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

Assignment 4, Issued: Tuesday, Sept. 25

To do this week

...in Tuesday software lab

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 4.1) problem on the on-line Tutor. This will not be graded.

...before the start of lab on Thursday

1. Read the lecture notes.
2. Do the on-line Tutor problems for week 4 that are due on Thursday (Part 4.2).
3. Read through the entire description of Thursday's lab.

...in Thursday robot lab

1. Answer the numbered questions in the robot lab and demonstrate them to your LA.
2. Do the nanoquiz; it will be based on the material in the lecture notes and the on-line Tutor problems due on Thursday.

...before the start of lecture next Tuesday

1. Do the lab writeup, providing written answers (including code and test cases) for **every** numbered question in this handout.

On Athena or the lab laptops make sure you execute:
`athrun 6.01 update`
so that you can get the `Desktop/6.01/lab4` directory which has the files mentioned in this handout.

- You need the file `polynomial.pyc` for the software lab.
- You need the file `diffeq.py` for the software lab.
- You need the file `soar-graph.py` for the software lab.

During software lab, if you are using your own laptop, download the files from the course Web site (on the Calendar page). Make sure you are using Python version 2.5, as we are giving you a "compiled" version of the polynomial class.

Tuesday Software Lab 1: Solving second-order difference equations

For this lab, you will be writing a Python program that solves arbitrary second-order homogeneous linear difference equations analytically, by computing natural frequencies. Your program should take as inputs the initial values x(0) and x(1) as well as coefficients a_1 and a_2 for the difference equation in the form

$$y(n) = a_1 y(n-1) + a_2 y(n-2).$$

Your program should print out the solution to the difference equation and also return a procedure that, when called with n, returns y(n). For example, if the difference equation is

$$y(n) = 2y(n-1) - 2y(n-2),$$

and the initial conditions are $y(0) = 0$, $y(1) = 1$, your program should print something equivalent to

```
y(n) = (2.7755575e-017+0.5j)*(1-1j)**n  + (-2.7755575e-017-0.5j)*(1+1j)**n
magnitude of natural frequencies: 1.4142135623730951, 1.4142135623730951
```

and your program should *also* return a function which can be used to evaluate y(n) for any n. **NOTE:** you can assume that the natural frequencies are distinct (but your program should indicate an error when the two natural frequencies are identical). The special case of what to do with repeated natural frequencies is studied in more advanced courses.

Polynomial manipulation

To save time, you should use our implementation of the polynomial manipulation program from the recent post-lab exercises and edit *diffeq.py* to create your second-order difference equation solver. Please be sure *diffeq.py* and *polynomial.pyc* are in the same directory as your difference equation solver. Executing *diffeq.py* in IDLE will import our version of the polynomial manipulation routines implemented in the class Polynomial. You can see the class interface by typing

```
help(Polynomial)
```

at the IDLE command line. Note that the Polynomial class has functions to add, multiply, print and find the roots of polynomials. The following code fragment

```
p = Polynomial([1, -6, 9])
>>> p.roots()
[(3+0j), (3-0j)]
```

generates an instance of the polynomial

$$x^2 - 6x + 9 = 0 \ ,$$

and then computes the polynomial's roots. Our implementation only computes roots correctly for polynomials up to third order (cubics).

Checkpoint: 11:45

- You should understand the polynomial class.

A note on complex numbers

For this problem, your procedure will sometimes need to compute z^n where z is a complex number. Python understands complex numbers to some extent, but you have to write them in the form

$$a + bj$$

where `a` is the real part and `b` is the imaginary part. Note that Python uses the convention that $j = \sqrt{-1}$. As an EECS student, you will have to learn to accept the fact that the variable `i` must be reserved for electrical current.

You can't raise a negative real number to a fractional power, but you can do so with a complex number. So, for example, the Python command

```
>>> (-4)**0.5
```

will produce an error, but the Python command

```
>>> (-4+0j)**0.5
```

produces `(1.2246063538223773e-016+2j)`. (Why the very small real part?)

Note: use `y**0.5` to compute the square root of a number. The python `sqrt` function does not understand complex numbers. You can get the real and imaginary parts of a complex number `x` with `x.real` and `x.imag`; also, `abs(x)` will return its magnitude.

It will be important to ensure that the function returned by your difference equation solver only returns real (not complex) values. Be sure you understand why, mathematically, the output should always be real if the coefficients and initial values are real.