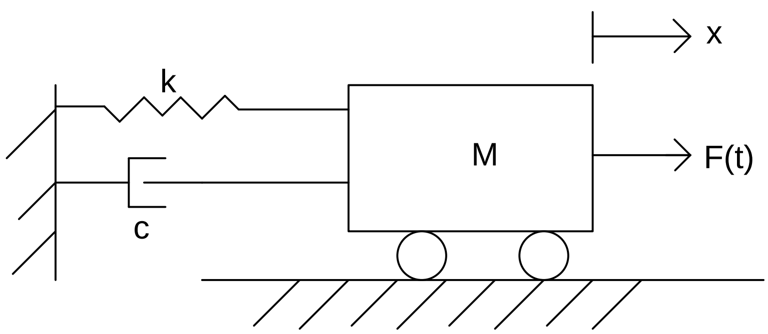
```
function second_oder_ode

% SOLVE  d2x/dt2+5 dx/dt - 4 x = sin(10 t)
% initial conditions: x(0) = 0, x'(0)=0

t=0:0.001:3;   % time scale

initial_x    = 0;
initial_dxdt = 0;

[t,x]=ode45( @rhs, t, [initial_x initial_dxdt] );

plot(t,x(:,1));
xlabel('t'); ylabel('x');

    function dxdt=rhs(t,x)
        dxdt_1 = x(2);
        dxdt_2 = -5*x(2) + 4*x(1) + sin(10*t);

        dxdt=[dxdt_1; dxdt_2];
    end
end
```



3 Simulation

Now ode45 is used to perform simulation by showing the solution as it changes in time.

Given a single degree of freedom system. This represents any engineering system whose response can move in only one direction. A typical SDOF (single degree of freedom) is the following mass/spring/damper system.



The first step is to obtain the equation of motion, which will be the second order ODE. Drawing the free body diagram and from Newton's second laws the equation of motion is found to be

$$mx'' + cx' + kx = f(\omega_f t)$$

In the above, ω_f is the forcing frequency of the force on the system in rad/sec.

The response of the system (the solution of the system, or $x(t)$) is simulated for different parameters.

For example, the damping c can be changed, or the spring constant (the spring stiffness) to see how $x(t)$ changes. The forcing function frequency ω_f can also be changed.

The following definitions are used in the Matlab code.

Natural frequency of the system $\omega = \sqrt{\frac{k}{m} - \left(\frac{c}{2m}\right)^2}$

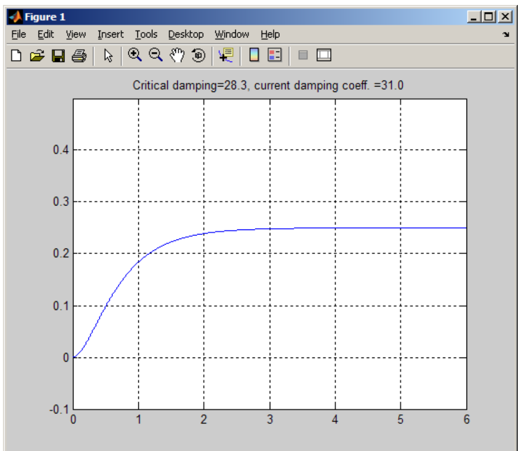
Damping ratio $\varsigma = \frac{c}{c_r}$ where c is the damping coefficient and c_r is the critical damping.

$$c_r = 2\sqrt{km}$$

When $c > c_r$ the system is called over damped. When $c < c_r$ the system is called underdamped

The following example runs a simulation showing the effect of changing the damping when the forcing function is a step function. The response to a step function is a standard method used to analyze systems.

```
function engr80_august_14_2006_2()

% shows how to use Matlab to animation response of one degree of
% freedom system.
% show the effect of changing the damping of the system on the response.
% by Nasser Abbasi, UCI.

clear all; close all;

t_start = 0;
t_end   = 6;  %final time in seconds.
time_span =t_start:0.001:t_end;

k = 40; % spring stiffness. N/m
m = 5;  % mass, kg

cr = 2*sqrt(k*m);  %critical damping

fprintf('critical damping coef. of system is %f\n',cr);

initial_position = 0;
initial_speed    = 0;

x0 = [initial_position  initial_speed];

% Now start the simulation, change damping.

for c = 0: .5 : cr+.1*cr

    [t,x]=ode45(@rhs,time_span,x0);
    plot(t,x(:,1));
    title(sprintf('Critical damping=%4.1f, current damping coeff. =%4.1f',cr,c));
    ylim([-.1 .5]);
    drawnow;
    pause(.1);

end

grid
%**************************************
% solves m x''+ c x' + k x = f(t)
%**************************************
    function xdot=rhs(t,x)

        xdot_1 = x(2);
        xdot_2 = -(c/m)*x(2) - (k/m)*x(1) + force(t)/m;

        xdot = [xdot_1 ; xdot_2 ];
    end
%*******************
% The forcing function, edit to change as needed.
%*******************
    function f=force(t)

        P = 100;   % force amplitude
        %f=P*sin(omega*t);

        f=10;  %unit step

        %if t<eps     %impulse
        %   f=1
        %else
        %    f=0;
        %end

        %f=P*t;  %ramp input
    end
end
```



4 Using ode45 with piecewise function

ode45 can be used with piecewise function defined for the RHS. For example, given $x''(t) - x(t) = c$ where $c = 1$ for $0 <= t < 1$ and $c = 20$ for $1 <= t < 2$ and $c = 3$ for $2 <= t <= 3$, the following code example shows one way to implement the above.

`ode45_with_piecwise.m.txt`

5 Listing of source code

first_order_ode.m

```
function first_oder_ode

% SOLVE  dx/dt = -3 exp(-t).
% initial conditions: x(0) = 0

t=0:0.001:5;   % time scalex
initial_x=0;

[t,x]=ode45( @rhs, t, initial_x);

plot(t,x);
xlabel('t'); ylabel('x');

    function dxdt=rhs(t,x)
        dxdt = 3*exp(-t);
    end
end
```

second_order_ode.m

```
function second_oder_ode

% SOLVE  d2x/dt2+5 dx/dt - 4 x = sin(10 t)
% initial conditions: x(0) = 0, x'(0)=0

t=0:0.001:3;   % time scale

initial_x    = 0;
initial_dxdt = 0;

[t,x]=ode45( @rhs, t, [initial_x initial_dxdt] );

plot(t,x(:,1));
xlabel('t'); ylabel('x');

    function dxdt=rhs(t,x)
        dxdt_1 = x(2);
        dxdt_2 = -5*x(2) + 4*x(1) + sin(10*t);
```

```

        dxdt=[dxdt_1; dxdt_2];
    end
end
```

engr80_august_14_2006_2.m

```
function engr80_august_14_2006_2()

% shows how to use Matlab to animation response of one degree of
% freedom system.
% show the effect of changing the damping of the system on the response.
% by Nasser Abbasi, UCI.

clear all; close all;

t_start = 0;
t_end   = 6;  %final time in seconds.
time_span =t_start:0.001:t_end;

k = 40; % spring stiffness. N/m
m = 5;  % mass, kg

cr = 2*sqrt(k*m);  %critical damping

fprintf('critical damping coef. of system is %f\n',cr);

initial_position = 0;
initial_speed    = 0;

x0 = [initial_position  initial_speed];

% Now start the simulation, change damping.

for c = 0: .5 : cr+.1*cr

    [t,x]=ode45(@rhs,time_span,x0);
    plot(t,x(:,1));
    title(sprintf('Critical damping=%4.1f, current damping coeff. ...
       =%4.1f',cr,c));
    ylim([-.1 .5]);
    drawnow;
    pause(.1);

end

grid
%**************************************
% solves m x''+ c x' + k x = f(t)
%**************************************
```

```
    function xdot=rhs(t,x)

        xdot_1 = x(2);
        xdot_2 = -(c/m)*x(2) - (k/m)*x(1) + force(t)/m;

        xdot = [xdot_1 ; xdot_2 ];
    end
%*******************
% The forcing function, edit to change as needed.
%*******************
    function f=force(t)

        P = 100;   % force amplitude
        %f=P*sin(omega*t);

        f=10;  %unit step

        %if t<eps      %impulse
        %   f=1
        %else
        %    f=0;
        %end

        %f=P*t;  %ramp input
    end
end
```

engr80_august_14_2006.m

```
function engr80_august_14_2006()
% shows how to use Matlab to animation response of one degree of
% freedom system.
% by Nasser Abbasi, UCI.

clear all; close all;

t_start = 0;
t_end   = 6;  %final time in seconds.
time_span =[ t_start t_end];
time_span =t_start:0.001:t_end;

k = 100; % spring stiffness. N/m
c = 20; % damping coeff. N-s/m
m = 5; % mass, kg

natural_damped_omega = sqrt( k/m - (c/(2*m))^2 );

fprintf('Natural damped frequency of system is ...
   %f\n',natural_damped_omega);

```

```
initial_position = 0;
initial_speed    = 0;

x0 = [initial_position  initial_speed];

for omega=0:0.1:natural_damped_omega+0.1
    [t,x]=ode45(@rhs,time_span,x0);
    plot(t,x(:,1));
    title(sprintf('forcing freq=%4.1f',omega));
%     ylim([-.1 2]);
    drawnow;
%     pause(.1);
end

    grid

%*************************************
% solves m x''+ c x' + k x = f(t)
%*************************************
function xdot=rhs(t,x)

xdot_1 = x(2);
xdot_2 = -(c/m)*x(2) - (k/m)*x(1) + force(t)/m;


xdot = [xdot_1 ; xdot_2 ];
end

%*******************
%
%*******************
function f=force(t)

P = 100;   % force amplitude
f=P*sin(omega*t);

%f=10;  %unit step

%if t<eps     %impulse
%   f=1
%else
%     f=0;
%end

%f=P*t;  %ramp input

end


end
```

ode45_with_piecwise.m

```

%---------------------------------
%
%Example solve  x''- x = c
%
%where c=1  for 0≤t<1
%      c=20 for 1≤t<2
%  c=3  for 2≤t≤3
%
%IC  x = 0, t = 0
%    x'= 1, t = 0
%


%--------------------
function  ode45_with_piecwise()

t             = 0:0.1:3;   % time scale
initial_x     = 0;
initial_dxdt = 1;

[t,x] = ode45( @rhs , t, [initial_x initial_dxdt] );

plot(t,x(:,1));
xlabel('t'); ylabel('x');

end

%--------------------
%ode45 rhs
function dxdt=rhs(t,x)
  dxdt_1 = x(2);
  dxdt_2 = x(1)+ 1*((0≤t)&(t<1)) + 20*((1≤t)&(t≤2))+ 3*((2≤t)&(t≤3));
  dxdt   = [dxdt_1; dxdt_2];
end
```