C200 Programming Assignment № 5

Dr. M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

October 20, 2024

Introduction

Due Date: 11:00 PM, Saturday, October 26th Submit your work to the Autograder `https://c200.luddy.indiana.edu/` and remember to add, commit and **push often** to GitHub **before the deadline. We do not accept late submissions.**

Instructions

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).

2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.

3. Make sure that the **code does not have infinite loop (that never exits)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.

4. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.

5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Recursion and Tail Recursion

In class for the second lecture we practiced recursion on this fuction:

$$h(n) \quad = \quad \begin{cases} 0 & n \leq 0 \\ 2n + h(n-1) + h(n-2) & o.w. \end{cases} \tag{1}$$

If we tried our first technique to implement tail recursion, we'd have this:

$$htr(n, acc0, acc1) \quad = \quad \begin{cases} [acc0, acc1][n] & n \leq 2 \\ htr(n-1, acc1, 2n + acc0 + acc1) & o.w. \end{cases} \tag{2}$$

The first ten values are:

```
1  h(0) = 1; htr(0) = 1
2  h(1) = 4; htr(1) = 4
3  h(2) = 9; htr(2) = 9
4  h(3) = 19; htr(3) = 19
5  h(4) = 36; htr(4) = 40
6  h(5) = 65; htr(5) = 79
7  h(6) = 113; htr(6) = 153
8  h(7) = 192; htr(7) = 288
9  h(8) = 321; htr(8) = 533
10 h(9) = 531; htr(9) = 971
```

Observe that for arguments 4,5,... the values differ. In class I said an alternative approach is using a while loop where we can manually build the stack frames. In some cases, the only while is to build bottom-up as we did with memoization. To give a simple example, suppose we want to implement tail recursion for factorial bottom-up. We add an additional variable for the state as shown here.

```
def f(n):
    if n:
        return n * f(n-1)
    else:
        return 1

def fw(n):
    if n in [0,1]:
        return 1
    else:
        acc = 1
        i = 2
        while i < n + 1:
            acc *= i
            i += 1
    return acc

def ftr(n,i=2,acc=1):
    if n in [0,1]:
        return 1
    elif i < n + 1:
        return ftr(n,i+1,acc*i)
    else:
        return acc

for n in range(10):
    print(n,f(n),fw(n),ftr(n))
```

produces:

```
0 1 1 1
1 1 1 1
2 2 2 2
3 6 6 6
4 24 24 24
5 120 120 120
6 720 720 720
7 5040 5040 5040
8 40320 40320 40320
9 362880 362880 362880
```

the while loop is emulated in the tail recursion. We added an additional variable i and counted up. In fact, n is never changed. This allowed for a bottom-up tail recursion. Take this into

account when solving the recursions.

$$h(n) = \begin{cases} 0 & n \leq 0 \\ 2n + h(n-1) + h(n-2) & o.w. \end{cases} \tag{3}$$

$$p(0) = 10000 \tag{4}$$

$$p(n) = p(n-1) + 0.02p(n-1) \tag{5}$$

$$c(1) = 9 \tag{6}$$

$$c(n) = 9c(n-1) + 10^{n-1} - c(n-1) \tag{7}$$

$$d(0) = 1 \tag{8}$$

$$d(n) = 3d(n-1) + 1 \tag{9}$$

$$\tag{10}$$

The following code:

```
  for i in range(5):
        print(f"n = {i}")
        print("c", c(i),ctr(i),cw(i))
        print("p", p(i), ptr(i))
        print("h", h(i), hmemo(i), hw(i), htr(i))
        print('d', d(i), dtr(i))
```

produces:

```
n = 0
c 9 9 9
p 10000 1000
h 1 1 1 1
d 1 1
n = 1
c 9 9 9
p 10200.0 1020.0
h 4 4 4 4
d 4 4
n = 2
c 82 82 82
p 10404.0 1040.4
h 9 9 9 9
d 13 13
n = 3
c 756 756 756
p 10612.08 1061.208
h 19 19 19 19
d 40 40
n = 4
c 7048 7048 7048
p 10824.3216 1082.43216
h 36 36 36 36
d 121 121
```

Deliverables for Problem 1

- Implement the function h using recursion, memoization, while-loop, and tail recursion
- Implement the functions p, d using recursion and tail recursion
- Implement the function c recursively, tail recursively, and with a while-loop

Problem 2: Starting quantum computing and quantitative finance

Quantum Computing is a different model of computation imagined by the physicist Feynman. Instead of whole numbers, like we use in the Turing model, it uses complex numbers. Complex numbers, if you remember, are actually pairs of real numbers written as:

$$\text{complex number} \quad = \quad x \pm y\, i \tag{11}$$

where $x, y \in \mathbb{R}$ (they're just numbers) and there's a lone $i = \sqrt{-1}$. The x is called the real part and the y is called the imaginary part. For example,

$$x^2 + 1 \quad = \quad 0 \tag{12}$$

$$x \quad = \quad \sqrt{-1} = i \tag{13}$$

Then

$$x^2 + 1 \quad = \quad (\sqrt{-1})^2 + 1 = -1 + 1 = 0 \tag{14}$$

In Python we use the function complex(x,y) to form a complex number $(x \pm yj)$ where j represents i and there's no space between the $\pm$ sign and numbers. Let's write the function for the values shown in 12, build the complex number, and show it's a solution.

```
>>> def f(x):
...     return x**2 + 1
...
>>> root = complex(0,1)
>>> f(root)
0j
>>> f(root) == 0
True
>>> f(root).real
0.0
>>> f(root).imag
0.0
```

For a quadratic from last homework, you know that the answer is either real or complex. To remind you, the answer is complex when the discriminant is negative. Fig. 1. visualizes what's happening–the curve for complex roots does **intersect** the abscissa. Observe that you'll get zero (on the x-axis or *abscissa*) if you put either of these two x values there. Sometimes the discriminant (the value in the squareroot) is negative. This is where i comes in. We simply multiply the value by -1, thereby allowing us to take the squareroot, but then we append an i to signal it's imaginary.

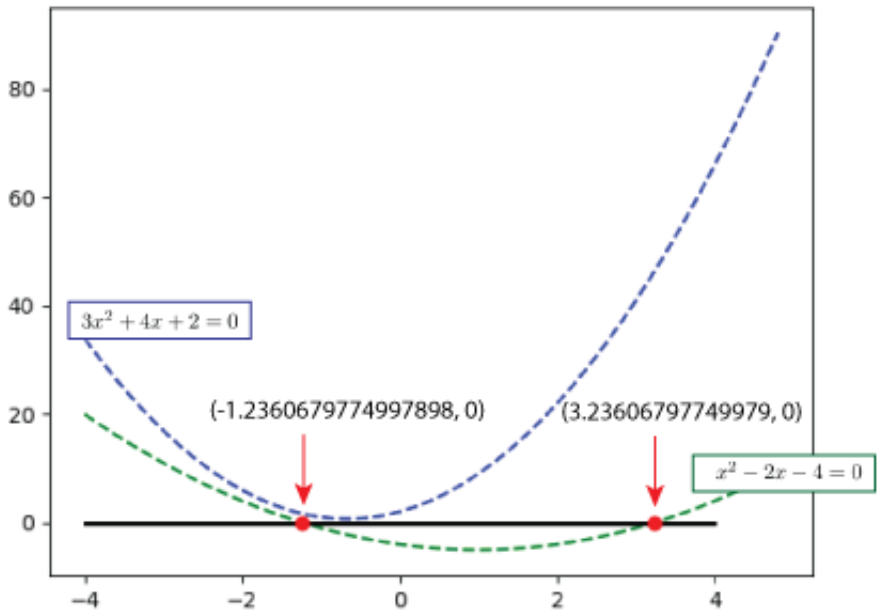


Figure 1: The red dots (with red arrows) are where your solutions are to $x^2 - 2x - 4 = 0$. The dash curve is the function itself. The horizontal line just makes it easier to see the x-axis. The two values are -1.2360679774997898 and 3.23606797749979. For the other function $3x^2+4x+2=0$ shown in blue, because it has an imaginary part, it cannot cross the axis. You'll use the matplotlib library you were introduced last homework to actually plot this!

For example, suppose we have $3x^2 + 4x + 2 = 0$. Then we find, after some algebra:

$$
\begin{aligned}
x &= \frac{-4 \pm \sqrt{-8}}{6} && (15)\\
&= \frac{-4 \pm \sqrt{-2 \times 4}}{6} && (16)\\
&= \frac{-4 \pm \sqrt{-2} \times \sqrt{4}}{6} && (17)\\
&= \frac{-4 \pm 2\sqrt{-2}}{6} && (18)\\
&= -\tfrac{2}{3} \pm \tfrac{\sqrt{2}}{3} i && (19)\\
&= (-\tfrac{2}{3} + \tfrac{\sqrt{2}}{3} i, -\tfrac{2}{3} - \tfrac{\sqrt{2}}{3} i) && (20)
\end{aligned}
$$

In Python we would write:

```
>>> import math
>>> complex(-2/3,math.sqrt(2)/3),complex(-2/3,-math.sqrt(2)/3)
((-0.6666666666666666+0.47140452079103173j), ↩
    (-0.6666666666666666-0.47140452079103173j))
>>> x = round(-2/3,2)
>>> y = round(math.sqrt(2)/3)
>>> complex(x,y), complex(x,-y)
((-0.67+0j), (-0.67+0j))
>>> def f(x):
...     return 3*(x**2) + 4*x + 2
```

```
...
>>> f(complex(x,y))
(0.6667000000000001+0j)
>>> f(complex(x,-y))
(0.6667000000000001+0j)
>>> f(complex(-2/3,math.sqrt(2)/3))
0j
```

Observe the difference between using the full root and only to two decimal places. You'll write a function q(t) where:

$$q((a,b,c)) \quad = \quad \begin{cases} (r_0, r_1) & \text{real roots if } b^2 - 4ac \geq 0 \\ (c_0, c_1) & \text{complex roots if } b^2 - 4ac < 0 \end{cases} \tag{21}$$

When returning the roots, round to two decimal places, *i.e.*, round(x,2). Here are a few examples:

$$q((3,4,2)) \quad = \quad ((-0.67 + 0.47j), (-0.67 - 0.47j)) \tag{22}$$

$$q((1,3,-4)) \quad = \quad (-4.0, 1.0) \tag{23}$$

$$q((1,-2,-4)) \quad = \quad (-1.24, 3.24) \tag{24}$$

Deliverables for Problem 2

- Complete the function.
- Return the smaller root as the first value followed by the other root.
- Don't forget to round to two decimal places.

Problem 3: Completing the Square

When the roots of a quadratic are real, we can transform the quadratic $ax^2 + bx + c = 0$ into a simpler form and find the roots more easily:

$$ax^2 + bx + c = (x + m)^2 + n$$

To find the roots then:

$$\begin{aligned} (x+m)^2 + n &= 0 \\ (x+m)^2 &= -n \\ x + m &= \pm\sqrt{-n} \\ x &= -m \pm \sqrt{-n} \end{aligned}$$

Through some simple algebra we find that

$$\begin{aligned} m &= \frac{b}{2a} \\ n &= c - \frac{b^2}{4a} \end{aligned}$$

We can then call our transform function that takes a, b, c and returns m, n as seen below:

```
1 def c_s(coefficients):
2     pass
3
4 def q_(coefficients):
5     m,n = c_s(coefficients)
6     pass
```

Deliverables for Problem 3

- Complete the functions.
- Round the output from c_s() and q_() function to 2 decimal places.
- The q_ function must call the c_s function.

Problem 4: Toward Statistical Analysis

In this problem, you'll write fundamental statistical functions. We assume a list of numbers $lst = [x_0, x_1, \ldots, x_n]$.

$$\begin{aligned} mean(lst) &= (x_0 + x_1 + \cdots + x_n)/\text{len}(lst) && (25)\\ \mu &= mean(lst) && (26)\\ var(lst) &= \frac{1}{\text{len}(lst)}((x_0 - \mu)^2 + (x_1 - \mu)^2 + \cdots + (x_n - \mu)^2) && (27)\\ std(lst) &= \sqrt{var(lst)} && (28) \end{aligned}$$

For example, $lst = [1, 3, 3, 2, 9, 10]$, rounding to two places

$$\begin{aligned} mean(lst) &= 4.67 && (29)\\ var(lst) &= 12.22 && (30)\\ std(lst) &= 3.5 && (31) \end{aligned}$$

The last function mean_centered takes a list of numbers $lst = [x_0, x_1, \ldots, x_n]$ and returns a new list $[x_0 - \mu, x_1 - \mu, \ldots, x_n - \mu]$. An interesting feature of the mean-centered list is that if you try to calculate its mean, it's zero:

$$\begin{aligned} \mu &= (x_0 + x_1 + \cdots + x_n)/n && (32)\\ lst &= [x_0 - \mu, x_1 - \mu, \ldots x_n - \mu] && (33)\\ mean(lst) &= ((x_0 - \mu) + \cdots (x_n - \mu))/n && (34)\\ &= ((x_0 + \ldots + x_n) + n\mu)/n && (35)\\ &= \mu - \mu = 0 && (36) \end{aligned}$$

Using the same list we have

$$mean(mean\_centered(lst)) = -0.0 = 0 \quad (37)$$

Deliverables for Problem 4

- Complete the functions.
- Round-up to 2 decimal places the ouptut of mean, var and std functions.

Problem 5: Market Equilibrium

When modeling market behavior we use two curves to indicate supply and demand. You can also think of supply as quantity and demand as want. When the two curves intersect (the common point where they cross), see Fig. 2, there is a point that satisfies both equations. Review the problem of intersecting two lines. Here we have specifically two quadratic functions. Assume

$$s(x) = -.025x^2 - .5x + 60 \tag{38}$$

$$d(x) = 0.02x^2 + .6x + 20 \tag{39}$$

for supply s and demand d. We have a function $equi$ that takes the coefficients of s, d and returns the solution:

$$equi((-.025, -.5, 60), (0.02, .6, 20)) = (20.0, -44.44) \tag{40}$$

Our solution returns the roots to a quadratic equation. Since s, d models the presence of items, only 20 makes sense. HINT: use your q function in Problem 2.

Deliverables for Problem 5

- Complete the function.
- Use q in Problem 2 to make the solution very quick.

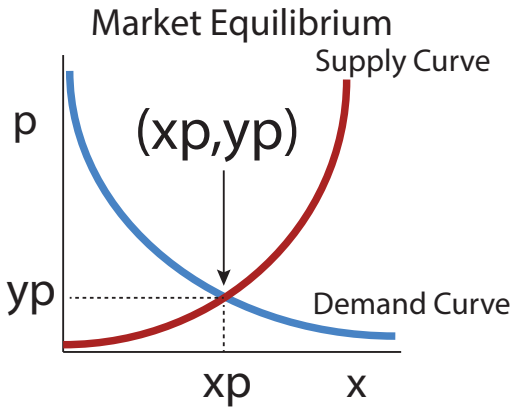


Figure 2: Market equilibrium with supply and demand

Problem 6: Computing Love

To match people, companies use trigonometry. Assume you have a list of people $[p_0, p_1, \ldots, p_m]$, we find the two different people who have the smallest angle. The way we calculate this is to assume each person is a list (mathematically vector) of 0s and 1s. We need three basic functions: inner product, magnitude, and $\cos^{-1}$. We assume two lists $x = [x_0, x_1, \ldots, x_n], y = [y_0, y_1, \ldots, y_n]$ of 0s and 1s of the same length:

$$inner\_prod(x, y) = x_0y_0 + x_1y_1 + \cdots + x_ny_n \quad (41)$$

$$mag(x) = \sqrt{inner\_prod(x, x)} \quad (42)$$

For the last function (where we calculate the angle), we know that:

$$\cos(\theta) = \frac{inner\_prod(x, y)}{mag(x)mag(y)} \quad (43)$$

In class we learned that we can invert a function and the inverted function is denoted as f^{-1}. So we can:

$$\cos^{-1}(\cos(\theta)) = \cos^{-1}(\frac{inner\_prod(x, y)}{mag(x)mag(y)}) \quad (44)$$

$$\theta = \cos^{-1}(\frac{inner\_prod(x, y)}{mag(x)mag(y)}) \quad (45)$$

The math module has math.acos() for $\cos^{-1}()$. Further, Python returns θ in radians. We have:

$$\pi \text{ radians} = 180 \text{ degrees} \quad (46)$$

Thus, to convert from radians to degree, you **must** multiply your answer by $\frac{180}{\pi}$.

Your task is to first complete the functions “inner_prd“, “mag“ and “angle“ and then use them to write the “match“ and “best_match“ functions. The match function takes a list of people where each person is a list of 0s 1s, and returns all unique pairs with the angle in degrees. Note that “match“ returns the output in the format: [[person 1, person2, angle], [person2, person3, angle], [person1, person3, angle]], where each person i.e., person1, person2-is also a list for example, [1,1,1] or [0,1,0]. So it returns a list of lists. For “angle“ function-round your answer to 2 decimal digits.

```
people0 = [[0,1,1],[1,0,0],[1,1,1]]
print(match(people0))
print(best_match(match(people0)))
```

gives an output

```
[[[0, 1, 1], [1, 0, 0], 90.0],
 [[0, 1, 1], [1, 1, 1], 35.26],
 [[1, 0, 0], [1, 1, 1], 54.74]]
([0, 1, 1], [1, 1, 1], 35.26)
```

We're displaying the result of match so you can see the structure. Each unique pair as an angle. For example:

$$
\begin{aligned}
inner\_prod([1,0,0],[1,1,1]) &= 1(1)+0(1)+0(1)=1 && (47)\\
mag([1,0,0]) &= \sqrt{inner\_prod([1,0,0],[1,0,0])}=\sqrt{1}=1 && (48)\\
mag([1,1,1]) &= \sqrt{inner\_prod([1,1,1],[1,1,1])}=\sqrt{3} && (49)\\
\theta &= (\frac{180}{\pi})\mathrm{math.acos}(\frac{1}{1\sqrt{3}}) \approx 54.74 && (50)
\end{aligned}
$$

Deliverables for Problem 6

- Complete the functions.
- Keep in mind that function angle() utilizes equation 45.
- Round the output of angle to two decimal places.

Problem 7: Solving a Small Linear System

A matrix, to remind you, is a rectangular collection of numbers. Matrices play vital roles in AI, finance, physics, *etc.*. A 2×2 square matrix is the smallest useful matrix:

$$\mathbf{x} = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \tag{51}$$

A matrix can be effectively captured with Python lists:

```
1 x = [[a,b],[c,d]]
```

The *determinant* of a matrix is a real number that indicates certain properties. It's calculated by finding the difference between diagonals:

$$\det(\mathbf{x}) = ad - bc \tag{52}$$

Your first task will be to implement a function det() that takes a 2×2 matrix and returns the determinant. For example, this code

```
1 def determinant(matrix):
2     (a,b),(c,d) = matrix
3     pass
4
5 print(determinant([[1,2],[2,3]]))
```

has output

```
1 -1
```

Observe that we were able to separate the rows from the matrix! You should experiment and remove the parentheses and observe the behavior. Given two lines

$$ax + by = c \tag{53}$$

$$dx + ey = f \tag{54}$$

we are interested in determining their intersection; that is, so they share a solution $(x*, y*)$. We can use the determinants to find the answer easily. We can consider this a collection of three columns (matrix 1×2):

$$\mathsf{c}_0 = \begin{bmatrix} a \\ d \end{bmatrix} \tag{55}$$

$$\mathsf{c}_1 = \begin{bmatrix} b \\ e \end{bmatrix} \tag{56}$$

$$\mathsf{c}_2 = \begin{bmatrix} c \\ f \end{bmatrix} \tag{57}$$

We can form a new square matrix by combining columns:

$$\mathbf{a} = [\mathsf{c}_2\ \mathsf{c}_1] \tag{58}$$

$$\mathbf{b} = [\mathsf{c}_0\ \mathsf{c}_1] \tag{59}$$

$$\mathbf{d} = [\mathsf{c}_0\ \mathsf{c}_2] \tag{60}$$

Then the solution $(x*, y*)$ is

$$x* = \frac{\mathsf{det}(\mathbf{a})}{\mathsf{det}(\mathbf{b})} \tag{61}$$

$$y* = \frac{\mathsf{det}(\mathbf{d})}{\mathsf{det}(\mathbf{b})} \tag{62}$$

Working through an example:

$$2x + 4y = 11 \tag{63}$$

$$-5x + 3y = 5 \tag{64}$$

$$\mathsf{c}_0 = \begin{bmatrix} 2 \\ -5 \end{bmatrix} \tag{65}$$

$$\mathsf{c}_1 = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \tag{66}$$

$$\mathsf{c}_2 = \begin{bmatrix} 11 \\ 5 \end{bmatrix} \tag{67}$$

$$\mathbf{a} = \begin{bmatrix} 11 & 4 \\ 5 & 3 \end{bmatrix} \tag{68}$$

$$\mathbf{b} = \begin{bmatrix} 2 & 4 \\ -5 & 3 \end{bmatrix} \tag{69}$$

$$\mathbf{d} = \begin{bmatrix} 2 & 11 \\ -5 & 5 \end{bmatrix} \tag{70}$$

Then

$$x* = \frac{11(3) - 4(5)}{2(3) + 5(4)} = \frac{13}{26} = \tfrac{1}{2} \tag{71}$$

$$y* = \frac{2(5) + 5(11)}{26} = \frac{65}{26} = \frac{5}{3} \tag{72}$$

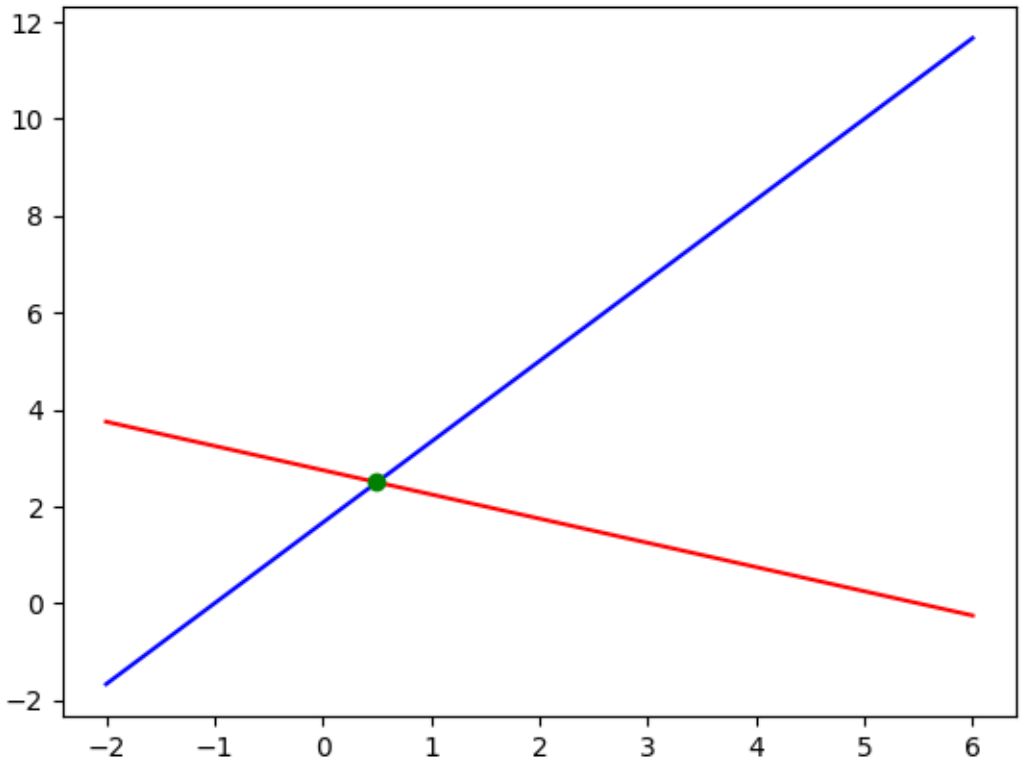
The following code:

```
1 eq1,eq2 = [[2,4,11],[-5,3,5]]
2 print(solve(eq1,eq2))
```

produces:

```
1 (0.5, 2.5)
```

When we plot the image, we see that the intersection (green dot) agrees with our answer.



The code to create the image is very small and uses a module matplotlib (with numpy) that we'll speak to a bit later.

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  # the first line
5  def f_1(x):
6      return (1/4)*(-2*x + 11)
7
8  # the second line
9  def f_2(x):
10     return (1/3)*(5*x + 5)
11
12 x_star,y_star = solve(eq1,eq2) #you must implement this function
13
14 x = np.linspace(-2,6,100)
15 plt.plot(x,f_1(x),'r') #plots the first line
16 plt.plot(x,f_2(x),'b') #plots the second line
17 plt.plot(x_star,y_star,'go') #plots the green point
18 plt.show()
```

Deliverables for Problem 7

- Complete the functions determinant and solve
- Round $x*, y*$ to two decimal places
- We will *not* check when the determinant is zero

Problem 8: City Block Distance

For this problem, we will only consider integer values. When calculating distance, typically Euclidean distance (yielding a decimal) is used:

$$d(p,q) = \sqrt{\sum_{1}^{n}(p_i - q_i)^2} \tag{73}$$

In practical settings, however, we need to adhere to the context. For travel in a city, for example, we must use blocks. In this problem we assume that blocks are squares. Each intersection represents the joining of four roads. Distance as city blocks is:

$$d(p,q) = \sum_{1}^{n}|p_i - q_i| \tag{74}$$

Consider Fig. 3. that shows the city block distance between the post office (red) and the recently chapter 11 TGIF (green). The path length is three. There are several routes that are three blocks, but that is distance you'll have to travel. In this problem, you'll implement three functions:

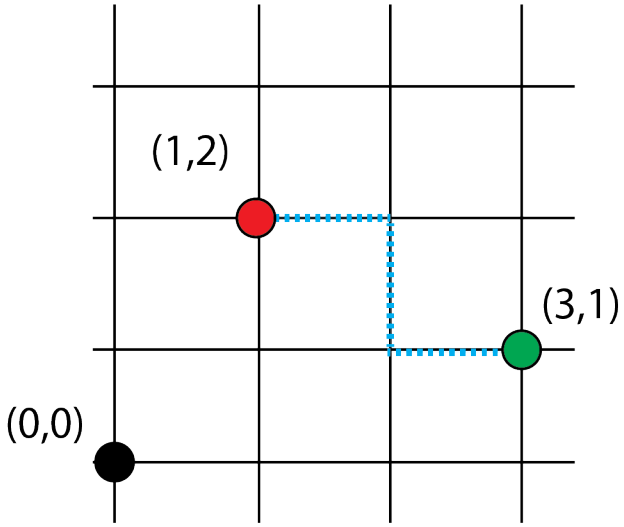


Figure 3: We have a city with straight, square roads. The center of town is at the origin (0,0). The city block distance between red and green is $d(p,q) = |1-3| + |2-1| = 3$. The blue dotted lines gives one of these paths; there are several so long as the number of blocks is three.

- `block_distance(p0,p1)` that gives city block distance between points `p0,p1`;
- `get_points(center,bd)` that generates all the points whose city block distance is `bd` or less;
- `intersection(x,y)` that yields the shared points in the lists `x,y` of points.

We're including a visualization to make the problem easier. The following code:

```
1
2 A = ((0,-1),2)
```

```
B = ((0,1),1)
C = ((4,4),1)
p = get_points(*A)
q = get_points(*B)
r = intersection(p,q)
s = get_points(*C)
t = intersection(s,q)

for points in p,q,r,s:
    print(points)

#begin matplotlib code
color = 'rgbmy'

for i,pts in enumerate([p,q,r,s,t]):
    plt.plot([x for x,_ in pts],[y for _,y in pts],color[i] + 'o')

plt.gca().legend(("A: ((0,-1),2)", "B: ((0,1),1)", r"$\mathsf{A}\cap\mathsf{B}$","C: ((0,1),1)", r"$\mathsf{B}\cap\mathsf{C}$"))
plt.axis([-7, 7, -7, 7])
plt.grid()
plt.gca().set_aspect("equal")

plt.grid(True)
plt.title("City with square streets.")
plt.show()
#end matplotlib code
```

produces:

```
[(-2, -1), (-1, -2), (-1, -1), (-1, 0), (0, -3), (0, -2), (0, -1), (0, 0), (0, 1), (1, -2), (1, -1), (1, 0), (2, -1)]
[(-1, 1), (0, 0), (0, 1), (0, 2), (1, 1)]
[(0, 0), (0, 1)]
[(3, 4), (4, 3), (4, 4), (4, 5), (5, 4)]
```

with visualization:

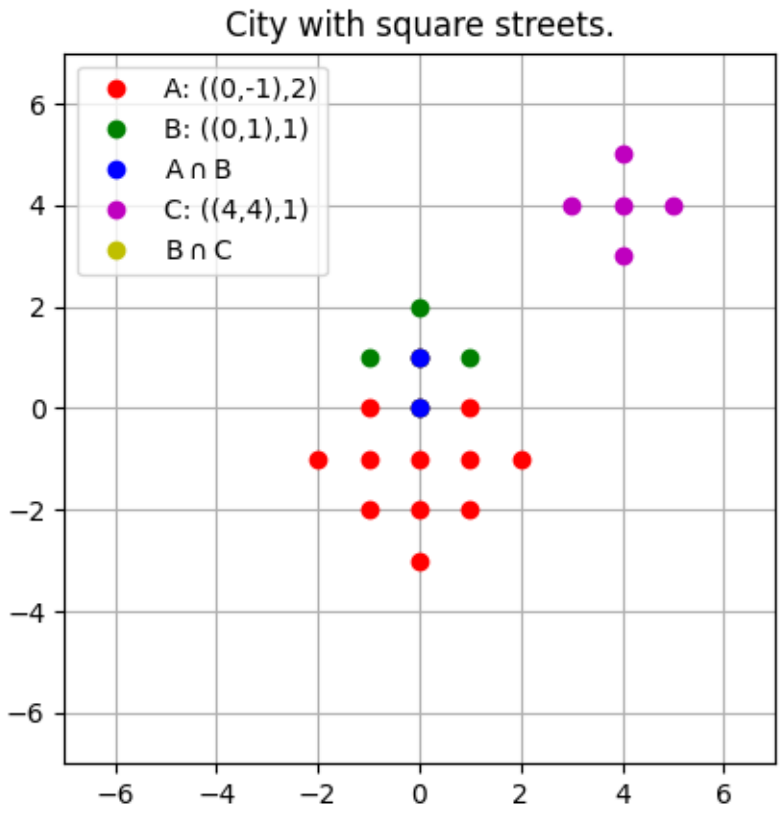


Figure 4: Observe no intersection between B,C.

Deliverables for Problem 8

- Complete the three functions.
- Implement `intersection(x,y)` as a single `return` statement using list comprehension. The function will look like this:

```
1 def intersection(x,y):
2     return [] #list comprehension
```

- Hint: *It is likely easier to generate more points then use the city block distance to select the ones that are within the distance than to code generating the exact points.*
- The order of the points is not important.

Problem 9: Geometric Progression

A geometric progression is described by $ar^0, ar^1, ar^2, \ldots$ for positive integers a, r. The ratio of successive terms is a constant *viz.*, $\frac{ar^1}{ar^0}, \frac{ar^2}{ar^1}, \ldots$. Thus, given a (finite) sequence we can determine whether it is geometric or not. In this problem you'll implement a function is_geometric_sequence(lst) that takes a possibly empty list of numbers (If the list is empty, return False). If there are at least three, then you can determine if it is a geometric series. For example,

```
1 data = [[1,2,4,6],[2,4,8,16],[10,30,90,270,810,2430]]
2 for d in data:
3     print(is_geometric_sequence(d))
```

produces

```
1 False
2 True
3 True
```

Deliverables for Problem 9

- Complete the function.
- As always you cannot use any functions not discussed in the class.

Problem 10: Portfolio Valuation

In this problem, you'll implement a function value(portfolio, market) that takes a person's portfolio of stocks and the current market and determines the value as a percent. A stock is a portion of ownership of a business. Since it's a portion, it is a fraction of what the business is worth in total. A stock as a price per share as a dollar amount and the number of shares (how much of the portion is owned). For example, if you have a stock that you purchased for $2.25/share and you purchased 11 shares then $\$2.25 \times 11 = \$24.75$ in total. A portfolio consists of the stock name, share price that was purchased, and the number of shares. We'll model this a dictionary:

```
1 portfolios =  {'A':{'stock':{'x':(41.45,45),'y':(22.20,1000)}},
2                      'B':{'stock':{'x':(33.45,15),'y':(12.20,400)}}}
```

shows two portfolios, one for 'A' and one for 'B'. 'A' has 45 shares of 'x' valued at $41.45 and 'y' valued at $22.20 with 1000 shares. The total value is: $41.45(45) + $22.20(1000) = $24065.25. The market value for stocks changes. In this example, our market is:

```
1 market = {'x':43.00, 'y':22.50}
```

This means that 'A's holdings in 'x' is worth now $\$43 \times 45 = \$1935.0$ instead of $\$41.45 \times 45 = 1865.25$. The value of a portfolio is the ratio of $\frac{\text{Market}-\text{Portfolio}}{\text{Portfolio}}$. If we were solely looking at the 'x' stock, then

$$\frac{\text{Market} - \text{Portfolio}}{\text{Portfolio}} = \frac{43.00(45) - 41.45(45)}{41.45(45)} \tag{75}$$

$$\approx 3.7\% \tag{76}$$

which means the value today is 3.7% more. Implement the function value(portfolio, market) that determines the change in percentage of worth. For example,

```
1 portfolios =  {'A':{'stock':{'x':(41.45,45),'y':(22.20,1000)}},'B':{'↩
    stock':{'x':(33.45,15),'y':(12.20,400)}}}
2
3 market = {'x':43.00, 'y':22.50}
4
5 for name, portfolio in portfolios.items():
6     print(f"{name} {value(portfolio,market)}")
```

produces

```
1 A 2.0
2 B 79.0
```

that gives 2% for A and 79% for B.

Deliverables for Problem 10

- Complete the function.
- The function returns a percent.
- Round to two decimal places.

Problem 11: Smoothing Data

When data is temporal (ordered by time), we sometimes want to “smooth" it by giving a simple average. The simplest is taking two consecutive values and finding their average. Given a (possibly empty) list of numbers lst $= [x_0, x_1, \ldots, x_n]$ the function smooth(lst) returns:

- The string “AveError: list empty" if the list is empty
- lst if list only contains one number
- a new list of numbers $[y_0, y_1, \ldots, y_{n-1}]$ such that $y_i = \frac{x_i + x_{i+1}}{2}$

The following code

```
1 data = [[], [1],[1,2],[1,2,2,3],[0,2,4,6,8]]
2 for d in data:
3     print(smooth(d))
```

produces

```
1 AveError: list empty
2 [1]
3 [1.5]
4 [1.5, 2.0, 2.5]
5 [1.0, 3.0, 5.0, 7.0]
```

Deliverables for Problem 11

- Complete the function.
- Round to two decimal places.

12: Recursive Hacking

This was suggested by a student in our class. You will implement a function `break_code(secret_code, combinations)` that breaks the code using the combinations. Both are varying:

```
def m(useless_parameter=0):
    rn.seed(useless_parameter+1)
    combinations = "".join([chr(i) for i in range(ord('0'),ord('0') + ↩
       rn.randint(5,35))])
    secret_code = ""
    for _ in range(rn.randint(4,8 + rn.randint(0,20))):
        secret_code += rn.choice(combinations)

    return secret_code, break_code(secret_code, combinations)

for i in range(5):
    print(m())
```

produces:

```
('417776', '417776')
('5G:C', '5G:C')
('2597:919074833;78876:', '2597:919074833;78876:')
(';6721', ';6721')
(';FGD@0>7D153;?7', ';FGD@0>7D153;?7')
```

Deliverables for Problem 12

- Complete the function.
- Implement it recursively.

Programming partners

cedrchen@iu.edu, mt87@iu.edu
egilmour@iu.edu, mitcal@iu.edu
nganesh@iu.edu, jewach@iu.edu
aare@iu.edu, cpmathew@iu.edu
enkarl@iu.edu, ozsachs@iu.edu
sydforem@iu.edu, nmenne@iu.edu
chjflor@iu.edu, rheashah@iu.edu
evarcarm@iu.edu, davepais@iu.edu
ahsans@iu.edu, aluckhau@iu.edu
alneyadk@iu.edu, olesmith@iu.edu
svdatla@iu.edu, evanmayo@iu.edu
heoji@iu.edu, pp22@iu.edu
natkling@iu.edu, lydiwang@iu.edu
aarepal@iu.edu, olsoncs@iu.edu
maboncos@iu.edu, corawang@iu.edu
mahchaud@iu.edu, hsh1@iu.edu
elscheng@iu.edu, hmahapat@iu.edu
idasilva@iu.edu, roldenbu@iu.edu
drussow@iu.edu, mjseymou@iu.edu
vicbarre@iu.edu, rvasude@iu.edu
jagoodal@iu.edu, elwesley@iu.edu
smhamrah@iu.edu, navann@iu.edu
zkabacin@iu.edu, kzerbino@iu.edu
clickc@iu.edu, jiajtang@iu.edu
agrief@iu.edu, jwubelho@iu.edu
iblay@iu.edu, jw363@iu.edu
akoduru@iu.edu, kalerobi@iu.edu
clarkina@iu.edu, anangla@iu.edu
alurs@iu.edu, jacloise@iu.edu
ethhanna@iu.edu, ctsvetan@iu.edu
felkhatt@iu.edu, liaobrie@iu.edu
coobratt@iu.edu, kiloscot@iu.edu
cdasari@iu.edu, ydpatil@iu.edu
nicdelg@iu.edu, kalovert@iu.edu
nafike@iu.edu, nguyen17@iu.edu
abondada@iu.edu, mclindbe@iu.edu
ldeel@iu.edu, bgtruong@iu.edu
meiyhe@iu.edu, sunathan@iu.edu
jdkakare@iu.edu, kpazdur@iu.edu

crigonza@iu.edu, risvphil@iu.edu
adatwani@iu.edu, nmeely@iu.edu
akeshav@iu.edu, robreese@iu.edu
jlaytin@iu.edu, angzamor@iu.edu
mahichan@iu.edu, manpate@iu.edu
mrb9@iu.edu, bobbyun@iu.edu
bahls@iu.edu, hailreid@iu.edu
rogoenka@iu.edu, samyadav@iu.edu
laigrant@iu.edu, tnerkar@iu.edu
jkuk@iu.edu, nealsing@iu.edu
yaljabri@iu.edu, prmundra@iu.edu
mjfarrin@iu.edu, fsafianu@iu.edu
carthugh@iu.edu, missavin@iu.edu
jaeblank@iu.edu, esspatel@iu.edu
evbutts@iu.edu, ayorgen@iu.edu
drmiguth@iu.edu, ssanjee@iu.edu
twchase@iu.edu, emasseng@iu.edu
bskendal@iu.edu, gsackie@iu.edu
kkunshet@iu.edu, lukthiel@iu.edu
florech@iu.edu, aidsteph@iu.edu
jk314@iu.edu, mjo4@iu.edu
jb108@iu.edu, jopantoj@iu.edu
kbussel@iu.edu, mawlorch@iu.edu
agotanco@iu.edu, negia@iu.edu
iyersan@iu.edu, aposton@iu.edu
damaatki@iu.edu, martibr@iu.edu
goldfusl@iu.edu, omwells@iu.edu
radutta@iu.edu, sstickl@iu.edu
nfarzane@iu.edu, fasrasoo@iu.edu
adodaro@iu.edu, sxanders@iu.edu
ethdeatr@iu.edu, bsati@iu.edu
jbosio@iu.edu, cpmann@iu.edu
dgiffor@iu.edu, jnimmala@iu.edu
jkittur@iu.edu, zhangsuo@iu.edu
hdjour@iu.edu, zs15@iu.edu
ssdatla@iu.edu, zz67@iu.edu
tborder@iu.edu, cjrodenb@iu.edu
dh43@iu.edu, ew38@iu.edu
mgoenka@iu.edu, antonegr@iu.edu
agahlau@iu.edu, jawegman@iu.edu
rkoranne@iu.edu, aidpride@iu.edu

bronburk@iu.edu, pmiesch@iu.edu
boydmad@iu.edu, shamonei@iu.edu
ckbarts@iu.edu, aralover@iu.edu
alehe@iu.edu, cmellgre@iu.edu
bdgonzal@iu.edu, mkreddy@iu.edu
timolaws@iu.edu, ksabloak@iu.edu
salvchri@iu.edu, irodeman@iu.edu
ebederma@iu.edu, kokulski@iu.edu
addfishe@iu.edu, isundara@iu.edu
dayinla@iu.edu, auskirvi@iu.edu
nimjain@iu.edu, ibshaw@iu.edu
kadlakha@iu.edu, neenpate@iu.edu
sdondeti@iu.edu, aramanu@iu.edu
mugabr@iu.edu, taylbryo@iu.edu
talfares@iu.edu, antzamor@iu.edu
jbriar@iu.edu, kw125@iu.edu
ljhorman@iu.edu, whitedyl@iu.edu
abrking@iu.edu, lvumpa@iu.edu
pchitwoo@iu.edu, vpottete@iu.edu
garmenda@iu.edu, nshnoble@iu.edu
edeitchl@iu.edu, cnshimi@iu.edu
ahmealsh@iu.edu, ssp4@iu.edu
heissler@iu.edu, tkthang@iu.edu
concraft@iu.edu, thebmill@iu.edu
joglickm@iu.edu, smitaa@iu.edu
mabana@iu.edu, msharsh@iu.edu
pikhatri@iu.edu, sandro@iu.edu
shkami@iu.edu, adenyu@iu.edu
blstcoop@iu.edu, dommonte@iu.edu
ealfalah@iu.edu, sumuth@iu.edu
skyangel@iu.edu, nieywill@iu.edu
sergonza@iu.edu, camoverm@iu.edu
mbenites@iu.edu, jl367@iu.edu
bllfost@iu.edu, wl52@iu.edu
dkembert@iu.edu, adivaidy@iu.edu
keswar@iu.edu, slopezve@iu.edu
adguhan@iu.edu, audscha@iu.edu
vkabra@iu.edu, marsh1@iu.edu
chfurlow@iu.edu, tlichten@iu.edu
rbernfel@iu.edu, cmekat@iu.edu
bcarpine@iu.edu, ndmyer@iu.edu

ajkoutsa@iu.edu, swaschow@iu.edu
ajafowl@iu.edu, anupand@iu.edu
keswa@iu.edu, mzuhair@iu.edu
blbinder@iu.edu, ansjmeht@iu.edu
sabyrap@iu.edu, seuryu@iu.edu
adj18@iu.edu, everma@iu.edu
jejea@iu.edu, dyllopez@iu.edu
mdevara@iu.edu, dmonroyh@iu.edu
nikdevra@iu.edu, reshetty@iu.edu
nwconn@iu.edu, alzemlya@iu.edu
adekolaw@iu.edu, wschrama@iu.edu
nitiarun@iu.edu, djtheodo@iu.edu
mailic@iu.edu, amarynow@iu.edu
nbriere@iu.edu, braxluns@iu.edu
jamcham@iu.edu, jrwaren@iu.edu
shkani@iu.edu, soyedele@iu.edu
jjithesh@iu.edu, mloko@iu.edu
mgbekele@iu.edu, kenmoses@iu.edu
jigarnes@iu.edu, cardnorr@iu.edu
asgurram@iu.edu, shwanara@iu.edu
tbrose@iu.edu, rvenigal@iu.edu
eberens@iu.edu, vpotluri@iu.edu
zahaider@iu.edu, kevwong@iu.edu
augguy@iu.edu, maymat@iu.edu
lafly@iu.edu, ishyadav@iu.edu
ramgowda@iu.edu, adimaran@iu.edu
ttfender@iu.edu, lemarc@iu.edu
kyalane@iu.edu, tersacks@iu.edu
pranchan@iu.edu, mmalviya@iu.edu
gibsontn@iu.edu, froy@iu.edu
elehaged@iu.edu, gl25@iu.edu
ehchoo@iu.edu, sasubb@iu.edu
michdool@iu.edu, aishravi@iu.edu
langarbe@iu.edu, jvmenach@iu.edu
mthinman@iu.edu, mszach@iu.edu
btcunnin@iu.edu, prschulz@iu.edu
coubenne@iu.edu, sholove@iu.edu
mfbah@iu.edu, smitfi@iu.edu
bboyapat@iu.edu, jhl14@iu.edu
adagar@iu.edu, mahonm@iu.edu
herncr@iu.edu, cs155@iu.edu

grhamlin@iu.edu, hshyama@iu.edu
moazimi@iu.edu, bawsung@iu.edu
gkreutes@iu.edu, rpolu@iu.edu
adhimat@iu.edu, anrajput@iu.edu
rdotsu@iu.edu, prnemani@iu.edu
dbarbari@iu.edu, jakpatel@iu.edu
ojallow@iu.edu, conaitis@iu.edu
zalsamri@iu.edu, ozopich@iu.edu
skalluru@iu.edu, youngct@iu.edu
cammlanc@iu.edu, lisalaza@iu.edu
tycarmo@iu.edu, ijrussel@iu.edu
banksda@iu.edu, andyni@iu.edu
ygajjar@iu.edu, tunnitha@iu.edu
jahjacks@iu.edu, mmaplet@iu.edu
ndodoh@iu.edu, muksingh@iu.edu
jaccryst@iu.edu, klohiya@iu.edu
nifredri@iu.edu, drimawi@iu.edu
okirchen@iu.edu, jasreese@iu.edu
nantinao@iu.edu, isapittm@iu.edu
jackchap@iu.edu, sahasrir@iu.edu
armstan@iu.edu, matyun@iu.edu
cgcheane@iu.edu, joevu@iu.edu
dakomi@iu.edu, dajshaw@iu.edu
am280@iu.edu, rosanghv@iu.edu
owencape@iu.edu, cmolaski@iu.edu
efoltyn@iu.edu, dematt@iu.edu
reclay@iu.edu, anryoun@iu.edu
xmdrew@iu.edu, vpremku@iu.edu
rkoston@iu.edu, jakepars@iu.edu
ghargro@iu.edu, vnathany@iu.edu
skapure@iu.edu, rasahu@iu.edu
dajuhnke@iu.edu, tanilitt@iu.edu
benchart@iu.edu, bogawa@iu.edu
kevgross@iu.edu, athason@iu.edu
bgelard@iu.edu, ainlukas@iu.edu
tbashee@iu.edu, rvarrier@iu.edu
aabdulw@iu.edu, pjminne@iu.edu
srkagema@iu.edu, vivitran@iu.edu
rgyasini@iu.edu, jjrinald@iu.edu
mkulekci@iu.edu, clushell@iu.edu
yc127@iu.edu, aamuse@iu.edu

eabilius@iu.edu, dmw6@iu.edu
xbchrist@iu.edu, atsarver@iu.edu
cjo3@iu.edu, mmmatias@iu.edu
rycostin@iu.edu, marcmarq@iu.edu
etferba@iu.edu, westje@iu.edu
ldeveau@iu.edu, jmsequei@iu.edu
btbasing@iu.edu, roberjuv@iu.edu
lddial@iu.edu, ncw1@iu.edu
jergalan@iu.edu, kr15@iu.edu
ag38@iu.edu, jskains@iu.edu
bedani@iu.edu, rnalluri@iu.edu
natkwon@iu.edu, isvoor@iu.edu
byersjac@iu.edu, cjlomax@iu.edu
bthenson@iu.edu, stewrach@iu.edu