CITS2401 Computer Analysis and Visualisation

Semester 1, 2014

Assignment 2

Set date: 17 April 2014
Due date: 5 pm, 16 May 2014
Total marks: 30

Assignment submission instructions:

- Submit 2 files via the CITS2401 page on LMS

(http://www.lms.uwa.edu.au)

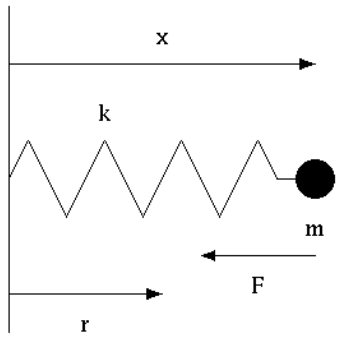
- Submit one Matlab function file called `spring.m` for Question 1
- Submit one Matlab function file called `massVar.m` for Question 2

Plagiarism

All submitted work should be your own. I am sure you agree that this is for your own good!! If you do not agree, please note that we have ways to detect plagiarism in code. Incidences of plagiarism will be taken seriously and will involve a follow-up with the Head of School, which will potentially affect your academic results.

Exercise 2: A two-spring system (20 marks)

Consider an object of mass *m* connected to the end of a spring:



If the object is pulled away from the rest position (*r*) of the spring, a force is created in the spring that will seek to pull the object back towards the spring. The spring will seek to pull the object back to the rest position of the spring.

Given a spring constant *k* in Nm and a rest displacement *r* in metres for a spring, the force pulling the object back towards the spring is given by:

$F = k(x - r)$

where *x* is the distance in metres the object is from the base of the spring.

If the object is released from this position, the object will shoot back towards the spring. The object will overshoot the rest position of the spring, travel through the base point of the spring (we will assume this is possible) and start to head away from the spring rest position again, this time on the opposite side of the release point.

As the object moves away from the spring rest position, the tension force in the spring increases and starts to pull the object back towards the spring rest position again. The object slows down, momentarily comes to a halt, and then starts to head back towards the spring rest position again. The object repeats its previous trajectory and continues to cycle between two extreme points. This cyclic motion is called *simple harmonic motion* and will continue forever if no other forces (like friction) act on the object.

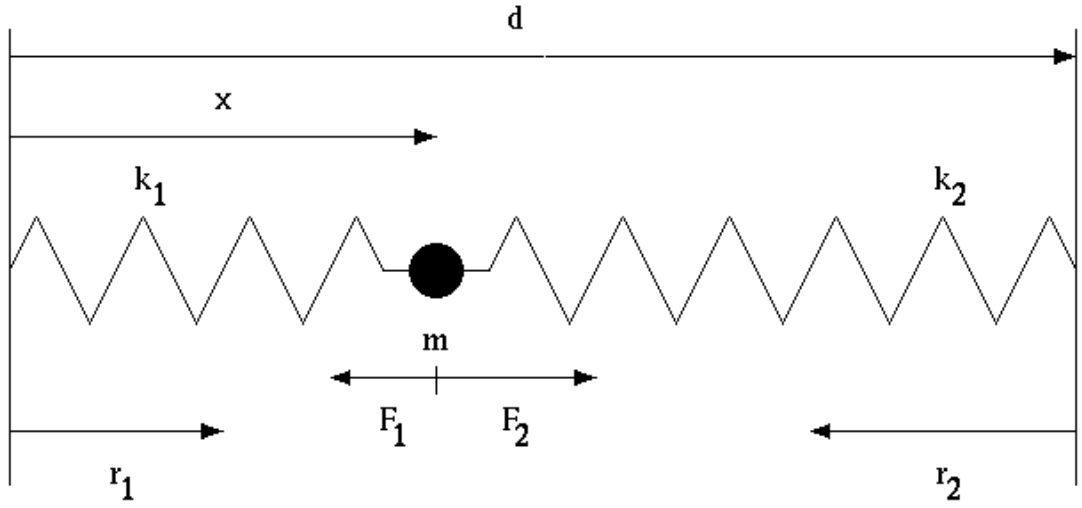
If other forces are considered, for every cycle some energy is lost from the system. Hence, the oscillatory motion dies out over time and the object eventually comes to rest at the rest displacement position of the spring.

A spring is said to be *damped* if the force of the spring is retarded by a function of the object's velocity. The *damping force* for this spring system is given by:

$F_{damping} = cv$

where c is the damping constant and v the velocity in ms^{-1} of the object.

Now consider the two spring system shown below:



If the object is pulled to some point and released, assuming the object is not already at its equilibrium point, the object will oscillate until an equilibrium point is reached. Since the object is released from rest, the initial velocity of the object will be zero.

If we define the origin of the two spring system to be positioned at the base of the left spring and define heading right as the positive x direction, the forces acting on the object are given by the following equations:

$F_1 = -k_1(x - r_1)$

$F_2 = k_2(d - x - r_2)$

The net force acting on the spring is the addition of these two forces minus the damping force:

$$F_{net} = -k_1(x - r_1) + k_2(d - x - r_2) - cv$$

The acceleration of the object is then given by:

$$F_{net} / m$$

The actual equation governing the motion of this system is a *differential equation*. Solving a differential equation precisely requires some additional higher maths, but we can still simulate the behaviour of the system if we make the following assumption:

For a very small time increment, the error produced by assuming that the acceleration during that time increment is constant and is small enough not to produce significant errors in the results.

If we accept the assumption (which we will without proof), we can now simulate the behaviour of the system by iterating until equilibrium is reached. Each iteration, we simply assume that the acceleration is constant and use the standard rectilinear motion equations to update the velocity and position of the object.

Every time step, we can calculate the new force and acceleration acting on the object using the equations given above. Once the acceleration is known, the new velocity of the object can be determined via the equation:

$$v = v_{old} + a\delta t$$

where v_{old} is the previous velocity in ms^{-1} and δt is the time increment in seconds.

The new location of the object is then given by:

$$x = x_{old} + v\delta t$$

where x_{old} is the previous location of the object in metres and δt is the time increment in seconds.

To simulate the position of the object over time, you will need to re-calculate F_{net}, a, v and x at each time interval. Because these values depend on each other, the order in which you update these values will affect the outcome. You should update these values in the following order, during each interval:

- F_{net}
- a
- v
- x

The process continues iterating until equilibrium is reached. Equilibrium in this case is defined as occurring when both the absolute velocity is less than some *velocity tolerance* and the absolute acceleration is less than some *acceleration tolerance*.

Write a function called `spring` with the following specification:

```
[x, t] = spring(ks, rs, startx, d, m, c, tinc, atol, vtol)
```

which will simulate the behaviour of this two spring system.

The `spring` function expects a number of arguments:

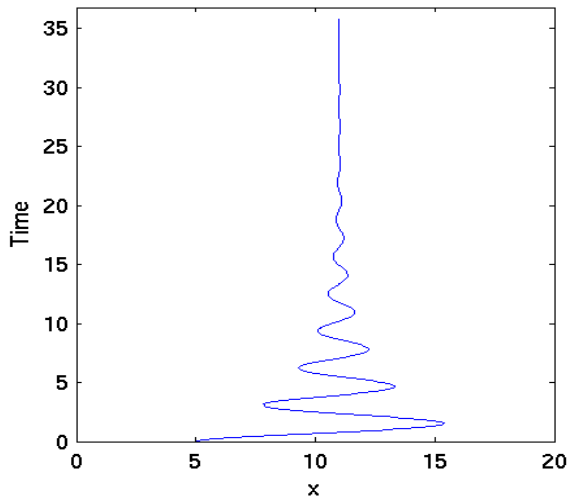
- *ks* - A 1x2 array containing the two spring constants (the left spring constant followed by the right spring constant).
- *rs* - A 1x2 array containing the two spring rest displacements (the left spring rest displacement followed by the right spring rest displacement).
- *startx* - A scalar defining the initial starting point of the object as measured from the base point of the first (left) spring.
- *d* - A scalar defining the distance between the two springs.
- *m* - A scalar defining the mass of the object.
- *c* - A scalar defining the damping constant of the system.
- *tinc* - A scalar defining the time increment to use in one iteration of the system.
- *atol* - A scalar defining the acceleration tolerance.
- *vtol* - A scalar defining the velocity tolerance.

The `spring` function should return two vectors *x* (the location of the spring at different time instances) and `t` (the time instances at which all computations were done). A quick way to make sure that the function is working properly is to plot the time `t` vs locations `x`. For example, the commands

```
>> [x, t] = spring([10, 10], [4, 2], 5, 20, 5, 1.5, 0.05, 0.001, 0.001);

>> plot(x, t);
```

Along with few more commands for plot labelling, following plot is produced:



The `spring` function should check the validity of its arguments, reporting an error if any of the arguments is invalid.

Hints:

- Make use of `abs` function when you are trying to determine when your code should stop iterating.
- Use a `while` loop to control the iterative process.

Exercise 2: A two-spring system plotting (10 marks)

From Exercise 1, it can be observed that variations in the mass values affect the behaviour of the two-spring system. In order to visualize this effect, write another function with the following specifications.

`res = massVar(mL1, mL2, ks, rs, startx, d, c, tinc, atol, vtol)`

This function should display a graph with four plots corresponding to the four different mass values. These mass values should be obtained from the parameters `mL1` and `mL2`, which are the two extreme mass limits (upper and lower or vice versa). Get four equally spaced values of `m` in the range between `mL1` and `mL2`

The `massVar` function requires the following inputs

- m_{L1} – lower or higher limit of mass
- m_{L2} – corresponds to the higher or lower limit of mass
- The other input variables are the same as in Exercise 1.

The `massVar` function should display a single graph with four separate plots, which correspond to the four different values of the mass that are equally spaced in the range of `mL1` and `mL2`.

Take care of the followings when writing your function:

1. All four plots should have the same axes for an easier comparison and analysis (the range of the plotted values on both axes in all four plots should be the same).
2. The function should also return a variable `res` which is a 1x4 vector with all the four values of the mass for which the two spring system response is plotted.
3. In the case of negative or zero values of the mass limits, the function should be terminated and it should display an error with a message explaining the reason of the error (for example, `the spring mass cannot be negative or zero. Please retry with positive values`).
4. The function should not be sensitive to the order of the mass limits.
5. Please label your plots properly, along with a title for every plot.

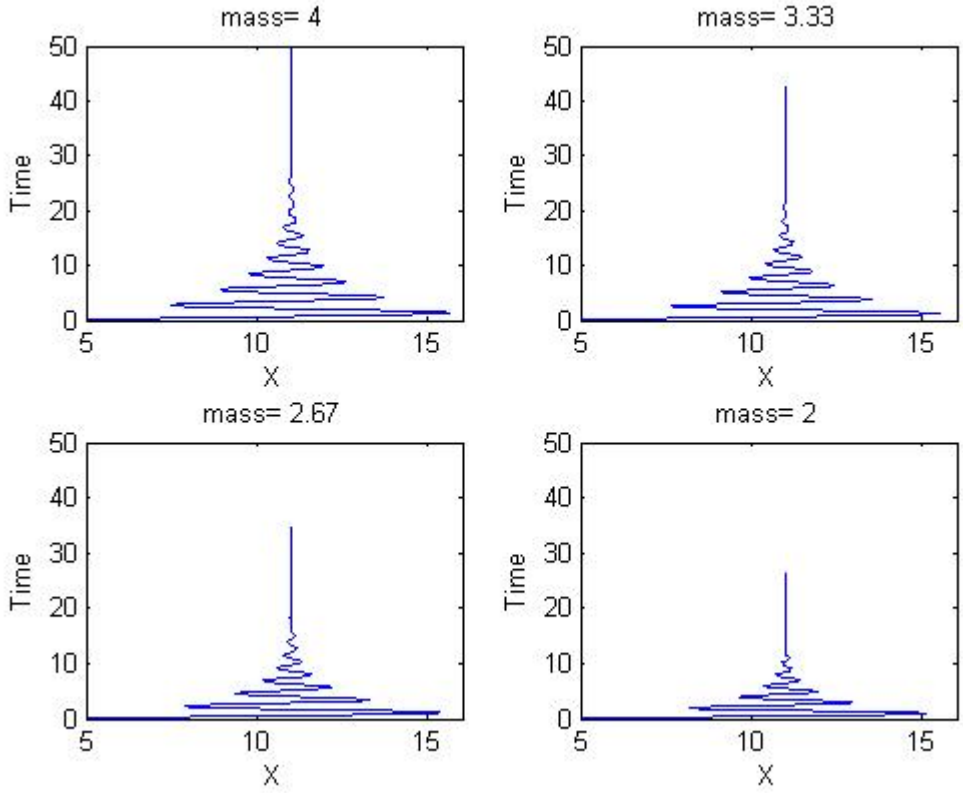
For example, the command

```
>> res = massVar( 2, 4, [10, 10], [4, 2], 5, 20, 1.5, 0.05, 0.001, 0.001)

res =

    4.0000    3.3333    2.6667    2.0000
```

produces the following plot



Hints:

- For `massVar` function, you will need to call your spring function.
- Use num2str to convert numbers to string