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

Assignment Week 5, Issued: Tuesday, Oct. 2

To do this week: NOTE: TWO LECTURES, TWO SOFTWARE LABS!

...in Tuesday software lab, 10/2

1. Start writing code and test cases for the numbered questions in the software lab. Paste all your code, including your test cases, into the box provided in the "Software Lab" (Part 5.1) problem on the on-line Tutor. This will not be graded.

...before lecture on Thursday, 10/4

1. Nothing.

...in Thursday software lab, 10/4

1. Start writing code and test cases for the numbered questions in the software lab. Paste all your code, including your test cases, into the box provided in the "Software Lab" (Part 5.2) problem on the on-line Tutor. This will not be graded.
2. Get the handout for next Thursday's lab.

...No Lecture Tuesday, 10/9

...before the lab Thursday, 10/11

1. Do the on-line Tutor problems for week 5 that are due on Thursday (Part 5.3).
2. Do the writeup for the 10/2 and 10/4 software labs (in this handout) providing written answers (including code and test cases) for **every** numbered question in this handout.

...before lecture Tuesday, 10/16

1. Do the writeup for the 10/11 design lab (NOT in this handout).
2. Do the nanoquiz; it will be based on the material in the lecture notes and the on-line Tutor problems due on Thursday.

On Athena or the lab laptops make sure you execute:
`athrun 6.01 update`
so that you can get the `Desktop/6.01/lab5` directory which has the files mentioned in this handout.

- You need the file `differenceEquationWithInput.py`.
- You need the file `soar-graph.py`.

During this lab, please use the lab laptop for Tuesday's software lab. If you are using your own laptop for Thursday's software lab, download the files from the course Web site (on the Calendar page).

Tuesday Software Lab: Speed of difference equation solving

In this lab, we will be using python's features for timing software. Since our experience is that the timing functions are not very portable, please work in pairs and use the lab laptops.

In earlier lectures, you learned that the solution to a homogeneous difference equation can be represented in closed form using the equation's natural frequencies. In today's lecture you learned that when a difference equation has an input, that is the equation is not homogeneous, the solution is more complicated to compute and that computation is probably best accomplished using a program.

One general form for a difference equation with input (we will see another form just below) is

$$y(n) = \alpha_1 y(n-1) + \alpha_2 y(n-2) + \ldots \alpha_K y(n-K) + \beta_o x(n) + \beta_1 x(n-1) + \ldots \beta_L x(n-L)$$

where $x(n)$ is the input and is a given sequence. To determine $y(n)$ from the above difference equation, it is necessary to specify the values of the coefficients $\alpha_1 \ldots \alpha_K$ and $\beta_o \ldots \beta_L$, the initial values $y(0) \ldots y(K-1)$, and value of $x(n)$ for all n.

One way to specify the input x is use a procedure. For example, consider defining $x(n)$ using the python procedure

```
def x(n):
    return 1
```

In this case $x(n) = 1$ for all n.

In today's lab we will be examining and modifying an implementation of a difference equation class, and will then use that class to develop a feel for orders of growth. Examine the difference equation class defined in `differenceEquationWithInput.py`. You will notice that the difference class uses a different format for specifying a difference equation, one that turns out to be much more easily manipulated when using transform techniques to be described in a later lecture. Specifically, the difference equation format used in the difference equation class is

$$\sum_{k=0}^{K} a_k y(n+k) = \sum_{l=0}^{L} b_l x(n+l).$$

A note should be made about the order of the coefficients. In the polynomial class, the coefficients for a polynomial $z^2 + 2z + 3$ would be given as a list $[1, 2, 3]$. In following that convention, the

coefficients of a difference equation go from higher to lower order, so that a list of the y coefficients would be $[a_k, a_{k-1}, ..., a_o]$. Note that this same seemingly backward convention is true for initial conditions as well, the intial values are $[y(K-1), .., y(0)]$. This may seem confusing, as the k^{th} coefficient of a difference equation is not the k^{th} element of the list. We agree, but the common convention is against us.

Consider the Fibonacci example. In order to create an instance of the difference equation class that can be used to compute the Fibonacci numbers, it is necessary to create an instance associated with the difference equation

$$y(n) = y(n-1) + y(n-2).$$

Here, there is no input, and therefore $x(n) = 0$ for all n. Loading differenceEquationWithInput.py and then typing the following command in the interpreter:

```
>>> fib = DifferenceEquationWithInput([1, 0], [1, -1, -1], [1], lambda n: 0)
```

will create an difference equation instance which will generate the Fibonacci numbers.

Question 1. How are the α_k's and β_l's related to the a_k's and the b_l's in the above two forms of difference equations?

If you examine the *DifferenceEquationWithInput* class, you will notice that two methods are implemented for computing the n^{th} value of the solution to the difference equation. One method uses iteration, and the other uses recursion. To test your understanding of the class implementation, please answer the following questions.

Question 2. What does the line *vals = [nextVal] + vals[:-1]* in the *valIter* function accomplish?

Question 3. What does the line

```
dotProd(self.outCoefficients[1:], [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. Please refer to the week 4 assignment for instructions on how to plot using SoaR (the assignment is on the course calendar page if you do not have it handy).

Question 4. Demonstrate that you understand how the difference equation class works by using the class to solve and plot the solution to the difference equation $y(n) = 0.5y(n-1)+x(n-1)$ where $y(0) = 0$ and $x(n) = 1$ for all n.

Question 5. Now use the difference equation class to solve and plot the solution to the difference equation $y(n) = -0.5y(n-1) + x(n-1)$ where $y(0) = 0$ and $x(n) = 1$ for all n.

Question 6. Explain why the two plots look the way they do, and explain the differences between the two plots.

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 7. 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 *differenceEquationWithInput.py* helpful (think about what *graphDiscrete* takes as input). A word to the wise: start with a small value of `n`, say around 10, and work your way up. Before generating a plot, determine the right `y` axis range to use. Note that the time required for `valIter` and `valRecur` to complete are very different.

Question 8. How would 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.

Question 9. Add `valMemo` to the DifferenceEquation class. Examine how the compute time of your memoized version of *valRecur* compares with the compute time for the original `valRecur` and `valIter`.

Question 10. When you add memoization to a function, the function then has internal state. When a function has internal state, it may not behave exactly the same way when called multiple times with the same argument. How will the fact that *valMemo* function has state impact the approach you use to generate a graph of the complexity of your memoized function?

Thursday Software Lab: Z Transform software

Last week, you used a simple feedback scheme for controlling the robot so that it drove down the center of a narrow corridor at a constant velocity. The scheme we suggested was that you make the robot's rotational velocity equal to the product of a "gain", `K`, and the measured distance from the center of the corridor. You then developed a mathematical model for the controller in the form of a homogeneous difference equation. An analysis of how the natural frequencies of the homogeneous difference equation changed with `K` suggested that the simple feedback scheme would not work, regardless of the value of `K`. It should be noted that the analysis of the difference equation model gave a great deal of insight, but keep in mind that the model is approximate, and may not predict robot behavior exactly.

Next week you will revisit the robot feedback control problem, but this time you will use the more sophisticated manipulation techniques associated with the `Z` transform to help you design a more effective controller. This controller should also allow you to change the robot's distance from the walls as it proceeds down the corridor.

As is often the case, the first step will be to develop some useful tools in Python, and then examine some model results before working with the robot. In particular, you will develop methods for manipulating transfer functions and methods for computing difference equations and natural frequencies from the transfer functions.

Transfer function class

As discussed in lecture, using Z-transforms makes it possible to specify primitive transfer functions as ratios of polynomials, and to combine transfer functions via serial and parallel composition, as well as by feedback. To effectively perform these compositions, you will design a class to represent transfer functions and to perform these compositions. In addition, you will provide a methods to convert between a transfer representation and a difference equation representation.

To begin, consider starting from a difference equation in the form

$$\sum_{k=0}^{K} a_k y(n+k) = \sum_{l=0}^{L} b_l x(n+l).$$

Recall that the transfer function relates the Z transform of x to the Z transform of y, as in

$$\tilde{Y}(z) = \tilde{H}(z)\tilde{X}(z)$$

where $\tilde{Y}(z) = \sum_{-\infty}^{\infty} y(n)z^{-n}$, $\tilde{X}(z) = \sum_{-\infty}^{\infty} x(n)z^{-n}$, and

$$\tilde{H}(z) = \frac{\sum_{l=0}^{L} b_l z^l}{\sum_{k=0}^{K} a_k z^k}.$$

is the transfer function.

As we saw in lecture, most ways of combining difference equations correspond to very simple manipulations of the associated transfer functions. For this post-lecture lab, you will write a python transfer function class which automates these manipulations.

You may find it easiest to represent your transfer function as a ratio of polynomials in z, as in

$$\frac{\sum_{l=0}^{L} b_l z^l}{\sum_{k=0}^{K} a_k z^k}.$$

Functions which are ratios of polynomials are referred to as rational functions.

Please write your transfer function class so that it can create, compose and manipulate transfer functions.

Question 11. Your transfer function class should be able to:

- create a transfer function from lists of numerator and denominator polynomial coefficients,
- create a new transfer function from the product of two transfer functions,
- create a new transfer function using a Black's formula transformation,
- compute the natural frequencies associated with a transfer function,
- take initial conditions and an input-defining procedure and create an instance of the *differenceEquationWithInput* class.

Note that for the purposes of this lab, we will assume the Black's formula tranformation of a transfer function is given by

$$\tilde{H}_B(z) = \frac{\tilde{H}(z)}{1 + \tilde{H}(z)}.$$

where $\tilde{H}(z)$ is the original transfer function and $\tilde{H}_B(z)$ is the transformed result. Please be sure that for your class, manipulations produce a rational function when appropriate.

You will likely find it easiest to represent the transfer functions as numerator and denominator polynomials, using the Polynomial class we gave you. The polynomial class has functions for adding and scaling polynomials, finding roots of the polynomials, and for polynomial multiplication. Recall that you can see the class interface by typing

```
help(Polynomial)
```

at the IDLE command line. We have given you a file in the `lab5` directory called `differenceEquationWithInput.py`. The file contains a definition of a difference equation class. You do not need to understand all its detail, but you should understand how to use its methods. You should add your definition of a `TransferFunction` class to this file. Note that the `z` method of the `DifferenceEquationWithInput` is intended to create an instance of the `TransferFunction` class. It may seem a little strange at first, but the idea that an instance of `DifferenceEquationWithInput` can create an associated transfer function instance, and vice-versa, is really not that surprising. We can convert back and forth between difference equations and transfer functions, and our software should be able to do the same.

As an example to test your program (you should try more than one), consider the two transfer functions

$$H_1(z) = \frac{z+1}{z+2} \quad H_2(z) = \frac{2z+1}{z+3}.$$

The product of the two transfer functions is

$$H_3(z) = H_1(z)H_2(z) = \frac{2z^2+3z+1}{z^2+5z+6}.$$

The Black's formula transformation of $H_3(z)$ is

$$H_4(z) = \frac{H_3(z)}{1+H_3(z)} = \frac{2z^2+3z+1}{3z^2+8z+7}.$$

Finally, the natural frequencies of $H_4(z)$ are $-\frac{4}{3}+j\frac{\sqrt{5}}{3}$ and $-\frac{4}{3}-j\frac{\sqrt{5}}{3}$. Here is a sequence of Python commands that you ought to be able to execute, once you've implemented this class, that will solve this example.

```
h1 = TransferFunction([1,1],[1,2])
>>> h2 = TransferFunction([2,1],[1,3])
>>> h3 = h1.compose(h2)
>>> h4 = h3.feedback()
>>> h4.naturalFreqs()
[(-1.3333333333333333+0.7453559924999299j),
(-1.3333333333333333-0.7453559924999299j)]
```

Then, if you want to see what happens when you feed a simple input signal (first value is 1 and subsequent values are 0) into this system, you can do:

```
>>> de = h4.diffEq([0,1], unitSample)
('inits', [0, 1], 'yCoeffs', [3.0, 8.0, 7.0], 'xCoeffs', [2.0, 3.0, 1.0])
>>> [de.valIter(n) for n in range(10)]
[1, 0, -2.0, 5.333333333333333, -9.5555555555555554, 13.037037037037038, -12.469135802
```

Question 12. Demonstrate that your program is correct on the above test example and then try it on an example of your own construction. Verify your result by comparing to hand computation.

Post-Lab Writeup for Tuesday and Thursday's labs: Due before lab lecture on October 11

As usual, both post-labs should be written in clear English sentences and paragraphs and we expect at least a couple of sentences or a paragraph in answer to **all** of the numbered questions in this lab handout. We also want the code you wrote for Tuesday's and Thursday's labs.

Concepts covered in this assignment

Here are the important points covered in this assignment:

- Learn about difference equations with inputs.
- Learn a little about computational complexity.
- Learn about accelerating recursion with memoization.
- Learn about transfer functions and how to manipulate them