Time Slicing Assignment

Hans Vangheluwe

Fall Term 2000

The Tool

Implement (in Python) a modelling and simulation system composed of

- A causal block-diagram graphical model editor. The look as well as behaviour of basic building blocks (integrator, adder, ...) may be hard-coded. It must be possible to create/delete/move (instances of) basic building blocks as well as edit their parameters. Building blocks can be connected. It must be possible to save/load a model. Hint: export Python code which instantiates the appropriate graphical objects. This would be exactly the same code called from the GUI interactively. It must be possible to export a model suitable as input for a Time Slicing simulator. Hint: export Python code which instantiates classes in the Time Slicing simulator.
- A Time Slicing simulator (pseudo code given in class) which takes the output of the graphical model editor as input as well as "experiment" information such as
  - model initial conditions and parameters,
  - solver parameters such as step-size.
  - where output should go (file of output processor).

  Structure the simulator so it is easily embedded in different scripts (such as optimization).
- A graphical output (plot) of generated trajectories. Both time- and phase- plots are needed.

The "Circle Test"

Test your time-slicing simulator by means of the equation $\frac{d^2x}{dt^2} = -x$. When x and $\frac{dx}{dt}$ are plotted in function of one another (a phase plot), a circle should result.

1. Re-write the equation in the form of a set of first order equations.
2. Draw the corresponding block-diagram in the graphical editor.
3. Run the simulation and plot the data in time and phase plots.
4. do the above for a "good" (sufficiently small) integrator step-size as well as for a "bad" (large) step-size (which should not result in a circle plot). Explain. The "good" step-size will give you an idea of what step-size to use for the rest of the assignment. Explain why it should be smaller than the value you used for the circle test.

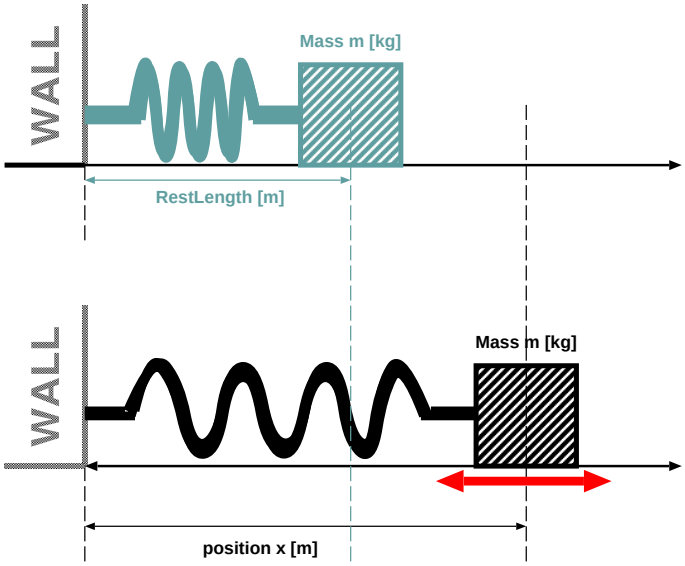

Figure 1: Mass-spring system

Mass-spring: simulation and calibration

The above mechanical system consists of a mass m which glides without friction over a surface. The mass is connected to a rigid wall by means of an "ideal" spring. In the absence of external forces, the system is in "rest" state and the distance of the centre of gravity of the mass object to the wall is *RestLength*. At any instant in time, the position (distance from the wall) of the mass is x.

An experiment has been carried out whereby the mass m was measured as well as the *RestLength* of the spring. $m = 0.23kg$, $RestLength = 0.2m$. To determine the spring constant $K[kg/s^2]$ of the ideal spring, the spring is extended to bring the mass at initial position $x(t = 0)$ with initial velocity $v(t = 0)$. $x(t = 0) = 0.3m$, $v(t = 0) = 0m/s$. (note: in many cases, in a simulation, one may have to set x(t=0) and/or v(t=0) to a small, non-zero value to avoid the simulator providing a trivial (zero) solution to the system equations).

This experiment whereby the mass is released and observed during the time interval $[0, 4[$ yields the following measurement data $x_{measured}$ in function of time.

Note: this plot was produced in gnuplot from the $x_{measured}$ data file (after removal of the first line) with the following commands:

```
set xlabel "time t [s]"</FONT>
set ylabel "position x [m]"</FONT>
plot "data" title 'measured position x'
```

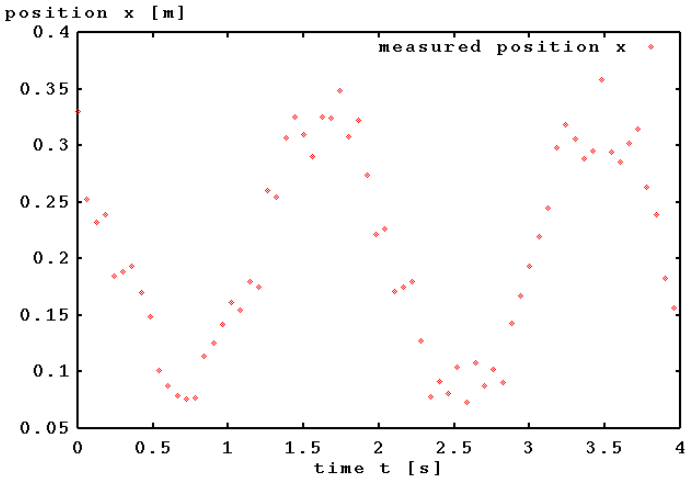

Figure 2: Measured displacement (noisy)

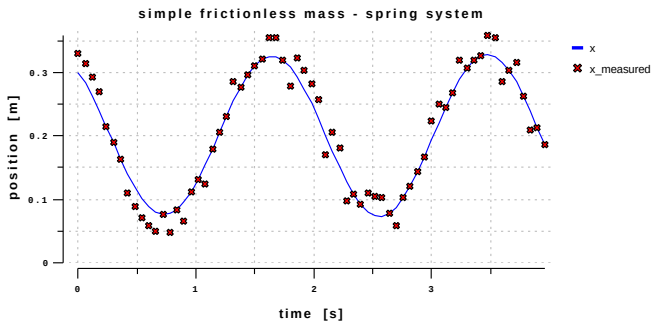

Figure 3: Calibrated model output

When you want a smooth curve rather than points, append `with lines` to the plot command. To plot column `B` of the data file in function of column `A`, append `using A:B` to the plot command.

With this "noisy" data, we need to "estimate" spring constant value K which, when used in a simulation of the dynamics of the system $x(t)$ optimally "fits" the measured data. Notice how we start with parameter estimation directly and we skip the "structure characterization phase" in which the most appropriate mathematical model for this system is determined. This, as we have the a-priori knowledge that this is a frictionless system and the spring is "ideal".

Assignment:

1. Describe the mathematical equations for the dynamics of this system (given the above a-priori knowledge).
2. As this will yield a higher order differential equation, rewrite this as a set of first order differential equations.
3. Represent this set of differential equations as a causal block diagram.
4. Represent this block diagram in your tool.
5. Use this as input to your time-slicing simulator.
6. run multiple simulations, varying (with a small enough step-size) K value in $[1, 10]$.
7. Check which of the K values gives the "best fit". Fit is defined in the "sum of squared errors" sense. Hereby, for each measured point in time, the difference between the measured value and the simulated value is taken and squared. The sum of all squared errors is a measure for the fit.

   Note: to give accurate results, the simulator may need a small step-size. To compare with measured data which is quite far apart, you simulator will have to implement a "communication interval" which allows the user to specify how often simulated values have to be output.

   Whether you use a very naive exhaustive search as described above or an advanced optimization algorithm (feel free to apply some of your optimization knowledge), an optimal K will result. Simulation with the "true" K ¿value will yield a graph as below.

Report

The full analysis, design (using UML notation), implementation (in Python) and simulation results should be documented an put on the web. Explicit links to code and data must be present.