

6.01 Final Exam: Spring 08

Name: **Answers**

This exam consists of 7 problems (pages 2-16 of this handout).

Enter your answers in the boxes provided.

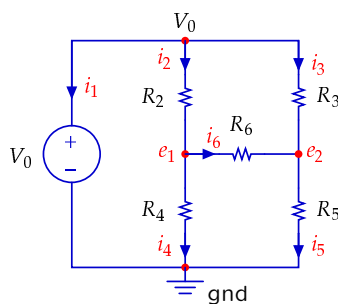
No computers, cell phones, or music players.

For staff use:

1.	/10
2.	/9
3.	/6
4.	/20
5.	/15
6.	/20
7.	/20
total:	/100

1 Short-Answer Questions (10 points)

Part a. Consider the following circuit.



Determine if the following equations and/or statements are

– **Always True** – i.e., true for all possible values of the resistors $R_2 - R_6$ and voltage V_0

or

– **NOT Always True** – i.e., false for some or all resistor and voltage values.

Check the appropriate box for each of the following:

NOT
Always
True

Always
True

If $\frac{R_2}{R_4} = \frac{R_3}{R_5}$ then $i_6 = 0$

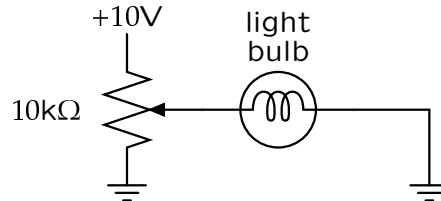
$i_2 + i_3 = i_4 + i_5$

$i_2 + i_6 = i_3$

$e_1 = \frac{R_4}{R_2 + R_4} V_0$

If $i_6 = 0$ then $\frac{R_2}{R_2 + R_4} = \frac{R_3}{R_3 + R_5}$

Part b. Our goal is to design a dimmer for a light bulb that can be modelled as a constant resistor of 10Ω . We have a single 10V power supply. Our first design uses a $10k\Omega$ potentiometer, with the goal of controlling the current through the lamp so that the current is **proportional** to the potentiometer setting, with a maximum current of 1A.



Briefly (using fewer than 50 words) describe problems with this circuit.

If the potentiometer is turned nearly all the way to the +10V side, then the current through the light bulb will be near 1A. Otherwise, the resistance of the potentiometer will limit the current through the light bulb to $< 1\text{mA}$. This the potentiometer will work more like an on/off switch than like a dimmer.

Suggest a better circuit. Draw it in the following box.

potentiometer -> unity buffer -> bulb -> ground

Explain briefly why your circuit is better.

The current into the op-amp buffer will be nearly zero. Therefore, the potentiometer will act as a voltage divider, and its output voltage will be proportional to the angle of its shaft. The op-amp buffer will drive its output to match the potentiometer voltage. Since we are modelling the light bulb as a resistor, its current will be proportional to the potentiometer shaft angle.

2 More Short-Answer Questions (9 points)

Part a. Describe a situation during robot localization where the $P(O|S)$ window looks very different from the Belief window. Explain very briefly.

Any situation where the local reading is ambiguous (could arise in many states) but which is disambiguated by the previous history of readings. For example, a flat wall right after seeing a unique feature in a room.

Part b. Describe a situation in the world in which these two behaviors will generate different robot commands, or explain why they are the same.

```
robot.behavior = If(isBlocked, RotateTSM(math.pi), ForwardTSM(1.0))
```

```
robot.behavior = Switch(isBlocked, RotateTSM(math.pi), ForwardTSM(1.0))
```

If there is a block 0.5 meters from where the robot starts, the first one (using If) will collide with the block, since it only tests the condition at the beginning of the motion. The second one (Switch) will check the condition every time so it will stop near the block and turn around.

Describe a situation in the world in which these two behaviors will generate different robot commands, or explain why they are the same.

```
robot.behavior = If(isBlocked, RotateTSM(math.pi), ForwardTSM(1.0))
```

```
if isBlocked(SensorInput()):
    robot.behavior = RotateTSM(math.pi)
else:
    robot.behavior = ForwardTSM(1.0)
```

They behave the same, assuming that nothing changes in the world between the time that the second one is evaluated and the robot behavior starts execution and that `SensorInput()` returns the actual sensor values.
Another acceptable answer is that `SensorInput` is not defined, except inside `step`.

Part c. You are told that we have a class **A**, with one initialization argument, with a method **f** that takes one argument and returns a number.

```
>>> x = A(2)
>>> x.f(3)
9
```

Define a class **B** that is a subclass of class **A**. Class **B** has the same initialization as **A** and a method **f** that returns twice whatever value the **f** method of **A** would return. You don't need to know what the code for **A** is.

```
>>> x = B(2)
>>> x.f(3)
18
```

```
class B(A):
    def f(self, x):
        return A.f(self, x)*2
```

3 In the Tank (6 points)

Imagine a cylindrical water tank with a source of water entering the bottom of the tank and a sensor to measure the depth of the water. The source of water can be adjusted so that water enters (or exits) the tank at a desired rate. In addition to this controlled source of water, there is also a leak in the tank, through which water flows regardless of the controlled flow. The diameter of the tank is such that one inch in depth is one gallon of water. The following variables characterize this problem:

- $d[t]$ is the depth, in inches, of the water in the tank at time t .
- $v[t]$ is the volume of water that flows through the controlled source of water in the time interval $[t, t + 1]$.
- $o[t]$ is the volume of water that leaks from the tank in the time interval $[t, t + 1]$. Assume $o[t]$ is proportional to $d[t]$, with a proportionality constant of 0.1
- $g[t]$ is the desired (goal) depth of water (in inches) in the tank at time t
- $s[t]$ is the sensed depth (in inches) of water in the tank at time t ; assume the sensor introduces one unit of delay, so that $s[t] = d[t - 1]$

a. Write a difference equation that describes the depth of water in the tank ($d[\cdot]$) in terms of the flow through the controlled source ($v[\cdot]$).

$$d[t + 1] = d[t] + v[t] - o[t]$$

$$o[t] = 0.1d[t]$$

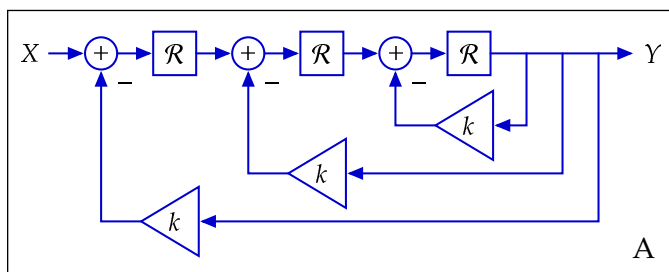
$$d[t + 1] - 0.9d[t] = v[t]$$

b. Write a difference equation describing a proportional controller that specifies $v[\cdot]$ as a function of sensed depth $s[\cdot]$ and desired depth $g[\cdot]$. Let α represent the constant of proportionality.

$$v[t] = \alpha(g[t] - s[t])$$

4 Linear Systems (20 points)

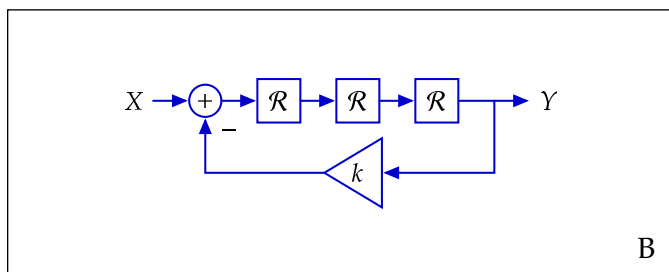
Part a. Several systems are illustrated below. The left column shows block diagrams. The right column shows system functions and pole locations for $k = 0.9$. Indicate which panel on the right (if any) corresponds to each panel on the left by drawing a straight line from A, B, C, and/or D to its partner. [Your answer should include 4 or fewer straight lines.]



$$\frac{Y}{X} = \frac{\mathcal{R}^3}{k\mathcal{R}^3 + k\mathcal{R}^2 + k\mathcal{R} + 1}$$

poles: 1.72; $-0.41 \pm 0.59j$

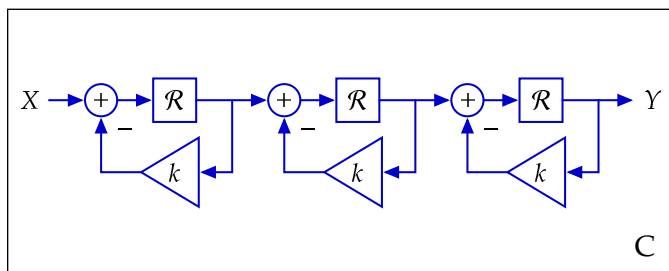
E



$$\frac{Y}{X} = \frac{1}{k\mathcal{R}^3 + 1}$$

poles: $-0.48 \pm 0.84j$; 0.96

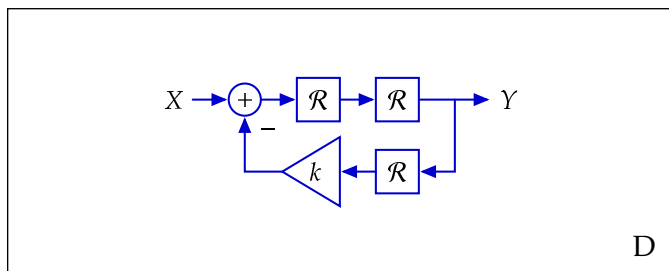
F



$$\frac{Y}{X} = \frac{\mathcal{R}^3}{k\mathcal{R}^3 + 1}$$

poles: $-0.48 \pm 0.84j$; 0.96

G



$$\frac{Y}{X} = \frac{\mathcal{R}^3}{k^3\mathcal{R}^3 + 3k^2\mathcal{R}^2 + 3k\mathcal{R} + 1}$$

poles: 0.9; 0.9; 0.9

H

A → E

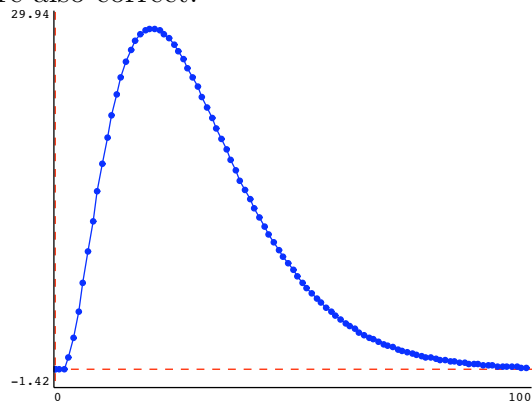
B → G

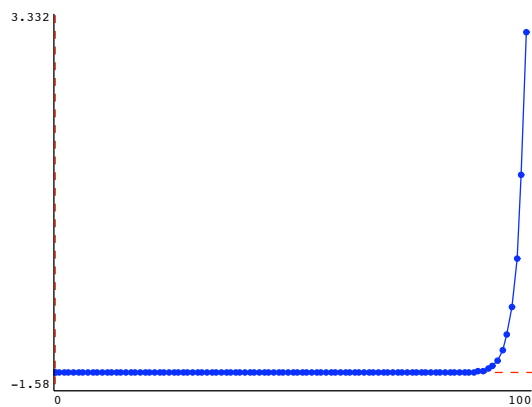
C → H

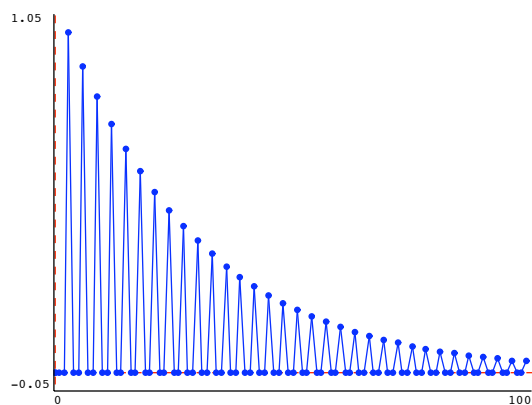
D → nowhere

Part b. Unit sample responses. Indicate which (if any) of the systems on the preceding page could have generated the following unit sample responses (with the system initially at rest) by entering A, B, C, D, E, F, G, H or “none” in the corresponding boxes.

Put exactly one answer in each box. You will receive full credit if your answer is correct regardless of whether other answers are also correct.







top: C or H

middle: A or E

bottom: D B,F,G

5 Friend or Foe (15 points)

Imagine that you are defending a city and there is an aircraft flying toward you. It may be important to know whether that aircraft is a 'friend' or a 'foe' (enemy). Assume you have a radar sensor that can give you noisy information about the type of the aircraft that is approaching.

We will model this situation as an HMM, in which:

- The state space is described by two components \mathbf{d} and \mathbf{a} , where \mathbf{d} is the number of miles (0, 1, ..., 10) away the target is and \mathbf{a} is its *attitude* to you, which is either 'friend' or 'foe'. The values of \mathbf{d} correspond to ranges of distance, so that, in fact, $\mathbf{d} == 0$ means the aircraft is somewhere between 0 and 1 miles away, etc.
- The observation space is {'oFriend', 'oFoe'}, which stands for *observed friend* and *observed foe*.
- There are no actions (or, if you prefer, a single action, which just waits a time step).
- The transition model is specified in the following Python method, which takes a state \mathbf{s} as input and returns a `DDist` over possible next states:

```
def transitionModel(s, i):
    # note that the i (the input action) is ignored
    (d, a) = s
    if d == 0:
        return DDist({(0, a): 1.0})
    else:
        return DDist({(d, a): 0.5, (d-1, a): 0.5})
```

- The observation model is as described in the following Python method, which takes a state \mathbf{s} as input and returns a `DDist` over possible observations:

```
def observationModel(s):
    (d, a) = s
    if a == 'friend':
        return DDist({'oFriend': 1 - d / 20.0, 'oFoe' : d / 20.0})
    else:
        return DDist({'oFoe': 1 - d / 20.0, 'oFriend' : d / 20.0})
```

Questions:

1. Assume that the aircraft are approaching with a constant velocity. What velocity, in miles per time step, would generate the transition model over distance intervals given in the transition model?

Answer:

0.5 miles/step

2. Provide an alternative transition model in which friendly aircraft move 1.5 miles per time step (and the foes move at the same rate as the given model.)

```
def transitionModel(s, i):
    (d, a) = s
    if d == 0 or (d == 1 and a == 'friend'):
        return DDist({(0, a): 1.0})
    else:
        if a == 'foe':
            return DDist({(d, a): 0.5, (d-1, a): 0.5})
        else:
            return DDist({(d-1, a): 0.5, (d-2, a): 0.5})
```

3. At what distance is our sensor most useful?

Answer:

0 miles

How does it behave at that distance?

Answer:

perfectly

4. Using the original transition and observation models, if the initial belief state, b_0 , is $\text{DDist}(\{(10, \text{'friend'}) : 0.5, (10, \text{'foe'}) : 0.5\})$ then what would b'_0 be, after receiving observation $o_0 = \text{'oFriend'}$.

Answer:

DDist((10, 'friend') : 0.5, (10, 'foe'): 0.5)

5. After that, what would the belief state b_1 be?

Answer:

DDist((9, 'friend') : 0.25, (9, 'foe'): 0.25, (10, 'friend') : 0.25, (10, 'foe'): 0.25)

6. How many non-zero entries are there in b_4 (the answer will be the same for any sequence of observations)?

Answer:

10

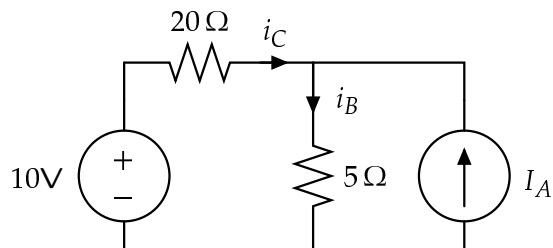
7. Starting again from the initial belief state, and imagining the following observation sequence: ['oFriend', 'oFoe', 'oFriend', 'oFoe'], is it more likely that the aircraft is a friend or a foe?

Answer:

foe

6 Circuits (20 points)

Consider the following circuit.



Part a. Find i_C if $I_A = 0$.

$$i_C = \boxed{0.4\text{A}}$$

$$i_C = \frac{10\text{V}}{20\Omega + 5\Omega} = \frac{2}{5}\text{A} = 0.4\text{A}$$

Part b. Find i_B if $I_A = 5\text{A}$.

$$i_B = \boxed{4.4\text{A}}$$

$$i_B = \frac{10\text{V}}{20\Omega + 5\Omega} + \frac{20\Omega}{5\Omega + 20\Omega} \times 5\text{A} = 4.4\text{A}$$

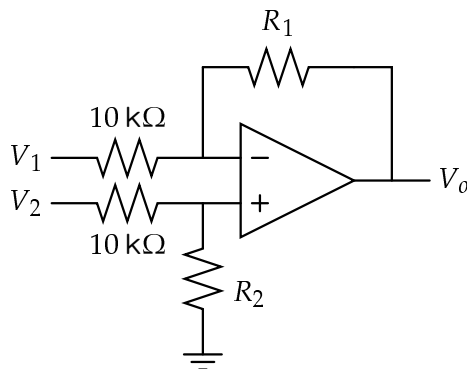
Part c. Find I_A so that $i_C = 0$.

$$I_A = \boxed{2\text{A}}$$

$$i_C = 0 = \frac{10\text{V}}{20\Omega + 5\Omega} - \frac{5\Omega}{5\Omega + 20\Omega} \times I_A = \frac{2}{5}\text{A} - \frac{1}{5}I_A$$

$$I_A = 2\text{A}$$

Part e. Consider the following op-amp circuit.



Fill in the values of R_1 and R_2 required to satisfy the equations in the left column of the following table. The values must be non-negative (i.e., in the range $[0, \infty]$). If the equation is impossible to implement with non-negative resistors, then write “impossible” for both resistor values.

	R_1	R_2
$V_o = 2V_2 - 2V_1$	$20\text{k}\Omega$	$20\text{k}\Omega$
$V_o = 2V_2 - V_1$	$10\text{k}\Omega$	∞
$V_o = V_2 - 2V_1$	$20\text{k}\Omega$	$5\text{k}\Omega$
$V_o = 4V_2 - 2V_1$	$20\text{k}\Omega$ or impossible	impossible

$$V_+ = \frac{R_2}{10\text{k}\Omega + R_2} V_2 = V_- = \frac{R_1}{10\text{k}\Omega + R_1} V_1 + \frac{10\text{k}\Omega}{10\text{k}\Omega + R_1} V_o$$

$$V_o = \frac{10\text{k}\Omega + R_1}{10\text{k}\Omega + R_2} \times \frac{R_2}{10\text{k}\Omega} \times V_2 - \frac{R_1}{10\text{k}\Omega} \times V_1$$

Solving the first and third rows requires just simple algebra.

Solving the algebra for the fourth row suggests that R_2 is negative. Therefore, this condition cannot be realized with non-negative resistors.

Solving the second row is a bit more tricky, since the algebraic solution suggests

$$2R_2 = 2R_2 + 20\text{k}\Omega$$

which can only be satisfied in the limit as $R_1 \rightarrow \infty$.

7 A Puzzle (20 points)

Consider the standard *Eight Puzzle*. It has 8 tiles arranged on a 3 by 3 grid. Here is one possible arrangement of the tiles:

2	8	3
1	6	4
7		5

The tiles neighboring an empty space can be slid into the space. We can think of the operations on the puzzle as *moving the space up, down, left, or right*. It obviously cannot be moved beyond the bounds of the puzzle. In the example above, if we were to slide the space **up**, then the **6** tile would be in the bottom row and the space would be in the middle.

We can solve this problem using search. A state of the search would be an array of numbers (and None for the space) showing the location of each tile on grid. A goal state would be some specified arrangement of the tiles.

Assuming we apply pruning rule 1, but not the others, how many descendants are there for the state shown above:

- at level 1 (immediate children)?

Answer:

- at level 2 (children of level 1)?

Answer:

7.1 Search

We would like to identify the advantages and disadvantages of each of the following search methods **for this problem**. We assume, as always, that we use pruning rules 1 and 2.

Enter T or F in the boxes provided if the following statement is true or false, respectively.

- **depth-first, no dynamic programming:**

- T This method is guaranteed to find a path.
- F The path that is found is guaranteed to be short.
- T The same board state may be visited multiple times.
- T The agenda is likely to remain short (relative to the other methods).

- **depth-first, with dynamic programming:**

- T This method is guaranteed to find a path.
- F The path that is found is guaranteed to be short.
- F The same board state may be visited multiple times.
- T The agenda is likely to remain short (relative to the other methods).

- **breadth-first, no dynamic programming:**

- T This method is guaranteed to find a path.
- T The path that is found is guaranteed to be short.
- T The same board state may be visited multiple times.
- F The agenda is likely to remain short (relative to the other methods).

- **breadth-first, with dynamic programming:**

- T This method is guaranteed to find a path.
- T The path that is found is guaranteed to be short.
- F The same board state may be visited multiple times.
- F The agenda is likely to remain short (relative to the other methods).

7.2 State machine

We can describe the puzzle and its evolution using a state machine, in much the same way we described the wolf-goat-cabbage problem. We will specify the state with two components:

- A pair of *row, column* indices, indicating where the space is; and
- A list of three lists of items; each item is a digit between 1 and 8, or `None`. This describes the state of the board.

So, for example, we could represent the puzzle state above as:

```
((2, 1), [[2, 8, 3], [1, 6, 4], [7, None, 5]])
```

Technically speaking, we don't need the first component of the state (it could always be computed from the list of three lists), but it will simplify our coding if we maintain both representations.

A skeleton of the state machine defining the eight puzzle is shown on the next page. Fill in the definition of `getNextValues`. If a move would produce an illegal state, then the new state should be the same as the current state. The output of the machine should just be the next state.

```
class EightPuzzle(SM):
    startState = ((0, 2), [[0, 1, None], [2, 3, 4], [5, 6, 7]])
    size = 3
    offsets = {'up': (-1, 0), 'down': (1, 0), 'left': (0, -1), 'right': (0, 1)}
    legalInputs = ['up', 'down', 'left', 'right']

    def getNextValues(self, state, inp):
        ((p1, p2), board) = state
        bcopy = copy.deepcopy(board)

        (off1, off2) = self.offsets[inp]
        if 0 <= p1+off1 < self.size and 0 <= p2+off2 < self.size:
            c = board[p1][p2]
            np1 = clip(p1+off1, 0, self.size-1)
            np2 = clip(p2+off2, 0, self.size-1)
            nc = board[np1][np2]
            bcopy[p1][p2] = nc
            bcopy[np1][np2] = c
            newState = ((np1, np2), bcopy)
            return (newState, newState)
        else:
            return (state, state)
```