How to pass information to the function called by the ode solver

a) Using **global variables**, for example, in the main program “**odesho.m**”

```
global k m  Declare k and m to be global variables, visible to functions with the
corresponding global statement.
m = 25/4;    Assign values to k and m
k = pi^2;
```

The ode solver is called from “odesho.m” with

```
[t yode] = ode45('sho',tt,yode0);
```

In the function “**sho.m**”

```
global k m  Provide access to the global variables k and m
rhs(2) = -k/m*y(1);  before using them.
```

To use different values for k and m, change the parameters in the main program and
call the ode-solver again.

Advantage: easy to program

Disadvantage: can be hard to debug, it is better to use “global” only with constants.

b) Using **nested functions**, for example, the main program “**odesho_ext.m**” calls the
nested function “**call_ode_sho.m**” (which in turn calls the ode solver) and passes k
and m as arguments, rather than declaring them to be global.

In the main program “**odesho_ext.m**”

```
[t yode] = call_ode_sho(tt,yode0,k,m);
```

The function “**call_ode_sho.m**” contains a nested function and calls the ode solver

```
% JLS, February 9, 2009
% call_ode_sho.m
% A nested function that calls the ode-solver with the parameters for the
% simple harmonic oscillator in one dimension

function [t yode] = call_ode_sho(tt,yode0,k,m) % the function call_ode_sho
% has more arguments than the ode solver allows
[t yode] = ode45(@sho,tt,yode0);  % start of call to ode solver, note the @

    function rhs = sho(t,y);       % start of nested function sho
        rhs = zeros(2,1);
        rhs(1) = y(2);
        rhs(2) = -(k/m)*y(1);
    end % end of nested function sho
end   % end of call to ode solver
```

To use different values for k and m, call the function call_ode_sho again with
changed arguments k and m.

Advantage: very transparent (once one is used to it)

Disadvantage: a bit more programming
How to make a figure with graphs of solutions for different parameters

a) Small number of graphs; example, phase-space plot of the simple harmonic oscillator for three different sets of initial conditions
file: odesho_three.m

1. Assign three sets of initial conditions
2. Initialize three solution arrays to zero
3. Solve the differential equation three times, once for each set of initial conditions.
4. Plot the solutions in the same graph

```matlab
% set initial conditions
x01 = 0; v01 = 100;
yode01 = [x01 v01];
x02 = 0; v02 = 100/2;
yode02 = [x02 v02];
x03 = 0; v03 = 100/4;
yode03 = [x03 v03];

% initialize solutions
yode1 = zeros(N,2);
yode2 = zeros(N,2);
yode3 = zeros(N,2);

% calculate
[tt yode1] = ode45('sho',t,yode01);
[tt yode2] = ode45('sho',t,yode02);
[tt yode3] = ode45('sho',t,yode03);

% display
plot(yode1(:,1),yode1(:,2),'b-',...
yode2(:,1),yode2(:,2),'r--',...
yode3(:,1),yode3(:,2),'k-.')
```

b) Large number of graphs; example, phase-space plot of the simple harmonic oscillator for ten different initial velocities using the “hold all” “hold off” commands in plotting
files: odesho_for_loop.m and odesho_for_loop_simple.m

1. Set parameters for a for loop over the initial conditions, i.e. set number of velocities, the increment and initialize the loop counter.
   • If you want to include a legend, initialize also a vector for the “plot handles” and a “cell array” for the legend entries of the initial velocity
2. Assign the first set of initial conditions
3. Initialize one solution array to zero.
4. Open a figure, set the axis limits, and hold the figure
5. Set up a for loop; inside the loop, solve the differential equation, plot the graph, and update the initial condition.
6. Set “hold off” and finish the figure.

c) Several graphs; example, phase-space plot of the simple harmonic oscillator for three of ten different initial velocities using a larger array to store the solution vectors
file: odesho_for_array.m

1. Perform steps 1 – 3 as above (there is no need to initialize plot handles)
2. Initialize arrays for the collected solutions to zero.
3. Set up a for loop; inside the loop, solve the differential equation, add the current solution vectors to the solution arrays, and update the initial condition.
4. Plot the results.