

Midterm 1 Practice

This is a collection of problems from past exams. Note that the first two were from the Spring 08 midterm, which was a three-hour take-home. This is much, much longer than our actual exam will be.

1 Robot behavior

In the following, we would like our robot behavior to transduce a stream of sensor inputs (sonar readings and poses, as before), to a stream of `io.Action` instances, specifying the forward and angular velocities for the robot.

So, here is a brain that would use a behavior that you write in the following problem:

```
def setup():
    robot.behavior = yourSMHere
    robot.behavior.start()

def step():
    robot.behavior.step(io.SensorInput()).execute()
```

- (10 points)** Define a class `TurnSmoothly`, which is a subclass of `SM`. It should use a proportional controller to rotate until its odometry heading is `thetaDesired`.
- (10 points)** Build a linear difference equation model of the robot's controller and of the world. Assume that there is one unit of delay in the dependence between the robot's angular velocity and its pose angle, and that there is one unit of delay in setting the desired angular velocity. You may do this algebraically by hand, or by using the `SystemFunction` software, but either way, your answer should be a difference equation, and you should show how you found it.
- (10 points)** Find a gain or gains for your controller that cause the system in your model above to converge monotonically to a heading of `thetaDesired`. Assume that T (the length of a delay) is 0.2. Explain how you found the gain. (You may do this algebraically by hand, or by using the `SystemFunction` software.)
- (5 points)** What is the effect on the stability of your controller of increasing the value of T . Explain your answer. Does it match what you might expect intuitively?
- (7 points)** *This problem may take more time than the others; leave it until the end.*

Now, imagine that instead of giving angular velocity commands to the robot, we can only give angular *acceleration* commands. We'll consider controllers for this problem that are analogous to the "proportionalPlusDelay" controller (with gains K_1 and K_2).

Define a function that creates a `SystemFunction` model of the robot's controller and of the world. Assume that there is one unit of delay in the dependence between the robot's

angular velocity and its pose angle, and a one unit of delay in the dependence between the robot's angular acceleration and its angular velocity.

Use Python to search the range of values of each K between -10 and 10. If $K_2 = 0$, are there values of K_1 for which the controller is stable? If $K_1 = 0$, are there values of K_2 for which the controller is stable? Are there combined values of K_1 and K_2 that make the controller stable. In each case, indicate the best gains that you found.

2 Weighted objective

2.1 Preliminary material

The following material was in a tutor problem from last term: A priority queue is a kind of data structure that you can add elements to in any order, but which gives you an operation that allows you to take out the smallest one. We'll assume that the elements being added are numbers, and the ordering is less than.

Create a class, called `PriorityQueue` that implements the priority queue abstract data type. It should have three methods: `__init__`, which initializes the priority queue, `insert`, which takes a number and adds it to the priority queue, and `extract`, which takes the smallest element out of the priority queue and returns it.

There are all kinds of fancy algorithms for doing these operations efficiently; you need not use them here. For example, you might just store the elements in a list. You can use the Python list operations `append`, `remove`, and `min`. Assuming that `items` is a list,

- `items.append(x)` adds element `x` to the end of the list, and returns `None`
- `items.remove(x)` removes the first occurrence of element `x` in the list, and returns `None`; it generates an error if `x` isn't in the list.
- `min(items)` returns the smallest element in the list

There can be multiple copies of a single value in the PQ. In such cases, you should remove each one separately. So, for instance, here's a test interaction your object should support:

```
> pq = PriorityQueue()
> pq.insert(10)
> pq.insert(8)
> pq.insert(10)
> pq.insert(20)
> pq.extract()
8
> pq.extract()
```

```

10
> pq.extract()
10
>pq.insert(30)
>pq.extract()
20

```

2.2 Main questions

In this problem, we are going to make a data structure similar to a priority queue (which we saw in tutor problem described above), but with a special method for selecting the element to be removed. We'll call it a WOPQ (weighted-objective priority queue). We can think of it as an abstract data type with the following operations:

- Make a new WOPQ, and store **features**, which is a list of functions, each of which takes an element as input and returns a number, and **weights**, which is a list of positive numbers that sum to 1, of the same length as **features**
- Insert an element to the WOPQ
- Extract the *best* element from the WOPQ and return it after deleting it from WOPQ. An element x is the best element if its priority

$$p(x) = \sum_i f_i(x) \cdot w_i \text{ ,}$$

where f_i is the i th function in **features** and w_i is the i th weight in **weights**, is greater than or equal to the priorities of all other elements in the WOPQ

A WOPQ shouldn't assume anything about the elements it will hold.

Here is a concrete example of a WOPQ:

- The elements will be lists of numbers
- The features are **sum** and **len**
- The weights are 0.6 and 0.4

In this case, if $\mathbf{a} = [1, 2, 3]$, then $\text{sum}(\mathbf{a}) = 6$ and $\text{len}(\mathbf{a}) = 3$. So its priority $p(\mathbf{a}) = 0.6 * 6 + 0.4 * 3 = 4.8$. For another list, $\mathbf{b} = [1, 1, 1, 1, 1]$, then $\text{sum}(\mathbf{b}) = 5$ and $\text{len}(\mathbf{b}) = 5$. So its priority $p(\mathbf{b}) = 0.6 * 5 + 0.4 * 5 = 5$. If our WOPQ contained both \mathbf{a} and \mathbf{b} , and we did an extract operation, then element \mathbf{b} would be deleted and returned.

But in other instances of WOPQs, the elements might be system functions and the features related to the poles, or the elements might be records describing bank accounts and the features related to balances and interest rates.

Here is the skeleton of a WOPQ class:

```
class WOPQ:
    def __init__(self, features, weights):
        self.features = features
        self.weights = weights
        self.elements = []

    def insert(self, x):
        self.elements.append(x)

    def extract(self):
        # your code here
```

1. (10 points) Write a Python procedure `argmax` that takes two arguments: the first is a list of elements, and the second is a function from elements to numerical values. It should return the element of the list with the highest numerical score.
2. (5 points) Let's make a WOPQ with fish. Here's the `Fish` class, and some fishy instances:

```
class Fish:
    def __init__(self, length, width):
        self.fishLength = length
        self.fishWidth = width

    def length(self):
        return self.fishLength
    def width(self):
        return self.fishWidth
```

```
Lee = Fish(12, 200)
Evelyn = Fish(30, 40)
Sydney = Fish(60, 23)
```

Provide a Python statement that would construct a WOPQ that measures fish so that the priority of a fish is 0.9 times its length plus 0.1 times its width.

3. (10 points) Write the `extract` method of the WOPQ class, which finds the element with the highest score, removes it from the list of elements, and returns it. Use `argmax`. Remember that if `x` is a list, then `x.remove(5)` will remove the first occurrence of 5 from `x` (but it doesn't return a value).
4. (5 points) Letting `wopq` be the WOPQ you constructed in the previous question, imagine that the following operations have been conducted on it:

```
wopq.insert(Lee)
wopq.insert(Evelyn)
wopq.insert(Sydney)
```

Now, what would be the result of executing

```
wopq.extract()
```

Explain which element was selected for extraction and why.

5. (10 points) After all that work, you discover that someone has already written a FPQ class, whose initialization method takes a function `priorityFun` as an argument, and whenever the `extract` method is called, it extracts the item with the highest value of that function.

That is, we have:

```
class FPQ:
    def __init__(self, priorityFun):
        self.priorityFun = priorityFun
        <etc>
```

Provide an implementation of the WOPQ class that uses inheritance to take advantage of the existing FPQ class.

6. (10 points) *This problem may take more time than the others; leave it until the end.*

Now, suppose you are given a list of candidate features, which are, as above, procedures that map an element into a number. We'd like to select a subset of these features, to use later in our WOPQ. Write a procedure `selectFeatures` that takes as input

- a list of candidate features
- a number `n` (which less than the number of features in the list of candidate features)
- a list of elements in our domain (to which the features may be applied)

Define the *range* of a feature f on the list of elements E to be $\max_{x \in E} f(x) - \min_{x \in E} f(x)$. (That is, the maximum value of the feature on all of the examples minus the minimum value of the feature on all of the examples). Your procedure should return a list of `n` of the given features for which the range on the given set of elements is the largest.

Note that, to sort a list of items according to some function of their values, you can use the `sorted` function. The example below sorts a list of numbers according to the squares of their values:

```
>>> foo = [1, 2, -30, -20, 3, 90, -27]
>>> sorted(foo, key = lambda x: x * x)
[1, 2, 3, -20, -27, -30, 90]
```

You can put any function of a single argument in as the **key** parameter to `sorted`. It is most straightforward to use and understand if the values output by the **key** function are numbers.

3 More OOP

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
```

4 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:

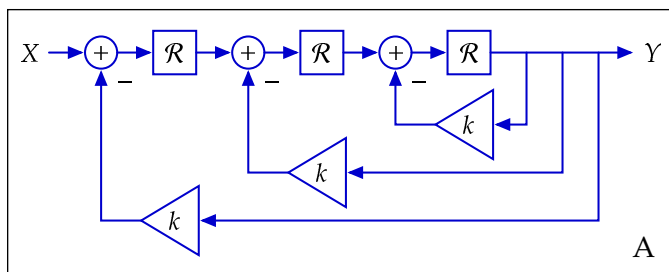
- $d[t]$ is the depth, in inches, of the water in the tank at time t .
- $v[t]$ is the volume of water that flows through the controlled source of water in the time interval $[t, t + 1]$.
- $o[t]$ is the volume of water that leaks from the tank in the time interval $[t, t + 1]$. Assume $o[t]$ is proportional 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 ($v[\cdot]$).

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.

5 Linear Systems (20 points)

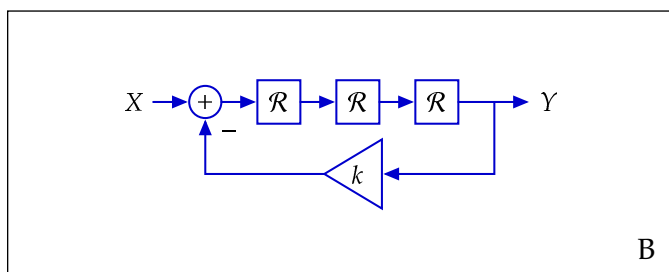
Part a. Several systems are illustrated below. The left column shows block diagrams. The right column shows system functions and pole locations for $k = -0.9$. Indicate which panel on the right (if any) corresponds to each panel on the left by drawing a straight line from A, B, C, and/or D to its partner. [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; $-0.41 \pm 0.59j$

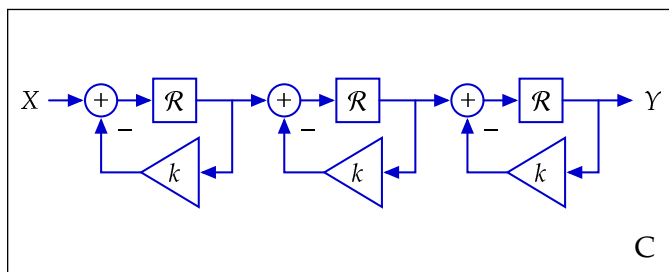
E



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

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

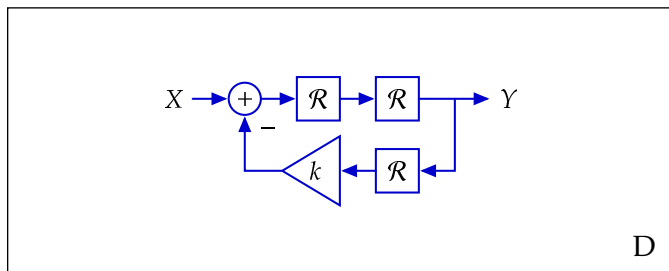
F



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

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

*Notice that there is a minus sign associated with the bottom input of each adder. This denotes that the input is subtracted (as if k were negative).

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.

