

Using Matlab ode45 to solve differential equations

Nasser M. Abbasi
August 10, 2006

May 11, 2012

PDF

1 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

2 description

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

We will use numerical ODE solver as the main tool to solve the ODE's. The matlab function ode45 will be used. The important thing to note is that ode45 can only solve a first order ODE's. Hence to solve a higher order ODE, we must convert the higher order ODE to a set (or array) of first order ODE's. As we know, an n order ODE can be converted to n first order ODE's.

We will start by solving a first order ODE

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

This is an example of how this is done in matlab. Consider $\frac{dx}{dt} = 3e^{-t}$ with initial conditions $x(0) = 0$

```

function test1

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

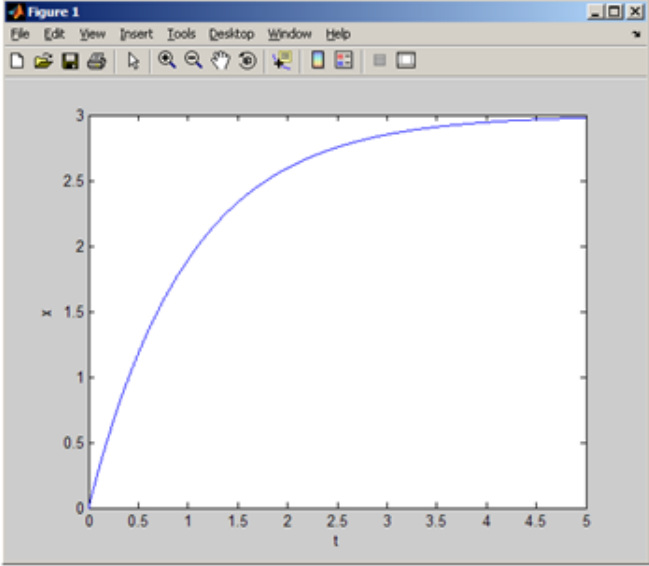
t=0:0.001:5; % time scalex
initial_x=0;

[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

```



Now we will solve a second order ODE. This is an example. Consider

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

Since ode45 can only solve a first order ode, we convert the above ODE to two first order ODE's as follows. Introduce 2 new state variables x_1, x_2 and write

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

Hence we have obtained now 2 new first order ODE's, which are

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

Now we can use ode45 to solve this in the same way as we did for the first example. Only difference is that now we will use an array to process things with instead of a scalar. This is the Matlab code to solve the above.

```

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;

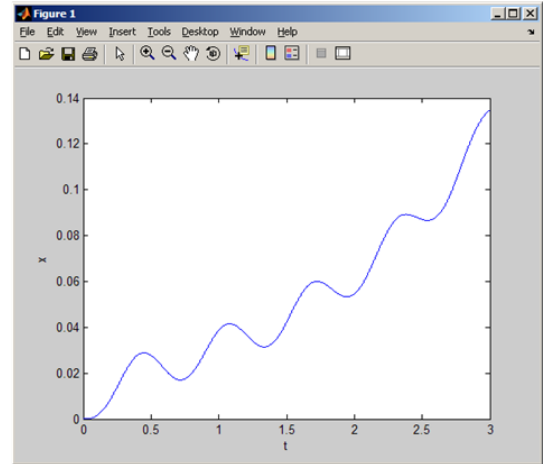
[t,x]=ode45( @rhs, t, [initial_x initial_dxdt] );

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

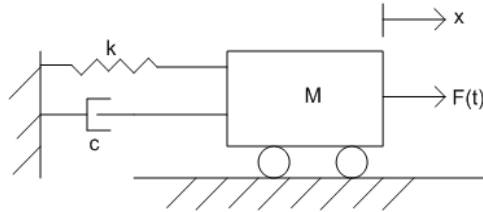
function dxdt=rhs(t,x)
    dxdt_1 = x(2);
    dxdt_2 = -5*x(2) + 4*x(1) + sin(10*t);

    dxdt=[dxdt_1; dxdt_2];
end
end

```



Now that we know how to use Matlab to solve ODE's, lets us now use it to do a simulation showing the solution of an engineering problem. Consider a single degree of freedom system. This is any engineering system whose response can move in only a single direction. In general we can have 6 degrees of freedom (3 along the x,y,z for translation, and 3 angles for rotations). A typical SDOF (single degree of freedom) system is the mass/spring/damper system.



The first step is to obtain the equation of motion, which will be the second order ODE, which we will solve using ode45. Draw the free body diagram and from Newton's second law we obtain the equation of motion as

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

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

Now we can simulation the response of the system (the solution of the system, or $x(t)$) as we vary different parameters. For example, we can change the damping c or we can change the spring constant (the spring stiffness) and observe how $x(t)$ changes. We can also change the forcing function frequency ω_f and observe what happens to the response, we should see resonance occurs when the forcing frequency approaches the natural frequency of the system. We will do all these using Matlab by running different simulations.

First recall the following definitions which we will use in the Matlab code.

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

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

$$c_r = 2\sqrt{k m}$$

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

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

```

