Software Lab 5 *6.01 – Fall 2011*

System Functions

Goals: Study properties of linear time-invariant systems, by developing software modules for their system functions. Specifically, we:

- Implement a SystemFunction class for representing LTI system functions as ratios of polynomials in $\mathcal{R}$.
- Develop operations for combining system functions.

1 Setup

Using your own laptop

- Be sure you have the 6.01 software libraries installed.
- Download and unzip `swLab05.zip` into a convenient folder (e.g., `~/Desktop/6.01/swLab05`).

2 System Function Class

Objective: Implement the python `SystemFunction` class for representing system functions, and determining basic properties of the system, based on its poles.

Resources:

- Examples showing the typical use of this class (in **Section 4** of this handout)
- **Section 5** of the course notes.
- Documentation for the **`sf` module**.
- `sfSkeleton.py`: template for developing your `SystemFunction` class.
- `swLabs05Work.py`: template for working with the `SystemFunction` class.

Some of the software and design labs contain the command athrun 6.01 getFiles. Please disregard this instruction; the same files are available on the 6.01 OCW Scholar site as a .zip file, labeled Code for [Design or Software Lab number].

A LTI system may be represented in terms of its system function, which is given by a ratio of polynomials in $\mathcal{R}$ (the "Right shift" operator, a unit delay). We can model this in Python with a `SystemFunction` class which has the following methods:

- `__init__(self, numeratorPoly, denominatorPoly)`: takes two instances of the `Polynomial` **class** as input and stores them in this `SystemFunction` instance as the attributes `numerator` and `denominator`.
- `poles(self)`: returns a list of the poles of the system. Remember that the poles are the roots of the polynomial in z, where $z = 1/\mathcal{R}$.
- `poleMagnitudes(self)`: returns a list of the magnitudes of the poles of the system. The magnitude of a real pole is simply its absolute value. The magnitude of a complex pole is the square root of the sum of the squares of its real and imaginary parts. The `abs` function in Python does the appropriate computation for both types.
- `dominantPole(self)`: returns one of the poles with greatest magnitude. If two or more poles have the same greatest magnitude, then any of these poles may be returned.

Detailed guidance:

Step 1. Edit `sfSkeleton.py` to contain your implementation of these methods. You can test it using `swLab05Work.py`, which will load your `sfSkeleton.py`. We have set up `swLab05Work.py` to import your definitions from `sfSkeleton.py` as `sf`, so that the examples match those in **Section 4**. Note that `swLab05Work.py` contains test cases for all of the parts of this lab, so if you run the whole file, you will get errors involving functions that you have not yet written.

Hints and cautions

- To create a Polynomial, use `poly.Polynomial([...])`
- None of the operations that you implement should change any of their arguments. Be very careful of list operations that modify the input lists; e.g., `x.append`, `x.insert` and `x.reverse`.
- If you have a list bound to the variable `x`, then `x.reverse()` reverses the order of the elements of the list `x`. If you want to avoid affecting the original `x` you need to copy the list first, for example, by doing `y = x[:]`. Note that `y = x` does not copy the list, it simply creates a new name for the same list.
- You might want to use the procedure `util.argmax(l, f)`, which takes as input a list `l` and a procedure `f` that can take an element of `l` as input and return a numerical score as ouput. The result is the element of `l` for which `f` outputs the highest score.

Wk.5.1.1 Once you have debugged your code in Idle, paste it into this tutor problem, check it, and submit it.

3 Combining System Functions

Objective: Develop two basic operations, `Cascade` and `FeedbackSubtract`, for combining system functions, analogous to operations we saw for state machines.

Step 2.

> **Wk.5.1.2** Get practice with cascade combination of system functions.

Step 3. In `sfSkeleton.py`, implement the procedure `Cascade(sf1, sf2)`, which takes two instances of the `SystemFunction` class and returns a new instance of that class that represents the cascade composition of the input systems. *Although this is a procedure and not a class, we capitalize the name by analogy with the* `sm.Cascade` *class.*

> **Wk.5.1.3** Once you have debugged your code in Idle, paste it into this tutor problem, check it, and submit it.

Step 4.

> **Wk.5.1.4** Get practice with feedback-subtract combination of system functions.

Step 5. In `sfSkeleton.py`, implement the procedure `FeedbackSubtract(sf1, sf2)`, which takes two instances of the `SystemFunction` class and returns a new instance of that class that represents the feedback subtract composition of the input systems. *Although this is a procedure and not a class, we capitalize the name by analogy with the* `sm.FeedbackSubtract` *class.*

> **Wk.5.1.5** Once you have debugged your code in Idle, paste it into this tutor problem, check it, and submit it.

4 Examples

These examples, drawn from the notes, are included in `swLab05Work.py`.
Real poles:

```
>>> s1 = sf.SystemFunction(poly.Polynomial([1]),
                           poly.Polynomial([0.63, -1.6, 1]))
>>> print s1
SF(1.000/0.630 R**2 + -1.600R + 1.000)
>>> s1.poles()
[0.90000000000000069, 0.69999999999999951]
>>> s1.poleMagnitudes()
[0.90000000000000069, 0.69999999999999951]
>>> s1.dominantPole()
0.90000000000000069
```

Complex poles:

```
>>> s2 = sf.SystemFunction(poly.Polynomial([1]),
                           poly.Polynomial([1.1, -1.9, 1]))
>>> print s2
SF(1.000/1.100 R**2 + -1.900R + 1.000)
>>> s2.poles()
[(0.94999999999999996+0.44440972086577957j), (0.94999999999999996-0.44440972086577957j)]
>>> s2.poleMagnitudes()
[1.0488088481701516, 1.0488088481701516]
>>> s2.dominantPole()
(0.94999999999999996+0.44440972086577957j)
```

Driving to a wall example from the notes:

```
>>> T = 0.1
>>> k = -2.0
>>> controller = sf.SystemFunction(poly.Polynomial([k]),
                                   poly.Polynomial([1]))
>>> print controller
SF(-2.000/1.000)
>>> plant = sf.SystemFunction(poly.Polynomial([-T, 0]),
                              poly.Polynomial([-1, 1]))
>>> print plant
SF(-0.100R/-1.000R + 1.000)
>>> controllerAndPlant = sf.Cascade(controller, plant)
>>> print controllerAndPlant
SF(0.200R/-1.000R + 1.000)
>>> wall = sf.FeedbackSubtract(controllerAndPlant)
>>> print wall
SF(0.200R/-0.800R + 1.000)
>>> wall.poles()
[0.80000000000000004]
```

6.01SC Introduction to Electrical Engineering and Computer Science

Spring 2011