6.01 Midterm 2: Spring 2010

Name:	**Section:**

Solutions: Not correct for the make-up exam.

Enter all answers in the boxes provided.

During the exam you may:

- read any paper that you want to
- use a calculator

You may not

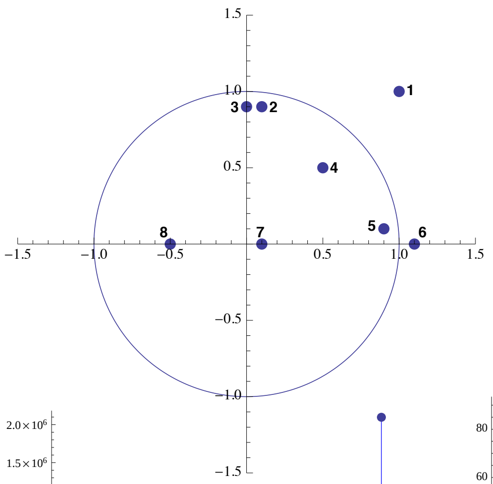
- use a computer, phone or music player

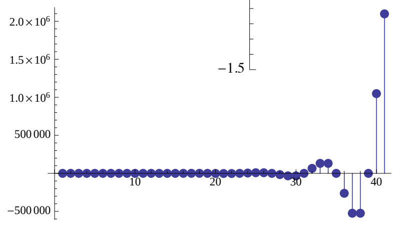
For staff use:

1.	/16
2.	/16
3.	/8
4.	/25
5.	/5
6.	/8
7.	/12
8.	/10
total:	/100

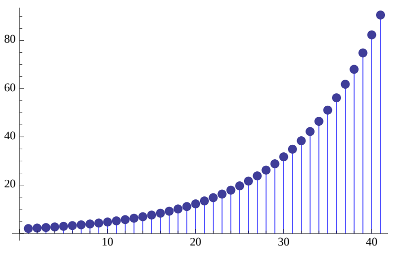
1 Pole Position (16 points)

The polar plot shows the dominant pole for several systems. Match each pole to the unit sample response of the system.

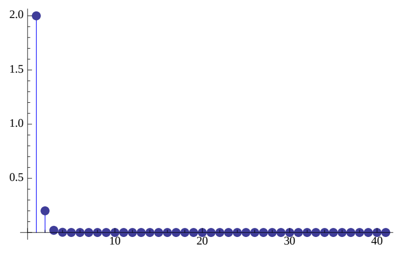




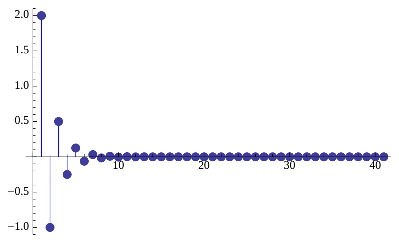
A = 1



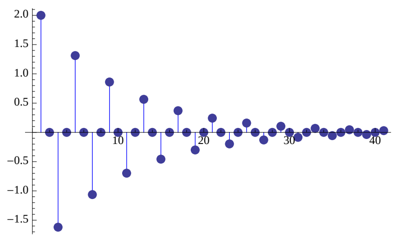
B = 6



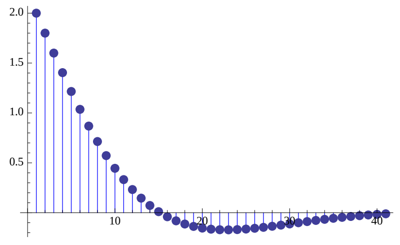
C = 7



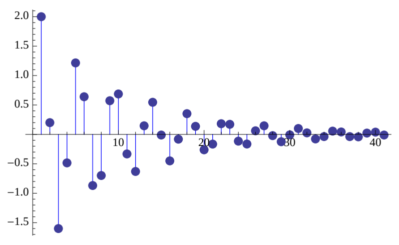
D = 8



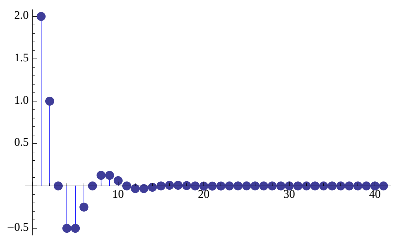
E = 3



F = 5



G = 2

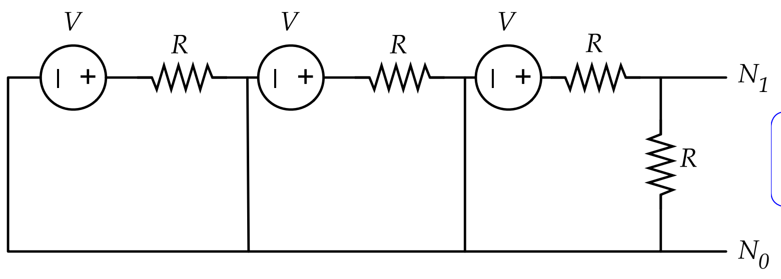


H = 4

2 What's the difference? (16 points)

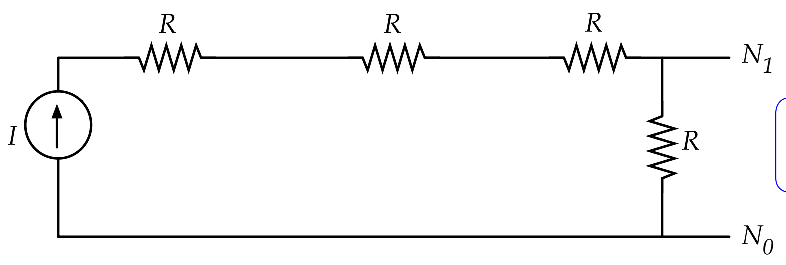
Assuming the voltage at node $N_0 = 0$, compute the voltage at node N_1 in each of these circuits.

2.1



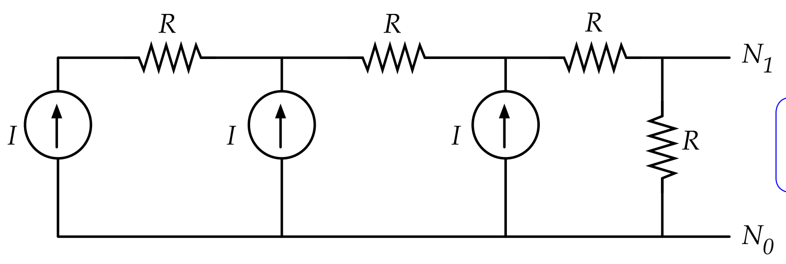
V/2

2.2



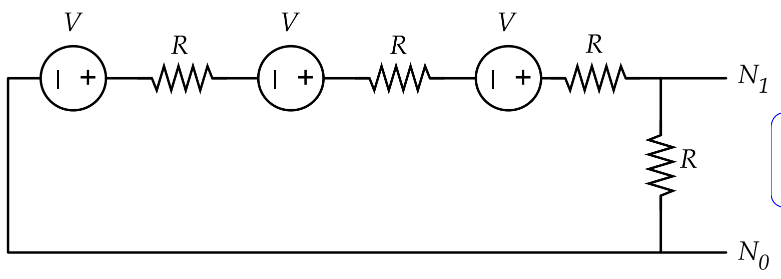
IR

2.3



3IR

2.4



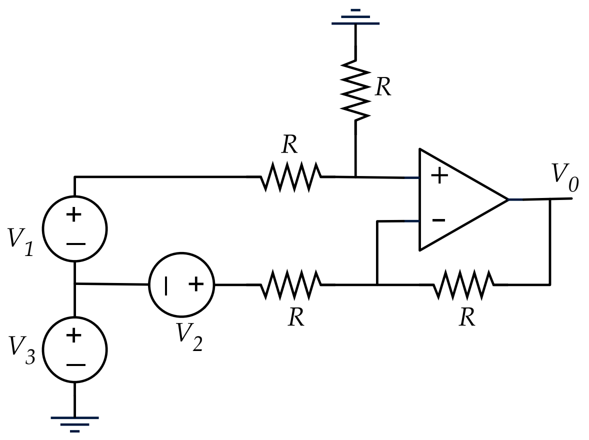
3V/4

Scratch Paper

3 Op Amps (8 points)

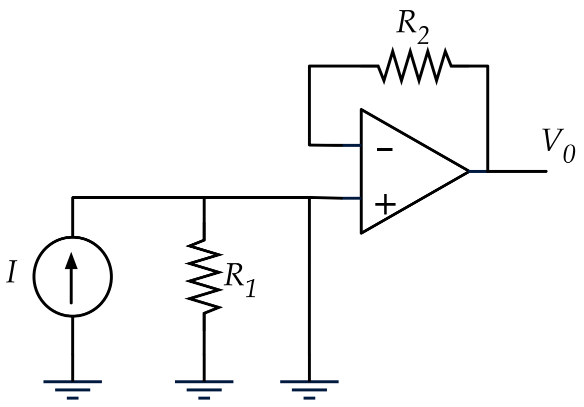
For each of the following circuits, give the output voltage V_0 as a function of the input voltage sources, input current sources, and resistances. Assume the op-amps are ideal.

3.1



$V_1 - V_2$

3.2



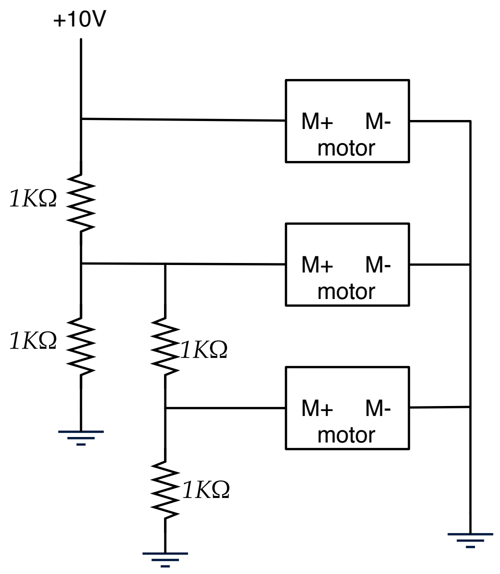
0

4 Motor control (25 points)

WhizzyLand engineers Kim, Pat, Jody, Chris, and Jamie are trying to design a controller for a display of three dancing robotic mice, using a 10V power supply and three motors. The first is supposed to spin as fast as possible (in one direction only), the second at half of the speed of the first, and the third at half of the speed of the second.

Each engineer has come up with a design, as shown below. Assume the motors have a resistance of approximately 5Ω and that rotational speed is proportional to voltage. For each design, indicate the voltage across each of the motors. Your answer only needs to be within 0.5V of the correct answer.

4.1 Jody

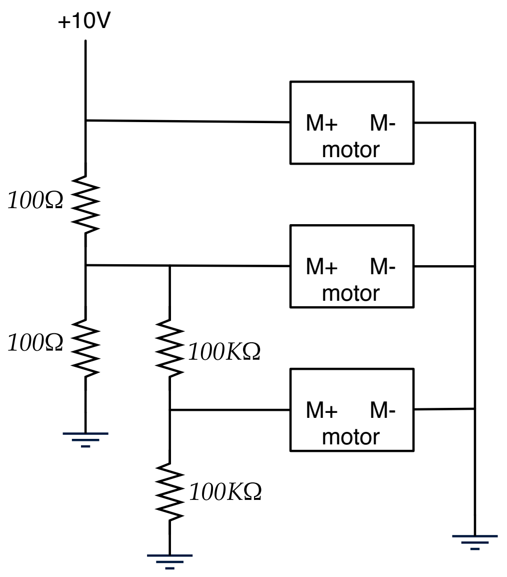


10

0.05

0

4.2 Chris

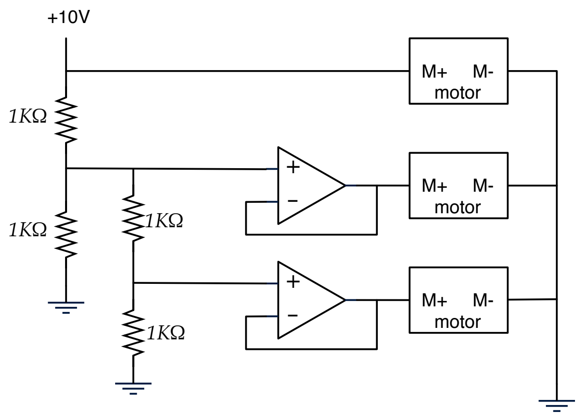


10

0.45

0

4.3 Pat

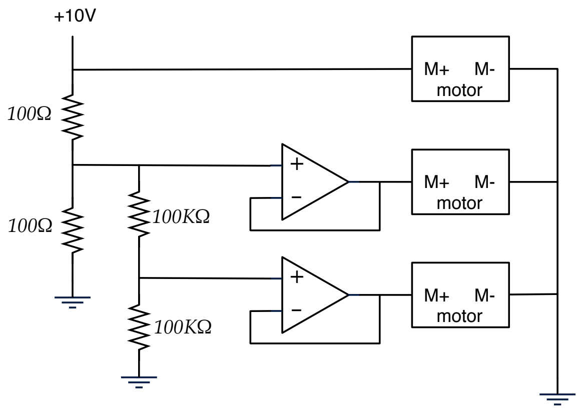


10

4

2

4.4 Kim

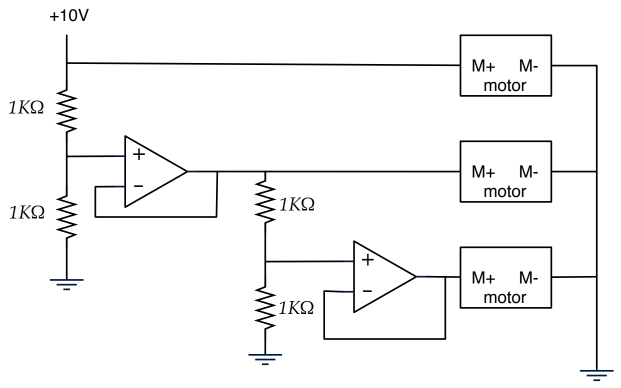


10

5

2.5

4.5 Jamie



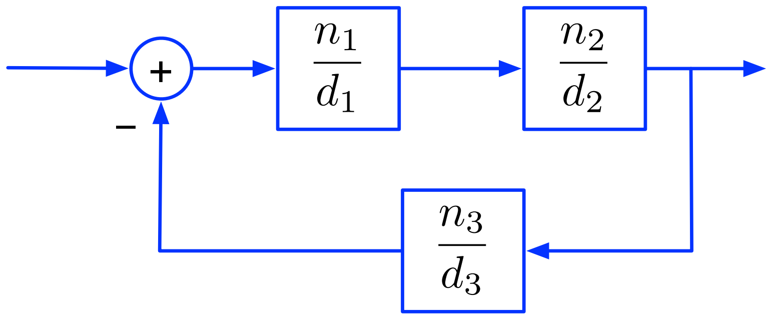
10

5

2.5

Scratch Paper

5 Composition (5 points)



Write an expression for the system function for this whole system, in terms of $n_1, d_1, n_2, d_2, n_3, d_3$, which are the numerator and denominator polynomials of the system function for each of the component systems.

$$\frac{d_3 n_1 n_2}{d_1 d_2 d_3 + n_1 n_2 n_3}$$

6 OOP (6 points)

If a procedure has no side effects but may be used multiple times, then a convenient concept for efficiency is called memoization – we keep track of whether the procedure has previously been called with a particular argument, and the value computed at that time. If it has already been computed, we just return the saved value, otherwise we do the computation, store away the value for future use, and return the value.

Here is an object oriented approach for memoization, with one piece missing. We assume that `proc`, the procedure to be "memoized", has a single argument.

```
class Memo:
    def __init__(self, proc):
        self.proc = proc
        self.history = {}
    def val(self,arg):
        if arg in self.history:
            print 'found', arg, 'in history'
            return self.history[arg]
        else:
            # YOUR ANSWER HERE
```

An example usage would be

```
>>> def sq(x): return x*x
>>> test = Memo(sq)
>>> test.val(2)
4
>>> test.val(3)
9
>>> test.val(3)
found 3 in history
9
>>> test.val(8)
64
>>> test.val(5)
25
>>> test.history
{8: 64, 2: 4, 3: 9, 5: 25}
```

6.1

Complete the definition of this class, by supplying the code for the else part of the val method. Make sure that you do not execute the procedure more than once for any given argument.

```
            self.history[arg] = self.proc(arg)
            return self.history[arg]
```

6.2

The approach above works with any procedure of one argument that does not have side-effects. However, if we know that the procedure has some additional structure, we can be even more efficient. Assume we know that our procedure takes a single number as an argument and that the procedure is **even**, that is, `proc(arg) == proc(-arg)`. Examples of even procedures are `abs`, `math.cos` and `sq` as defined above.

Write a definition for the `MemoEven` class. Your solution must use inheritance, and ensure that there is a `val` method defined. Do not change the definition of `Memo` or repeat any of the code it contains.

```
class MemoEven(Memo):
    def val(self, arg):
        return Memo.val(self, abs(arg))
```

7 Red vs Blue (12 points)

Let's build a voting tabulator for the national elections (except that we're going to assume that there are only 4 states). The definitions below initialize the instance and allow us to enter votes, by state, for each candidate.

We're going to keep the votes data in the dictionary `stateVotes`. There is an entry in `stateVotes` for each state. The value stored for a state is itself a dictionary that stores the votes from that state for each candidate. A simple example, for two states (`'MA'` and `'TX'`) and two candidates, `'O'` and `'M'`, would be:

```
{'MA': {'O':100, 'M':50}, 'TX': {'O':50, 'M':100}}

class Votes:
    def __init__(self):
        self.states = ['MA', 'TX', 'CA', 'FL']
        self.candidates = ['O', 'M', 'N']
        self.stateVotes = {}
    def addVotes(self, cand, votes, state):
        if not state in self.stateVotes:
            self.stateVotes[state] = {cand : votes}
        else:
            if not cand in self.stateVotes[state]:
                self.stateVotes[state][cand] = votes
            else:
                self.stateVotes[state][cand] += votes
```

7.1

Write a method for the `Votes` class, called `popularVotesTotal`, that computes the total number of votes for a candidate, across all states.

- Use a list comprehension, not a `for` loop.
- Use the Python `sum` procedure, which takes a list of numbers as input and returns the sum.

```
    def popularVotesTotal(self, cand):
        return sum([self.stateVotes[state][cand] for state in self.states])
```

7.2

Write a method for the `Votes` class, called `stateWinner`, that computes the winner for a state, that is, the candidate with the most votes in that state.

Use the procedure `argmaxDict(d)`, which is called with a dictionary `d` and returns the key in `d` whose associated value is highest.

```
def stateWinner(self, state):
    return argmaxDict(self.stateVotes[state])
```

7.3

Write a method for the `Votes` class, called `statesWon`, that takes a candidate as an argument and returns a list of the states that candidate won. Use the `stateWinner` method you just implemented.

```
def statesWon(self, cand):
    return [state for state in self.states if self.stateWinner(state) == cand]
```

7.4

Write a method for the `Votes` class, called `winnerOfMostStates`, that returns the candidate that won the most states. Use the `statesWon` method you just implemented. For full credit, use `util.argmax`.

If `l` is a list of items and `f` is a procedure that maps an item into a numeric score, then `util.argmax(l, f)` returns the element of `l` that has the highest score.

```
def winnerOfMostStates(self):
    return util.argmax(self.candidates,
                       lambda cand: len(self.statesWon(cand)))
```

8 Balanced Machine (10 points)

Write a state machine that counts the depth of nesting of parens in a string. It takes a character from the string as input and outputs an integer, indicating how many unmatched open parens there are (on or) before this character or `None` if the parens are unbalanced. Note that once the parens become unbalanced, all future outputs will be `None`. Here are some examples:

```
>>> bp = BalancedParens()
>>> bp.transduce('(ab (c (d (e)) f))')
[1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 3, 2, 2, 2, 1, 0]
>>> bp.transduce('(())  ((()))  ()()()')
[1, 2, 1, 0, 0, 0, 1, 2, 3, 2, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0]
>>> bp.transduce('(()))  ()()()')
[1, 2, 1, 0, None, None, None, None, None, None, None, None, None]
```

Your state machine should be a subclass of `sm.SM`.

```
class BalancedParens(sm.SM):
    startState = 0
    def getNextValues(self, state, inp):
        if state == None:
            return (None, None)
        elif inp == '(':
            return (state+1, state+1)
        elif inp == ')':
            if state == 0:
                return (None, None)
            else:
                return (state-1, state-1)
        else:
            return (state, state)
```