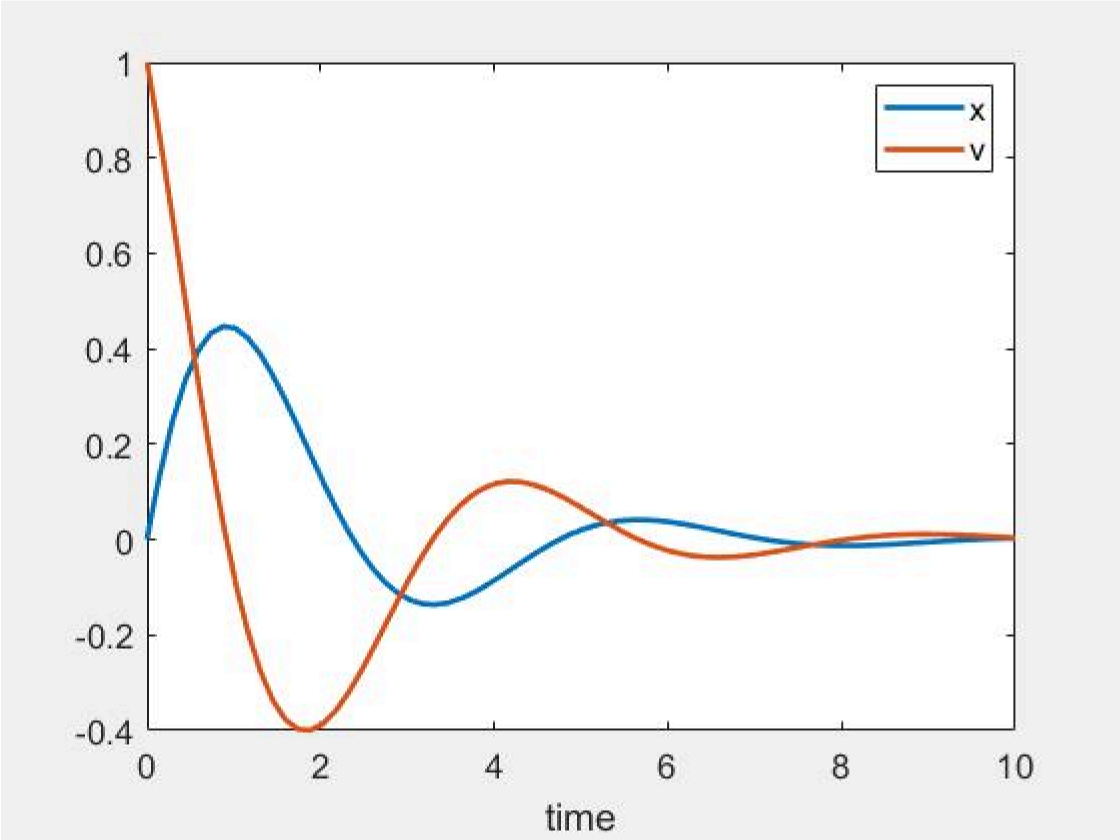
Parameter Estimation in Mass-Spring-Damper model

Table of Contents

Run with some initial parameters

```
[T,Y] = ode45(@(t,y)massspringdamper_diff(t,y, [2 1]), [0 10], [0 1]);
plot(T,Y(:,1), T,Y(:,2), 'LineWidth',2);
xlabel('time'); legend({'x','v'});
```



Grab Experimental data

Assume you are given the following experimental data. Unfortunately, the authors did not give us the numerical data, so we have to eyeball the values from the figure.

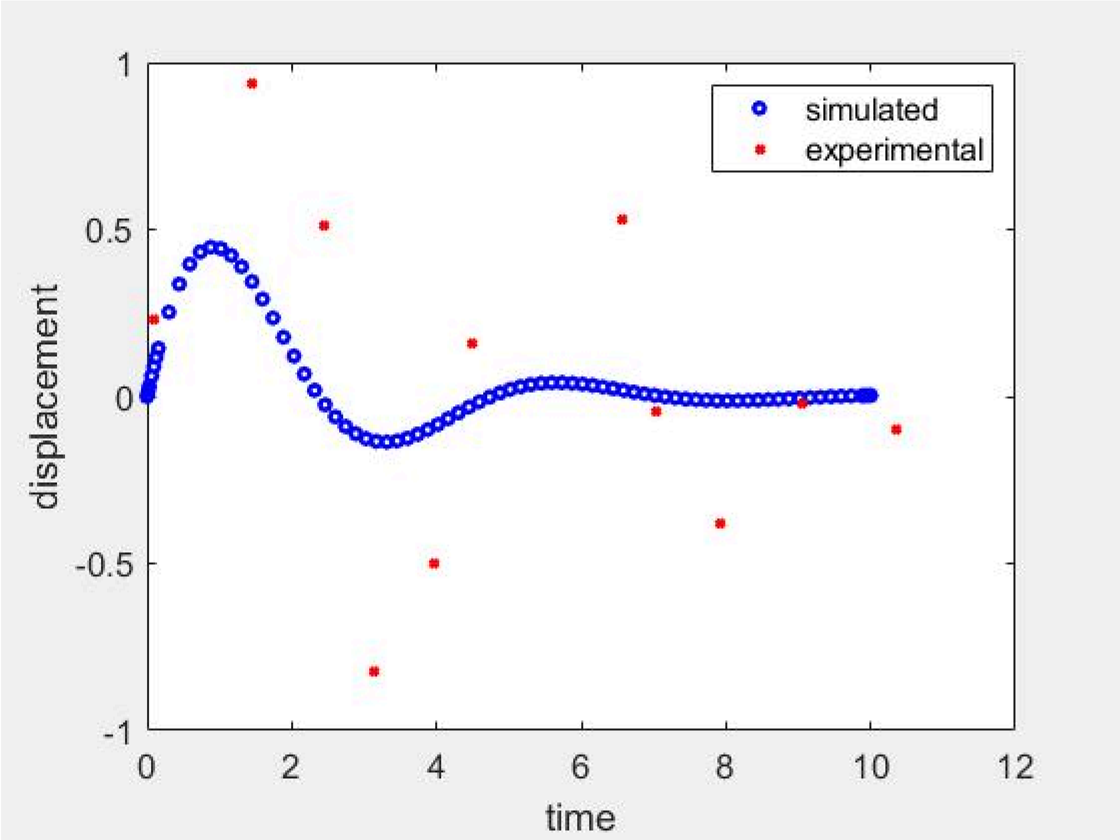
There are a number of tools available to extract experimental data from graphs. I have used the "Grabit" tool (available on Matlab Exchange) to extract & save the experimental data into massspringdamper_data_grabbed.mat

```
load('massspringdamper_data_grabbed.mat')
Tdata = massspringdamper_data_grabbed(:,1);
Ydata = massspringdamper_data_grabbed(:,2);
```

Initial Error

Let's overay the experimental data & simulated data and also calculate the error using comparetwotime-series() function.

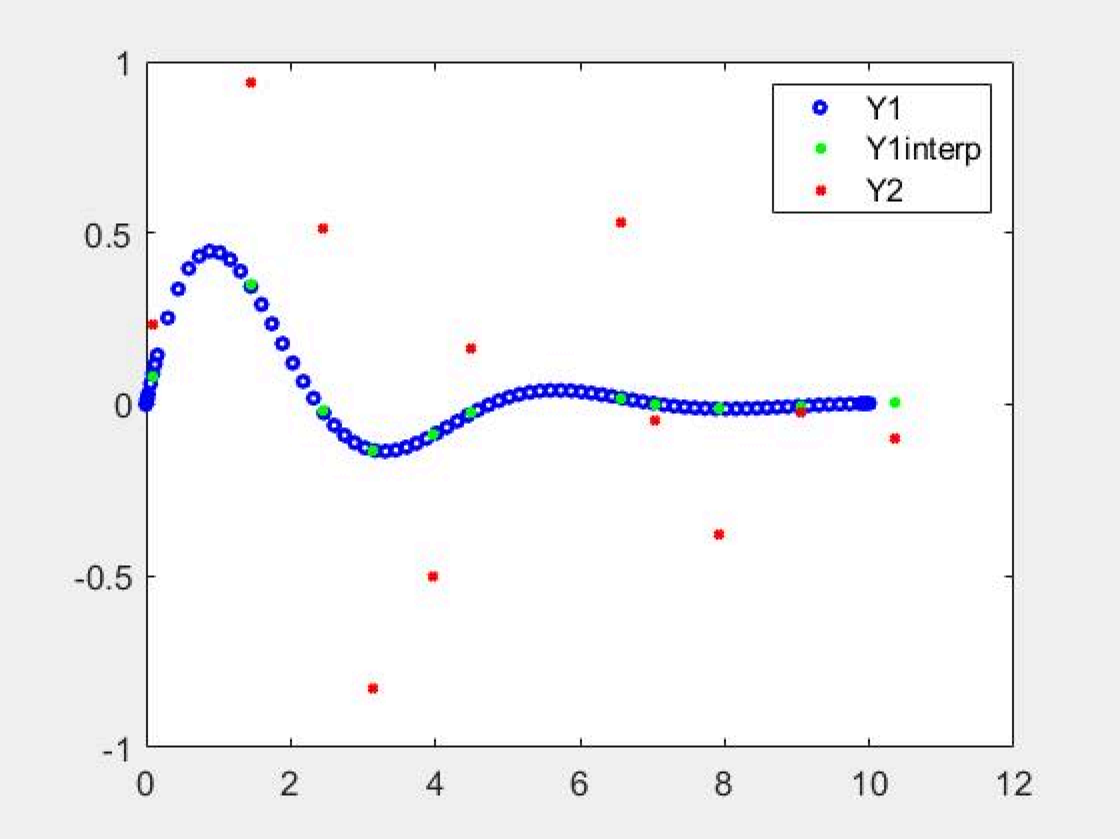
```
plot(T,Y(:,1),'ob', Tdata,Ydata,'xr', 'LineWidth',2,'MarkerSize',4);
xlabel('time'); ylabel('displacement');
 legend({'simulated','experimental'});
```




```
error=comparetwotimeseries(T,Y(:,1), Tdata,Ydata,true)
```

error =

0.1583



Objective Function

Define a function that returns the error, for a given parameter vector. The function massspringdamper_eval() does exactly what we did above, but allows us to calculate error for any given parameters.

```
error = massspringdamper_eval([2 1])
```

```
error =

    0.1583
```

```
error = massspringdamper_eval([2 2])
```

```
error =

    0.1915
```

Unconstrained Optimization

Let's have matlab find us the best pair of k/c parameters:

```
[xbest, besterror] = fminunc(@massspringdamper_eval,[2 1],
 optimset('display','iter'))
```

```
                                                        First-order
 Iteration  Func-count       f(x)        Step-size      optimality
     0           3         0.15828                        0.0529
     1          18       0.0910317       12.8958          0.0641
     2          30       0.0894757       0.07722          0.0903
     3          36       0.0812331      0.380055          0.0403
     4          39       0.0805085             1          0.0179
     5          42       0.0803733             1         0.00178
     6          45       0.0803709             1        0.000438
     7          48       0.0803707             1        3.16e-05
     8          51       0.0803707             1        3.44e-06
     9          54       0.0803707             1        6.21e-08

Local minimum found.

Optimization completed because the size of the gradient is less than
the value of the optimality tolerance.


xbest =

    1.7095    0.1985


besterror =

    0.0804
```

Nelder-Mead Optimization

Let's search again, now using fminsearch.

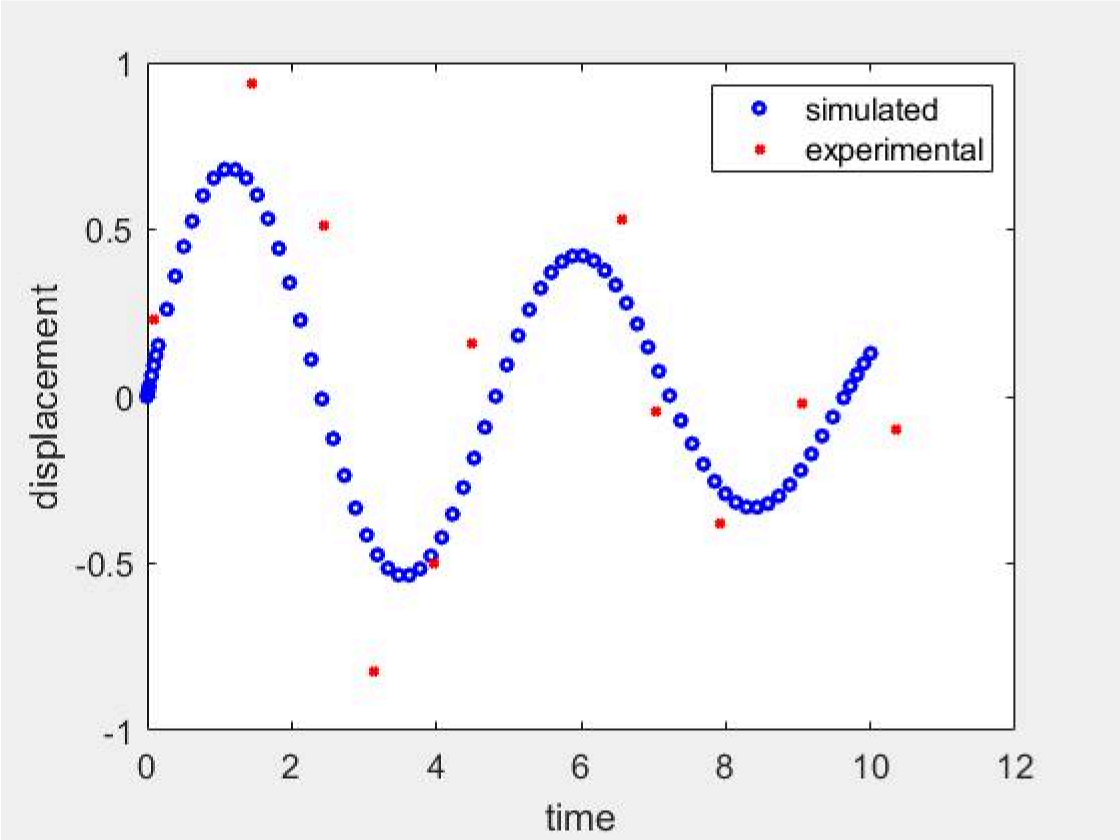
```
[xbest, besterror] = fminsearch(@massspringdamper_eval,[2 1],
 optimset('display','iter'))
```

```
 Iteration   Func-count     min f(x)         Procedure
     0            1          0.15828
     1            3          0.15828         initial simplex
     2            5         0.156659         expand
     3            7         0.150959         expand
     4            8         0.150959         reflect
     5            9         0.150959         reflect
     6           11         0.141016         expand
     7           12         0.141016         reflect
     8           14         0.131842         expand
     9           16         0.103639         expand
    10           18        0.0922316         reflect
    11           20        0.0824488         contract outside
    12           22        0.0824488         contract inside
    13           24        0.0824488         contract inside
    14           26        0.0809445         contract outside
```

```
    15          28         0.0808312         contract inside
    16          30         0.0807137         contract inside
    17          32         0.0804126         contract inside
    18          34         0.0804126         contract inside
    19          36         0.0804126         contract inside
    20          38         0.0803828         contract inside
    21          40         0.0803774         contract inside
    22          42          0.080376         contract inside
    23          44         0.0803735         contract inside
    24          46         0.0803714         contract inside
    25          48         0.0803714         contract inside
    26          50         0.0803709         contract outside
    27          52         0.0803708         contract inside
    28          54         0.0803708         contract inside
    29          56         0.0803707         contract inside
    30          58         0.0803707         contract inside
    31          60         0.0803707         contract inside
    32          62         0.0803707         contract outside
    33          64         0.0803707         contract inside
    34          66         0.0803707         contract inside

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX
 of 1.000000e-04
 and F(X) satisfies the convergence criteria using OPTIONS.TolFun of
 1.000000e-04


xbest =

    1.7095    0.1985


besterror =

    0.0804
```

Re-plot the optimized simulation results

```
[T,Y] = ode45(@(t,y)massspringdamper_diff(t,y, xbest), [0 10], [0 1]);
plot(T,Y(:,1),'ob', Tdata,Ydata,'xr', 'LineWidth',2,'MarkerSize',4);
xlabel('time'); ylabel('displacement');
 legend({'simulated','experimental'});
```

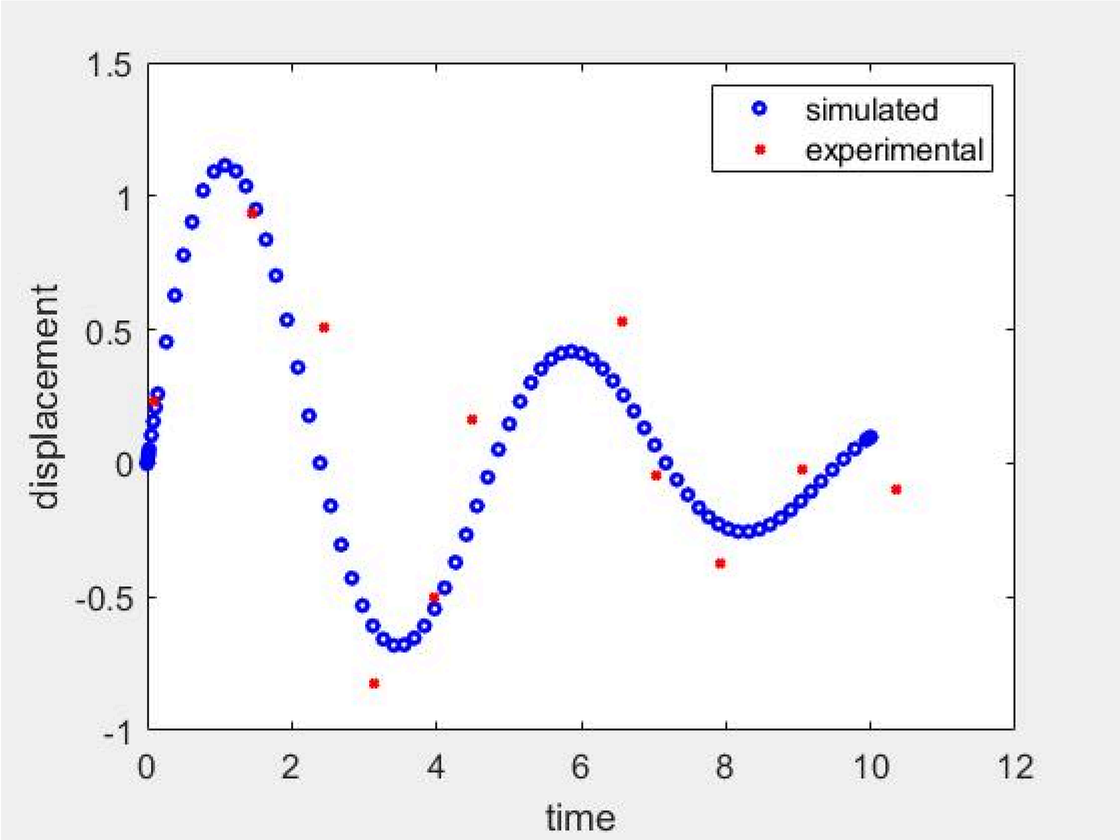


Optimize initial conditions as well

Optimizing the parameters improved the predictions, but it looks like our initial conditions also did not match the experimental data. Let's re-optimize, this time allowing the initial conditions to also change.

```
[xbest, besterror] = fminsearch(@massspringdamper_eval2,[0 1 2 1],
 optimset('display','none'))

[T,Y] = ode45(@(t,y)massspringdamper_diff(t,y, xbest(3:end)), [0 10],
 xbest(1:2));
plot(T,Y(:,1),'ob', Tdata,Ydata,'xr', 'LineWidth',2,'MarkerSize',4);
xlabel('time'); ylabel('displacement');
 legend({'simulated','experimental'});
```

```
xbest =

   -0.0017    1.8499    1.7694    0.4093


besterror =

    0.0619
```

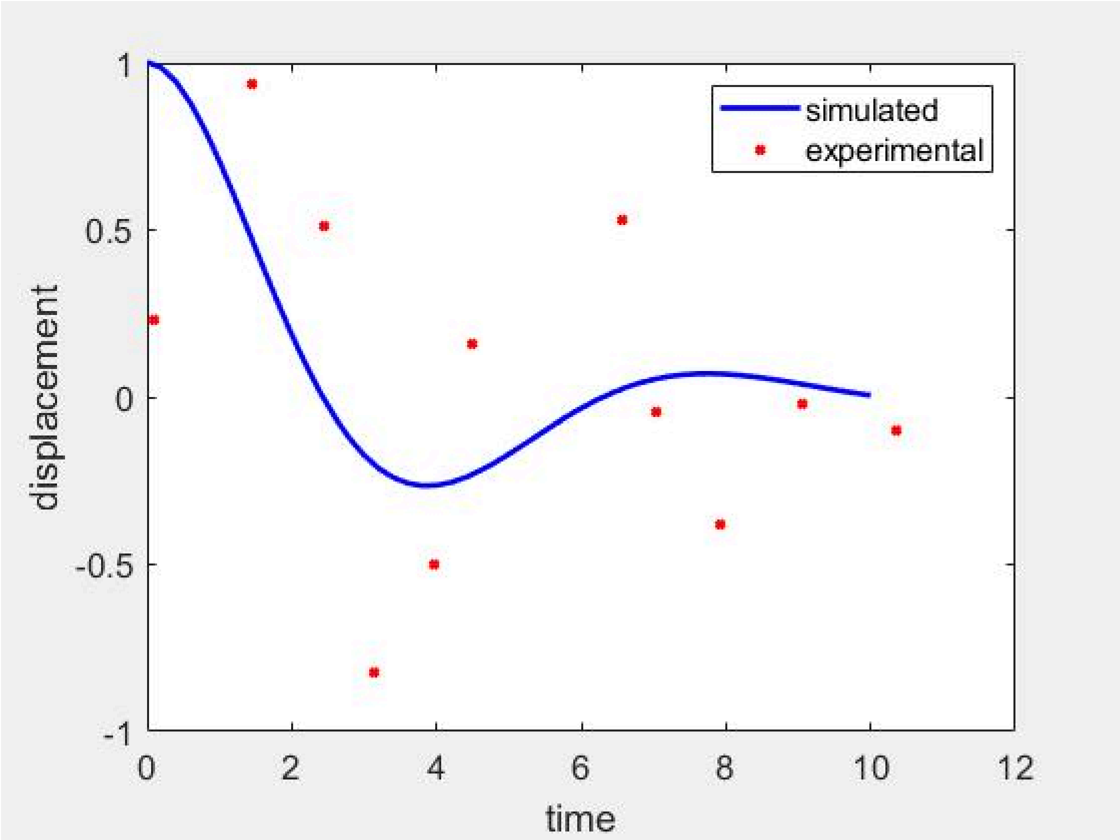


Optimize a Simulink Model

```
% Error before optimization.
error=massspringdamper_sim_eval([2 1])


error =

    0.2947


%error after optimization:
[xbest, besterror] = fminsearch(@massspringdamper_sim_eval,[2 1],
 optimset('display','none'))


xbest =

    0.7764    0.6868


besterror =

    0.1948


%call again to plot.
```

```
error=massspringdamper_sim_eval(xbest,true)
```

```
error =

    0.1948
```



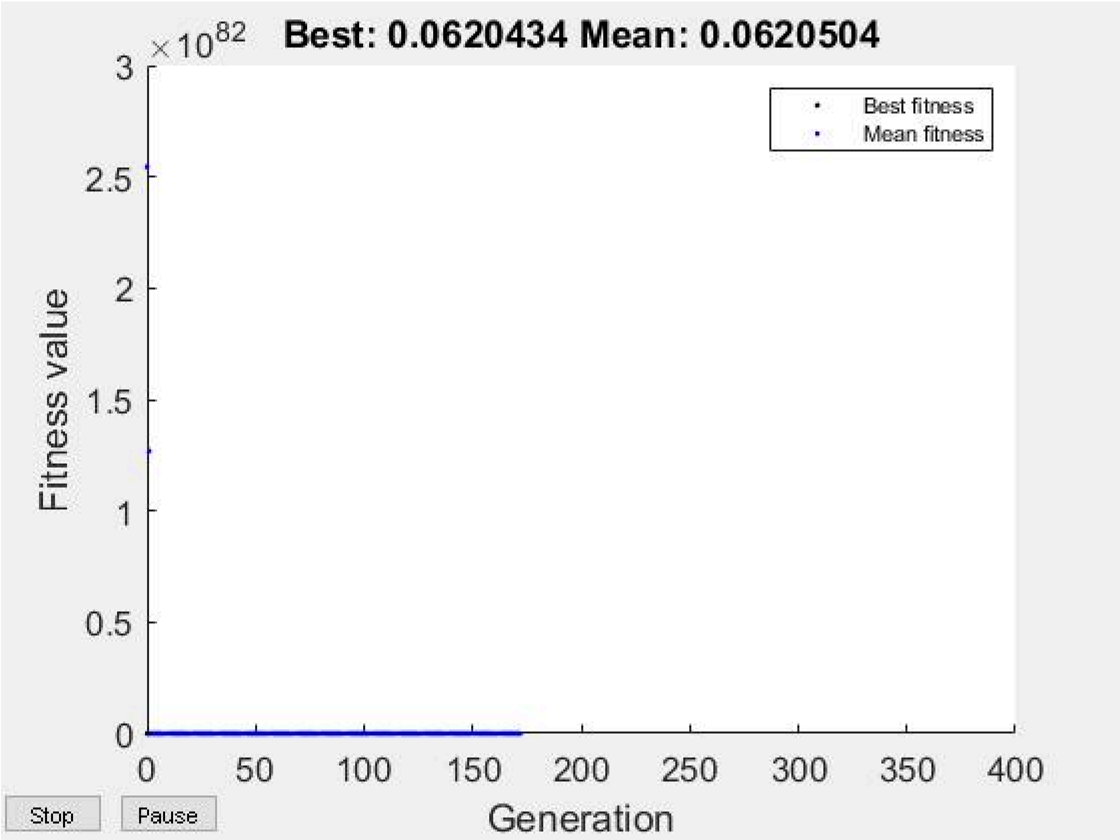
Error is larger with simulink, because x0=1, v0=0 are hard-coded into the simulink model. TODO: Change simulink model to allow us to vary x0,v0; write an objective function for params&initialconditions and repeat optimization with this new objective function.

Optimization with Genetic Algorithms

```
lowerbound=0;
upperbound=10;
[xbest, besterror] = ga(@massspringdamper_eval2,4,
[],[],[],[], lowerbound, upperbound, [],
 optimset('display','none','PlotFcn',@gaplotbestf))
```

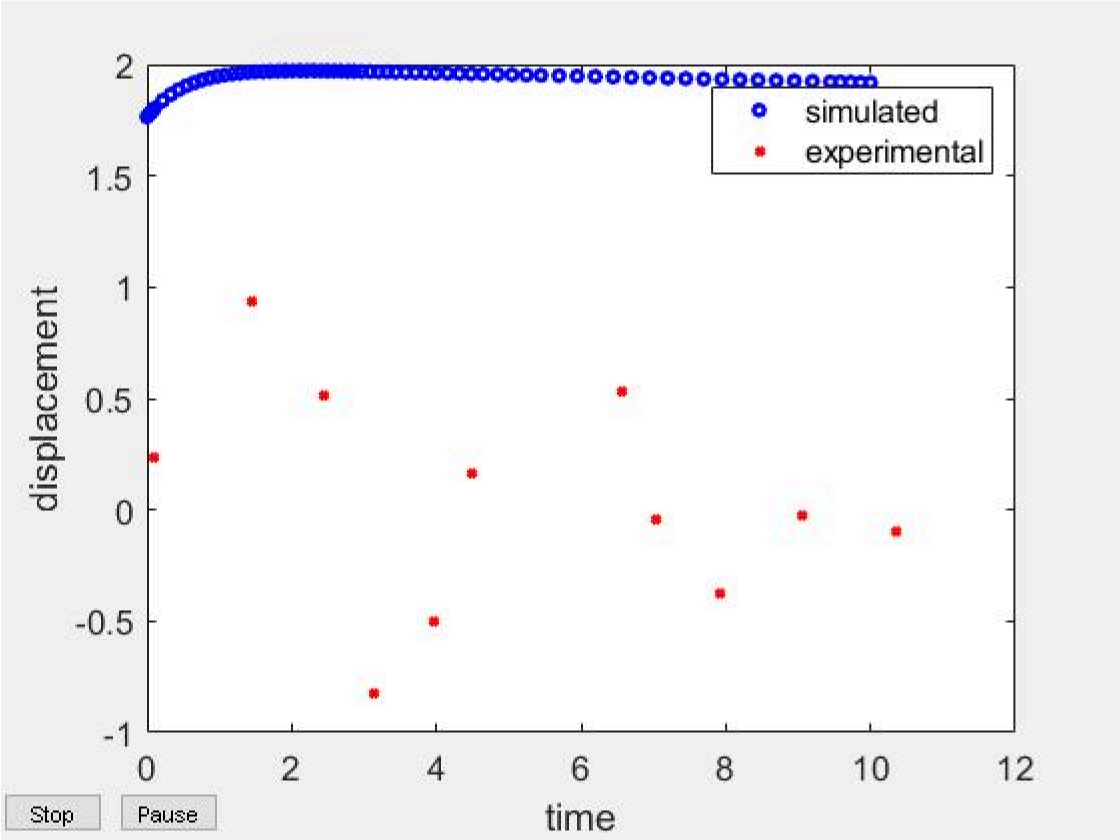
```
xbest =

    0.0069    1.8514    1.7645    0.4118
```

```
besterror =

    0.0620
```



Let's plot the optimized simulation of GA.

```
[T,Y] = ode45(@(t,y)massspringdamper_diff(t,y, xbest(1:2)), [0 10],
 xbest(3:4));
plot(T,Y(:,1),'ob', Tdata,Ydata,'xr', 'LineWidth',2,'MarkerSize',4);
xlabel('time'); ylabel('displacement');
 legend({'simulated','experimental'});
```



Auxiliary Functions

```
type massspringdamper_diff
```

```
function dy = massspringdamper_diff(t,y, params)
% Y: [x:displacement, v:velocity]
% params: [k, c]. assume mass is constant (m=1)
% by AhmetSacan

% dx: v
% dv: -(k/m)x - (c/m)v

%% Extract Species Values
% Y: [x:displacement, v:velocity]
x = y(1);
v = y(2);


%% Extract Parameter Values
% if params are provided, use them, otherwise define defaults here.
if ~exist('params','var')
 params = [1 1];
end
% params: [k, c]. assume mass is constant (m=1)
k = params(1);
```

```
c = params(2);
m = 1;


%% Differential Equations
dx = v;
dv = -k/m*x - c/m*v;


% Y: [x:displacement, v:velocity]
dy = [dx dv]';
```

```
type massspringdamper_eval
```

```
function error = massspringdamper_eval(params)
% params: [k, c].
% this is the objective function for a given params vector.
% We simulate the system for 10 seconds and compare the simulation
 data
% with experimental data; and return the error.
% TODO: pass in Tdata, Ydata to speed up the optimization.
% params: [k, c].
% by AhmetSacan

[T,Y] = ode45(@(t,y)massspringdamper_diff(t,y, params), [0 10], [0
 1]);
load('massspringdamper_data_grabbed.mat');
Tdata = massspringdamper_data_grabbed(:,1);
Ydata = massspringdamper_data_grabbed(:,2);
error=comparetwotimeseries(T,Y(:,1), Tdata,Ydata);
```

```
type massspringdamper_eval2
```

```
function error = massspringdamper_eval2(initsandparams)
% initsandparams: [x0, v0, k, c].
% this is the objective function for a given initialconditions &
 params vector.
% We simulate the system for 10 seconds and compare the simulation
 data
% with experimental data; and return the error.
% TODO: pass in Tdata, Ydata to speed up the optimization. Or make
 them
% persistent.
% by AhmetSacan

inits =initsandparams(1:2);
params=initsandparams(3:end);

[T,Y] = ode45(@(t,y)massspringdamper_diff(t,y, params), [0 10],
 inits);
load('massspringdamper_data_grabbed.mat');
Tdata = massspringdamper_data_grabbed(:,1);
```

```
Ydata = massspringdamper_data_grabbed(:,2);
error=comparetwotimeseries(T,Y(:,1), Tdata,Ydata);


type massspringdamper_sim_eval


function error=massspringdamper_sim_eval(params, doplot)
% params: [k, c].
% Simulate the Simulink model and return the error when compared to
% experimental data.
% by AhmetSacan

%This is where I have my simulink mode. You may not need this line of
 code
% if the slx file is in your current folder.
addpath([io_dirname(which(mfilename)) '/../simulink']);


k=params(1);
c=params(2);
m=1;

sim('massspringdamper',[],simset('SrcWorkspace','current'));

load('massspringdamper_data_grabbed.mat');
Tdata = massspringdamper_data_grabbed(:,1);
Ydata = massspringdamper_data_grabbed(:,2);
error=comparetwotimeseries(x.Time,x.Data, Tdata,Ydata);


if exist('doplot','var')&&doplot
 plot(x.Time,x.Data,'-b', Tdata,Ydata,'xr',
 'LineWidth',2,'MarkerSize',4);
 xlabel('time'); ylabel('displacement');
 legend({'simulated','experimental'});
end

type comparetwotimeseries


function [err]=comparetwotimeseries(T1,Y1,T2,Y2, doplot)
% Interpolate Y1 into T2 domain and compare it with Y2.
% if doplot=true, we do plotting for demonstration purposes.
%by AhmetSacan.

%remove any NaN's from data before interpolation. NaN's may result
 from
%divergent/unstable simulations.
I=any(isnan(Y1),2); T1(I)=[]; Y1(I,:)=[];
I=any(isnan(Y2),2); T2(I)=[]; Y2(I,:)=[];
if isempty(T1)||isempty(T2); err=inf; return; end

Y1interp=interp1(T1,Y1,T2,'pchip');
```

```
%I=~isnan(Y1interp);
%err=sum((Y1interp(I) - Y2(I)).^2)/nnz(I);
%ignoring nan's is being too nice. don't let nan's get away so easily.
errs = abs(Y1interp - Y2);
I=isnan(errs);
if any(I)
 if all(I); err=inf; return; end
 maxerr=max(max(errs(:)), max(Y2(:))-min(Y2(:))); %max/min ignore nan
 values.
 errs(I)=maxerr;
end

err=sum(errs(:).^2)/numel(errs);

if exist('doplot','var')&&doplot
 if size(Y1,2)~=1; error('doplot only available for single column
 Y.'); end
 plot(T1,Y1,'ob', T2,Y1interp,'*g', T2,Y2,'xr',
 'LineWidth',2,'MarkerSize',4);
 legend({'Y1','Y1interp','Y2'});
end
```

Published with MATLAB® R2020a