6.01 Midterm 1 Spring 2018

Name:

Kerberos (Athena) name:

Please WAIT until we tell you to begin.

During the exam, you may refer to any written or printed paper material.
You may NOT use any electronic devices (including calculators, phones, etc).

If you have questions, please **come to us at the front** to ask them.

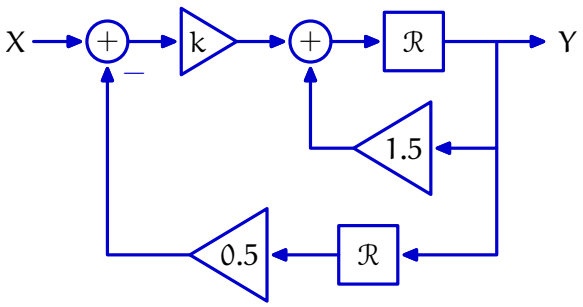
Enter all answers in the boxes provided.

Extra work may be taken into account when assigning partial credit,
but only work on pages with QR codes will be considered.

Question 1:	19 Points
Question 2:	22 Points
Question 3:	15 Points
Question 4:	19 Points
Question 5:	19 Points
Total:	94 Points

1 Stability (19 Points)

Consider the system described by the block diagram below, where k is a real number:



1.1 Specifics

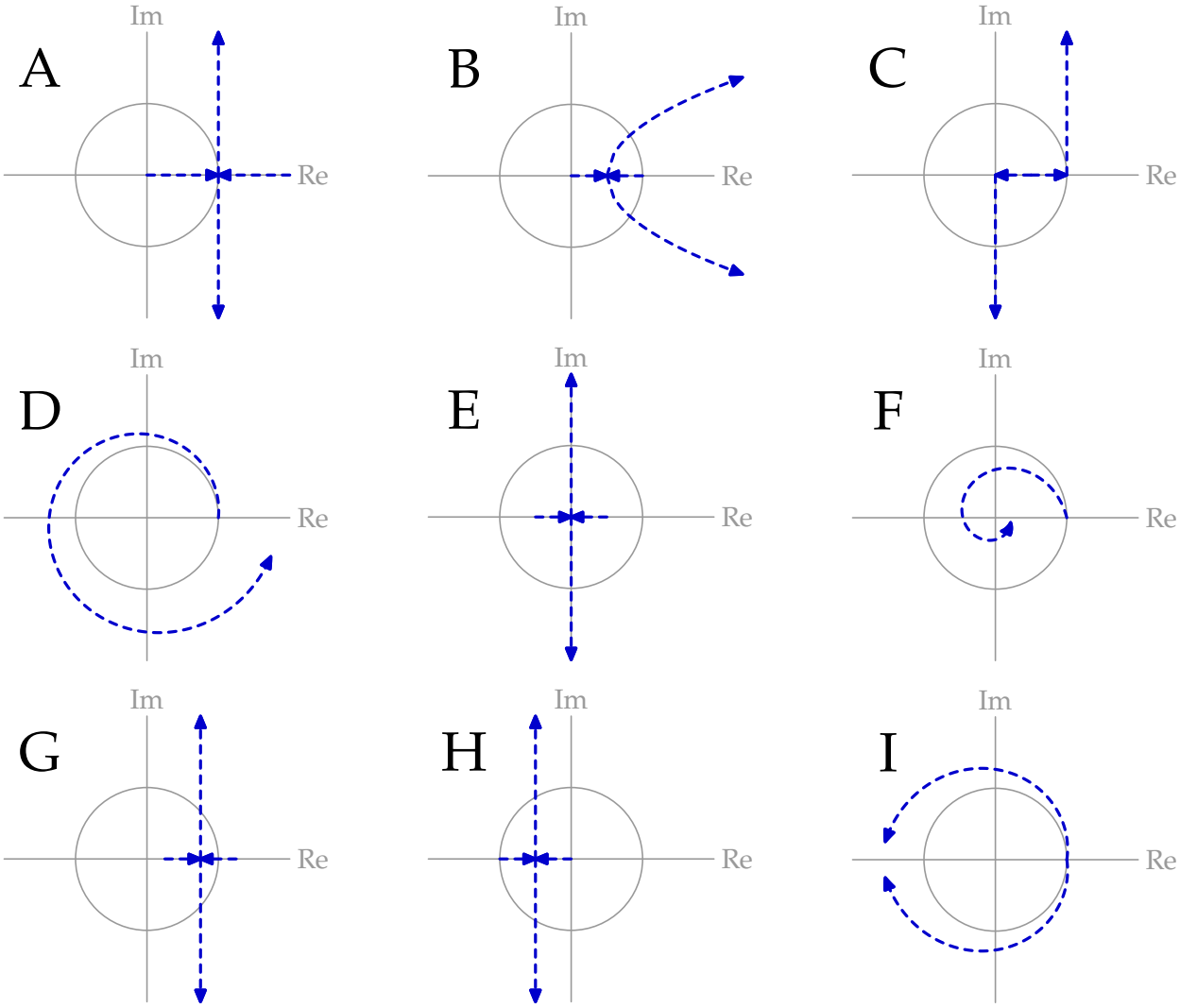
What value of k leads to the fastest convergence of the system?

What is the maximum value of k for which the system is stable?

What is the minimum value of k for which the system is stable?

1.2 Matching

Which of the following plots most closely describes how this system's poles move through the complex plane as k is varied? Assume that in all graphs, arrows are drawn in the direction of increasing k.



Plot that best describes the system (circle one letter below):

A **B** **C** **D** **E** **F** **G** **H** **I**

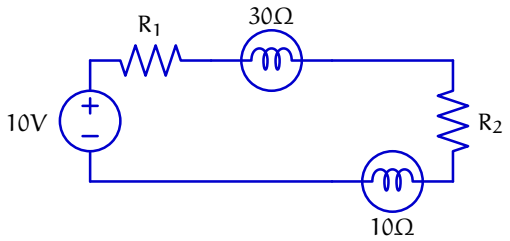
2 Let There Be Light (22 Points)

2.1 Part 1

In each of the following circuits, is it possible to adjust the positive, finite, non-zero resistances R_1 and R_2 such that the light bulbs are equally bright? If so, enter one example of a valid combination of R_1 and R_2 in the boxes. If not, enter `None` in each box.

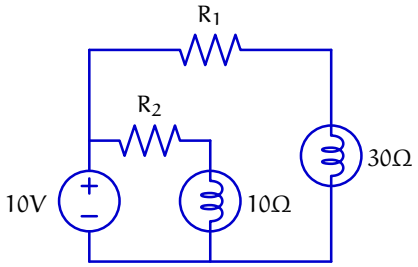
We can model each light bulb as a resistor. In each circuit, there are two different bulbs (one with a 10Ω resistance, and one with a 30Ω resistance). Because of the specific properties of these bulbs, we will assume that they have the same brightness when the voltage drop across them is the same.

2.1.1 Circuit 1



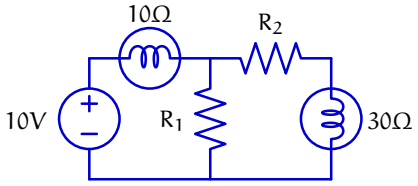
$R_1 =$ $R_2 =$

2.1.2 Circuit 2



$R_1 =$ $R_2 =$

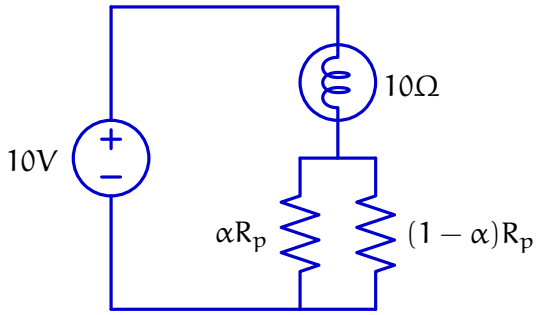
2.1.3 Circuit 3



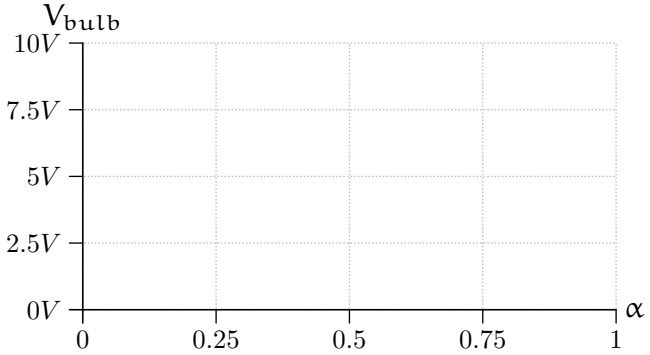
$R_1 =$ $R_2 =$

2.2 Part 2

Ben Bitdiddle was trying to build a circuit to allow him to adjust the brightness of a light bulb by turning the shaft of a potentiometer. His goal was to create a circuit where brightness (proportional to the voltage drop across the bulb) increases linearly with α. Here is the circuit Ben constructed, using a potentiometer where $R_p = 120\Omega$:

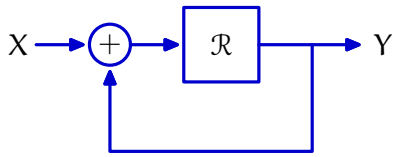


Did Ben accomplish his goal? On the axes below, sketch the relationship between α and the voltage drop across the light bulb:



3 The Art of the State (15 Points)

Consider the following system, where the delay element is initialized so that its output on the first timestep is a value `a` (note that this system is *not* being started from rest).

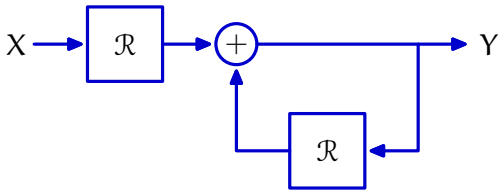


This system can be represented by the following Python code, using our simulator infrastructure:

```
system1 = FeedbackAdd(R(a), Gain(1))
```

3.1 System 2

Consider also the following system, where the delays have been initialized so that their outputs on the first timestep are `b` and `c`, respectively:



This system can be represented by the following Python code using our simulator infrastructure:

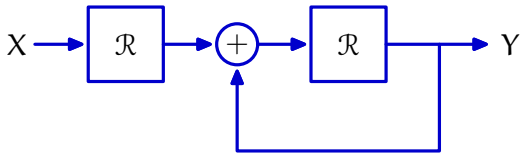
```
system2 = Cascade(R(b), FeedbackAdd(Gain(1), R(c)))
```

Is it possible to adjust `b` and `c` such that `system2` and `system1` produce identical output sequences for all possible input sequences? If so, enter values of `b` and `c` that accomplish this goal in the box below (in terms of `a`). If not, enter `None` in each box.

`b =` ☐ `c =` ☐

3.2 System 3

Consider also the following system, where the delays have been initialized so that their outputs on the first timestep are `d` and `e`, respectively:



This system can be represented by the following Python code, using our simulator infrastructure:

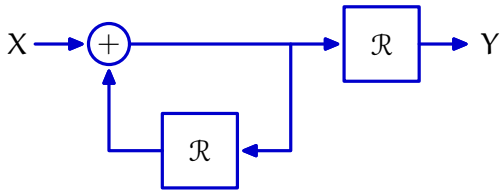
```
system3 = Cascade(R(d), FeedbackAdd(R(e), Gain(1)))
```

Is it possible to adjust `d` and `e` such that `system3` and `system1` produce identical output sequences for all possible input sequences? If so, enter values of `d` and `e` that accomplish this goal in the box below (in terms of `a`). If not, enter `None` in each box.

`d =` ☐ `e =` ☐

3.3 System 4

Consider also the following system, where the delays have been initialized so that their outputs on the first timestep are `f` and `g`, respectively:



This system can be represented by the following Python code, using our simulator infrastructure:

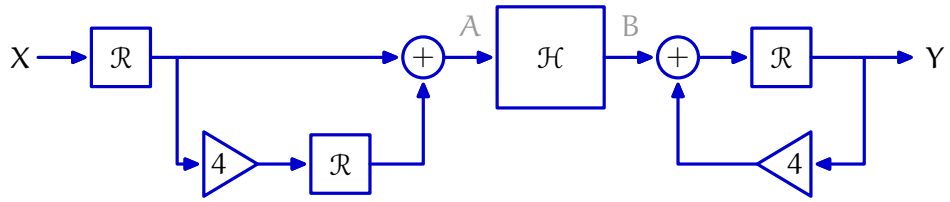
```
system4 = Cascade(FeedbackAdd(Gain(1), R(f)), R(g))
```

Is it possible to adjust `f` and `g` such that `system4` and `system1` produce identical output sequences for all possible input sequences? If so, enter values of `f` and `g` that accomplish this goal in the box below (in terms of `a`). If not, enter `None` in each box.

`f =` ☐ `g =` ☐

4 Stuck in the Middle With You (19 Points)

Consider the system represented by the block diagram below, where $\mathcal{H} = \frac{B}{A}$ is an unknown subsystem:



4.1 Response

Compute the first 4 samples A when the input to the system, X, is a unit sample signal, and the system starts at rest.

$a[0] =$ $a[1] =$ $a[2] =$ $a[3] =$

4.2 Response 2

Compute the first 4 samples of Y when the signal B is the unit sample signal and the system starts at rest.

$y[0] =$ $y[1] =$ $y[2] =$ $y[3] =$

4.3 Mystery

Determine the system functional $\mathcal{H}$ in the system from above, such that the unit sample response of the entire system ($\frac{Y}{X}$) is given by the following:

$$h[n] = \begin{cases} 4 & \text{if } n = 3 \\ 0 & \text{otherwise} \end{cases}$$

Enter your result in the box below:

$\mathcal{H} =$

5 Root Locus Plotter (19 Points)

In this problem, we will create a function that generates root locus plots for arbitrary systems specified using our `System` framework.

On the facing page, complete the definition of `plot_root_locus`, which should construct a root-locus diagram for an arbitrary system. Your function should take as arguments:

- `system`, a function that takes as input a value of a tunable parameter `k`, and returns an instance of `System` representing the system of interest for that value of `k` (documentation for the `System` class has been included on the last page of this exam)
- `k_min`, the minimum value of `k` to include in the plot
- `k_max`, the maximum value of `k` to include
- `k_step`, how much `k` should change by on each step

You may assume `k_max` > `k_min`, and that `k_step` is positive.

Your function should compute the poles of the system from several `k` values, starting at `k_min` and increasing by `k_step` until `k_max` is reached.

For example, with `k_min = 1`, `k_max = 2`, and `k_step = 0.25`, your function should use the values `1`, `1.25`, `1.5`, `1.75`, and `2` for `k`.

With `k_min = 5`, `k_max = 6`, and `k_step = 0.4`, your function should use the values `5`, `5.4`, and `5.8` for `k`.

For each `k` value, we would like to mark points on a complex plane, according to the rules below:

- If a particular `k` causes the system to be stable, all of the poles associated with that `k` value should be plotted in green.
- Otherwise, all of the poles associated with that `k` value should be plotted in red.

Assume that you are given a function `plot_point(color, a0, b0)` that will plot $a_0 + b_0 j$ in the complex plane. For example, to plot the point `j` in blue, you could use:
`plot_point("blue", 0, 1)`

Your function should use `plot_point` to generate a root locus plot, but it does not need to return anything. Your function should work for arbitrary systems, regardless of the number of poles that system has.

Note that you can determine the real and imaginary parts of a number `x` with `x.real` and `x.imag`, respectively; and you can determine the magnitude of a number `x` with `abs(x)`, even if `x` is complex.

```
def plot_root_locus(system, k_min, k_max, k_step):
    # your code here
```

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Worksheet (intentionally blank)

Documentation for `System` Class

`System(numerator, denominator=None, previous_inputs=None, previous_outputs=None)`

Represents Linear Time Invariant systems, and supports step-by-step simulation.

Parameters:

- `numerator`: a list of numbers representing the numerator of the system functional, starting with lowest order coefficient
- `denominator`: a list of numbers representing the denominator of the system functional, starting with lowest order coefficient
- `previous_inputs`: a list of previous input numbers to store in the initial state (for use in simulation)
- `previous_outputs`: a list of previous output numbers to store in the initial state (for use in simulation)

Instance Variables:

- `numerator`: list of numbers representing the numerator of the system functional
- `denominator`: list of numbers representing the denominator of the system functional
- `initial_state`: tuple of two lists representing the previous inputs and outputs to the system

Methods:

- `calculate_step(state, inp)`
  Parameters:
  - `state`: object representing a current hypothetical state of the system
  - `inp`: input value to the system

  Returns: a tuple containing the system's new state and output after a hypothetical step using the given state and input.

  Note that this is a *pure function* which does not store or modify any class, instance, or global variables.

- `dominant_pole()`
  Returns: the dominant pole of this system.

- `poles()`
  Returns: a list containing the poles of this system.

Worksheet (intentionally blank)