MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.081—Introduction to EECS I
Spring Semester, 2007

Week 4 Software Lab

Issued: Thursday, March 1st

This handout contains:

- Software Lab for Thursday, March 1
- Post-lab exercises due Tuesday March 6

Thursday Software Lab: Speed of difference equation solving

As we've seen, a kth-order linear difference equation is an equation of the form:

$$y[n] = a_1 y[n-1] + \ldots + a_k y[n-k] \quad .$$

To fully specify the behavior of a difference equation, we need to supply the coefficients $a_1 \ldots a_k$ and the initial values $y[0] \ldots y[k]$. The lab for today will be to examine and modify an implementation of a difference equation class, and use it to develop a feel for orders of growth.

Download the *diffeq.py* file from the 6.081 website and examine the difference equation class. To generate an object whose difference equation generates the Fibonacci numbers, as in

$$y[n] = y[n-1] + y[n-2],$$

one can load diffeq.py and then type the following command in the interpreter:

```
>>> fib = DifferenceEquation([0, 1], [1, 1])
```

If you examine the *DifferenceEquation* class, you will notice that two methods are implemented for computing the n^{th} value of the difference equation, one iterative and one recursive. To insure you understand the class implementation, please answer the following questions.

Question 1. What does the line *vals = vals[1:] + [nextVal]* in the *valIter* function accomplish?

Question 2. What does the line

```
dot(self.coeffs, [self.valRecur(n-i-1) for i in range(self.order)])
```

in the *valRecur* function accomplish?

A clever plot

It can often be revealing to plot the series of values a difference equation generates. In order to use SoaR's plotting features, recall that you must import the needed functions by typing the following lines in the interpreter:

```
import SoaR
from SoaR.Util.GraphingWindow import GraphingWindow
```

Once you have imported the functions from SoaR, you can plot the values of a function by opening up a GraphingWindow and then using the graphDiscrete function. For example, to plot the values of the function f from $n = 0$ to $n = 7$ on a graph whose y axis ranges from -10 to 10, one would type

```
plotWindow = GraphingWindow(500, 500, 0, 7, -10, 10, "Plot")
plotWindow.graphDiscrete(f)
plotWindow.helpIdle()
```

Question 3. Experiment with the following difference equations:

```
fib = DifferenceEquation([0, 1], [1, 1])
fab = DifferenceEquation([0, 1], [1, -1])
fob = DifferenceEquation([0, 1], [-1, 1])
fub = DifferenceEquation([0, 1], [-1, -1])
```

Plot the results (just go up to $n = 10$ or $n = 20$), and make sure you understand why you got the values you got. Explain it to your LA.

To get a feeling for how the time for the two difference equation solution methods increases with n, it is helpful to graph time versus n for the iterative and recursive methods.

Question 4. Generate graphs of compute time versus n for the two difference equation solution methods applied to the Fibonacci difference equation, and explain your results. You will find the function *timef* in *diffeq.py* helpful (think about what *graphDiscrete* takes as input). A word to the wise: start with a small-ish (10) `n` and work your way up.

Question 5. How do your graphs change if you try a third-order difference equation (such as $y[n] = y[n-1] + y[n-2] + y[n-3]$)? Why?

One strategy for speeding up the recursive procedure is to *memoize* the computation of values. One could add a new method to the DifferenceEquation class, `valMemo`, which uses the recursive method with memoization. That is, one can add a list of stored values to the class, update the stored list for each as yet unseen n, and use values from the stored list for already seen n's. Such a "by-hand" memoization is not as elegant as the generic approach described in lecture, but is far easier to understand.

Question 6. Add `valMemo` to the DifferenceEquation class, generate graphs of compute time versus n for your new solution methods applied to the Fibonacci difference equation, and then compare the results to using `valRecur`.

Due in lecture on March 6

Hand in written answers to the above questions, including your `valMemo` program, along with the Tuesday February 27th Robot Lab write-up. Both are due Tuesday March 6th.