

# 6.01 Midterm 1 Solutions: Spring 2009

Name:

Key, Answer

Enter all answers in the boxes provided.  
No computers, cell phones, or music players.

For staff use:

1.	<b>20</b>	/20
2.	<b>20</b>	/20
3.	<b>20</b>	/20
4.	<b>20</b>	/20
5.	<b>20</b>	/20
total:	<b>100</b>	/100

## 1 Python and OOP (20 points)

Part a. What does this program print? Write your answers in the boxes provided.

```
class NN:
    def __init__(self):
        self.n = 0
    def get(self):
        self.n += 1
        return str(self.n)
    def reset(self):
        self.n = 0
class NS(NN):
    def get(self, s):
        return s + NN.get(self)
foo = NS()
print foo.get('a')
```

**a1**

```
print foo.get('b')
```

**b2**

```
foo.reset()
print foo.get('c')
```

**c1**

**Part b.** Write a Python expression that uses `optOverLine` to find the **maximum** of  $x^3 - 6$  for  $x$  in the range  $-10$  to  $10$ . Test 200  $x$  values in that range.

Here is the documentation for `optOverLine`:

```
optOverLine(objective, xmin, xmax, numXsteps, compare=<built-in function lt>)
```

Parameters:

- **objective**: a function that takes a single number as an argument and returns a value
- **compare**: a function from two values (of the type returned by **objective**) to a Boolean; should return **True** if the first argument is preferred and **False** otherwise.

Returns: a pair,  $(\text{objective}(x), x)$ , where  $x$  is one of the numeric values achieved by starting at **xmin** and taking **numXsteps** equal-sized steps up to **xmax**; the particular value of  $x$  returned is the one for which  $\text{objective}(x)$  is best, according to the **compare** operator.

```
return optOverLine(lambda x: x**3 - 6, -10, 10, 200, operator.gt)[0]
```

## 2 SystemFunction Class (20 points)

Consider two system functionals:

$$H_1 = \frac{Y}{X} = \frac{\mathcal{R}^2 + \mathcal{R}}{\mathcal{R} + 3}$$

and

$$H_2 = \frac{Y}{X} = \frac{\mathcal{R}^3 + \mathcal{R}^2}{\mathcal{R}^2 + 3\mathcal{R}}$$

**Part a.** Determine the corresponding difference equations.

Difference equation corresponding to  $H_1$ :

$$y[n] = -\frac{1}{3}y[n-1] + \frac{1}{3}x[n-1] + \frac{1}{3}x[n-2]$$

Difference equation corresponding to  $H_2$ :

$$y[n-1] = -\frac{1}{3}y[n-2] + \frac{1}{3}x[n-2] + \frac{1}{3}x[n-3]$$

Briefly describe the implications of these difference equations for similarities and differences between the two systems.

$H_2$  is equivalent to  $H_1$  with an extra pole and an extra zero, both at  $z = 0$ . Because these poles cancel, the systems will give identical responses to identical inputs; to see this, substitute  $n-1$  for  $n$  in the first equation. (Note: as an example of such a pair of systems, imagine two identical state machines being fed the same input, but one gets the input one time step later than the other. Clearly both generate the same output for a given input, but in the second one both the input and output are delayed by one time step.)

**Part b.** The numerators and denominators of  $H_1$  and  $H_2$  contain factors of  $\mathcal{R}$ :

$$H_1 = \frac{Y}{X} = \frac{\mathcal{R}^2 + \mathcal{R}}{\mathcal{R} + 3} = \frac{\mathcal{R} \times (\mathcal{R} + 1)}{\mathcal{R} + 3}$$

$$H_2 = \frac{Y}{X} = \frac{\mathcal{R}^3 + \mathcal{R}^2}{\mathcal{R}^2 + 3\mathcal{R}} = \frac{\mathcal{R} \times \mathcal{R} \times (\mathcal{R} + 1)}{\mathcal{R} \times (\mathcal{R} + 3)}$$

If we cancel one  $\mathcal{R}$  factor from the numerator and denominator of  $H_2$ , then  $H_2 = H_1$ .

Write a Python function called `ReducedSystemFunction`, which

- takes an input `inp` of type `SystemFunction`,
- cancels **all** factors of  $\mathcal{R}$  that are common to the numerator and denominator of `inp`, and
- returns the result as a new object of type `SystemFunction`.

`ReducedSystemFunction` should not modify `inp`. Some relevant software documentation follows the box. You can assume that we have already `import sf` and `poly`.

```
def ReducedSystemFunction(inp):
    num= inp.numerator.coeffs[:]
    den= inp.denominator.coeffs[:]
    while num[-1] == 0 and den[-1] == 0:
        num.pop()
        den.pop()
    return sf.SystemFunction(poly.Polynomial(num),
                             poly.Polynomial(den))
```

---

#### Attributes of SystemFunction Class:

<code>__init__(self, numPoly, denomPoly)</code>	
<code>poles(self)</code>	returns a list of the poles of the system
<code>poleMagnitudes(self)</code>	returns a list of the magnitudes of the poles of the system
<code>dominantPole(self)</code>	returns the pole with the largest magnitude
<code>__add__(self, other)</code>	
<code>numerator</code>	Polynomial in $\mathbb{R}$ representing the numerator
<code>denominator</code>	Polynomial in $\mathbb{R}$ representing the denominator

---

#### Attributes of Polynomial Class:

<code>__init__(self, coeffs)</code>	
<code>add(p1, p2)</code>	returns a new polynomial, which is sum
<code>__add__(p1, p2)</code>	
<code>__sub__(p1, p2)</code>	
<code>scalarMult(self, s)</code>	returns a new polynomial, with coefs multiplied by $s$
<code>mul(p1, p2)</code>	returns a new polynomial equal to the product
<code>__mul__(p1, p2)</code>	
<code>val(self, x)</code>	returns value of polynomial with variable assigned to $x$
<code>roots(self)</code>	return list of roots
<code>coeffs</code>	list of polynomial coefficients, highest order first
<code>order</code>	polynomial order; one less than number of coeffs

### 3 Inheritance and State Machines (20 points)

Recall that we have defined a Python class `sm.SM` to represent state machines. Here we consider a special type of state machine, whose states are always integers that start at 0 and increment by 1 on each transition. We can represent this new type of state machines as a Python subclass of `sm.SM` called `CountingStateMachine`.

We wish to use the `CountingStateMachine` class to define new subclasses that each provide a single new method `getOutput(self, state, inp)` which returns *just the output* for that state and input; the `CountingStateMachine` will take care of managing and incrementing the state, so its subclasses don't have to worry about it.

Here is an example of a subclass of `CountingStateMachine`.

```
class CountMod5(CountingStateMachine):
    def getOutput(self, state, inp):
        return state % 5
```

Instances of `CountMod5` generate output sequences of the form 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, ....

**Part a.** Define the `CountingStateMachine` class. Since `CountingStateMachine` is a subclass of `sm.SM`, you will have to provide definitions of the `startState` instance variable and `getNextValues` method, just as we have done for other state machines. You can assume that every subclass of `CountingStateMachine` will provide an appropriate `getOutput` method.

```
class CountingStateMachine(sm.SM):
    def __init__(self):
        self.startState = 0
    def getNextState(self, state, inp):
        return (state + 1, self.getOutput(state, inp))
```

**Part b.** Define a subclass of `CountingStateMachine` called `AlternateZeros`. Instances of `AlternateZeros` should be state machines for which, on even steps, the output is the same as the input, and on odd steps, the output is 0. That is, given inputs,  $i_0, i_1, i_2, i_3, \dots$ , they generate outputs,  $i_0, 0, i_2, 0, \dots$

```
class AlternateZeros(CountingStateMachine):
    def getOutput(self, state, inp):
        if not state % 2:
            return inp
        return 0
```

## 4 Signals and Systems (20 points)

Let  $H$  represent a system whose input is a signal  $X$  and whose output is a signal  $Y$ . The system  $H$  is defined by the following difference equations:

$$y[n] = x[n] + z[n]$$

$$z[n] = y[n - 1] + z[n - 1]$$

**Part a.** Which of the following systems are valid representations of  $H$ ? (Remember that there can be multiple “equivalent” representations for a system.)

	equivalent to $H$ (yes/no)?	<b>YES</b>
	equivalent to $H$ (yes/no)?	<b>NO</b>
	equivalent to $H$ (yes/no)?	<b>YES</b>
	equivalent to $H$ (yes/no)?	<b>NO</b>

**Part b.** Assume that the system starts “at rest” and that the input signal  $X$  is the unit sample signal. Determine  $y[3]$ .

$$y[3] = \boxed{4}$$

**Part c.** Let  $p_o$  represent the dominant pole of  $H$ . Determine  $p_o$ .

Enter  $p_0$  or **none** if there are no poles:  $\boxed{2}$

## 5 System Behaviors (20 points)

Consider the following system functionals:

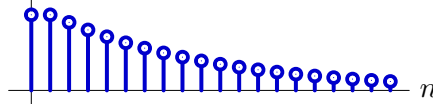
$$H_1(\mathcal{R}) = \frac{1}{1 - 3\mathcal{R} + 2\mathcal{R}^2}$$

$$H_2(\mathcal{R}) = \frac{1}{1 - \mathcal{R} + 0.1\mathcal{R}^2}$$

$$H_3(\mathcal{R}) = \frac{1}{1 - 0.1\mathcal{R}^2}$$

$$H_4(\mathcal{R}) = \frac{1}{1 - 2\mathcal{R} + 0.1\mathcal{R}^2}$$

**Part a.** Which (if any) of the system functionals have the following unit sample response?  
unit sample response



Circle all correct answers(s):  $H_1(\mathcal{R})$    $H_2(\mathcal{R})$   $H_3(\mathcal{R})$   $H_4(\mathcal{R})$  **none**

**Part b.** Which (if any) of the system functionals have the following unit sample response?  
unit sample response



Circle all correct answers(s):  $H_1(\mathcal{R})$   $H_2(\mathcal{R})$    $H_3(\mathcal{R})$   $H_4(\mathcal{R})$  **none**

**Part c.** Which (if any) of the system functionals have the following unit sample response?  
unit sample response



Circle all correct answers(s):  $H_1(\mathcal{R})$   $H_2(\mathcal{R})$   $H_3(\mathcal{R})$   $H_4(\mathcal{R})$   **none**

**Part d.** Which (if any) of the system functions correspond to stable systems?

Circle all correct answers(s):  $H_1(\mathcal{R})$    $H_2(\mathcal{R})$    $H_3(\mathcal{R})$   $H_4(\mathcal{R})$  **none**