MAT 1630 Final Exam

Colin Gibbons-Fly, City Tech

2023-12-20

```
# import required libraries
import numpy as np
import matplotlib.pyplot as plt
```

Problem 1 (5 points)

Compute the sum:

$$\sum_{i=1}^{100} \frac{1}{i^2}$$

Hint: You can use either a for loop or a list comprehension.

Problem 1 Solution

```
#Using list comprehension to gather a list of i as 1 / i**2, ranging to 100
sequence = [i**(-2) for i in range(1,101)]

#Taking the sum of the list
sum(sequence)
```

```
1.6349839001848923
```

Problem 2 (10 points)

Given a list `a`, the `max()` function in Python's standard library computes the largest element in `a`: `max(a)`. Write your own max function, and call it `my_max(a)`. Apply your function `my_max()` to the list `a=[4,15,3,4,11,6,19,12,9,18]` and print the result.

Hint: Initialize a variable `max_elem` by the first element in the list, then iterate through all the remaining elements in the list (`a[1:]`), compare each element to `max_elem`, and if it's greater than the current value in `max_elem`, then set `max_elem` equal to that element.

Problem 2 Solution

```
#Defining max element function
def my_max(a):
  max_elem = a[0]
  for i in a:
    if i > max_elem:
      max_elem = i
  return max_elem

a=[4,15,3,4,11,6,19,12,9,18]

my_max(a)
```

```
19
```

Problem 3 (22 points)

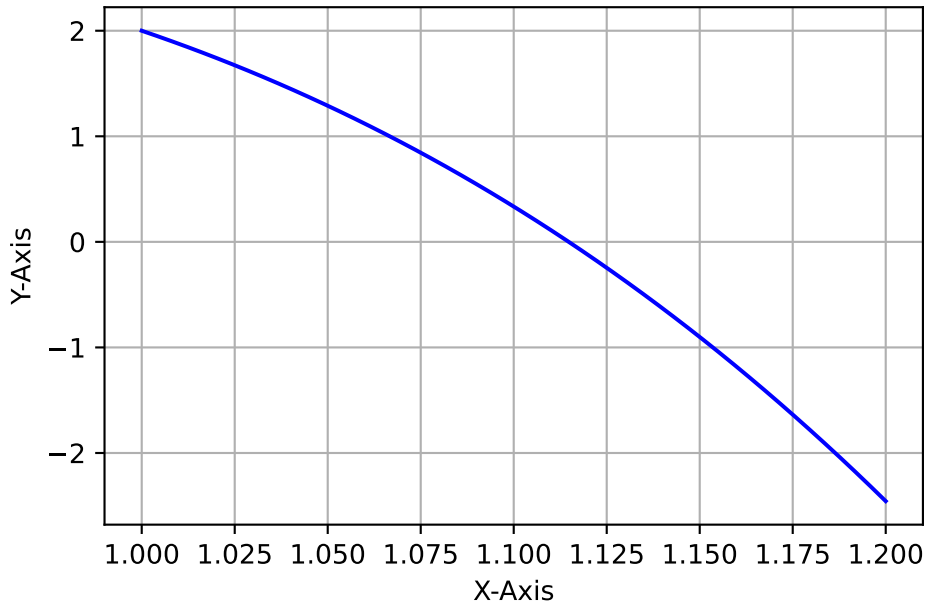
Consider the polynomial $p(x) = 16x^3 - 15x^4 + 1$.

a. (10 points) Plot the graph of $p(x)$ over the interval $[1, 1.2]$.
b. (10 points) Use the **bisection method** to find a root (to the nearest 0.001) of the polynomial $p(x)$ in $[1, 1.2]$.
c. (2 points) Evaluate $p(x)$ at the root you found in part 2. Is it close to zero?

Problem 3 Solution

```
plt.clf()

#Plot the polynomial
x = np.linspace(1,1.2)
y = (16*x**3) - (15*x**4) + 1
plt.plot(x,y,color='blue')
plt.grid()
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show()
```




```
#Implement the bisection method
def bisection(a,b,f,err):
  c=(b+a)/2
  while c-a > err:
    if f(c) == 0:
      return c
    elif f(a)*f(c) > 0:
      a=c
```

```
    else:
      b=c
    c=(b+a)/2
  return c

#Define polynomal function
def f(x):
  return (16*x**3) - (15*x**4) + 1

#Call bisection method
answer = bisection(a=0,b=3,f=f,err=0.001)
answer
```

```
1.115478515625
```

```
#Plugging in our answer to the original equation to see if it is close to 0
evaluation = (16*answer**3) - (15*answer**4) + 1
evaluation
```

```
-0.01624903932059496
```

Problem 4 (23 points)

Suppose you invested \$2000 into a savings account with a 5% annual interest rate compounded monthly.

- (a) (10 points) How much money will be in your account after 10 years, if you make monthly contributions of \$30?
- (b) (13 points) What is the minimum monthly contribution `c`, within 1c, needed to ensure that your savings account has at least \$20,000 in it after 10 years? **Hint:** Write a Python function `P(c)` for the current balance, as a function of the monthly contribution `c`, and use it in a while loop to find `c` such as `P(c) >= 20000`.

You may use the formula below for the principal balance at time t months:

$$P(t) = P \cdot \left(1 + \frac{r}{n}\right)^t + \frac{cn}{r} \cdot \left(\left(1 + \frac{r}{n}\right)^t - 1\right)$$

Problem 4 Solution

```
P0 = 2000 #Initial Investment
rate = 5/100 #Interest rate
n = 12 #Compounding periods are monthly
t = 10 #Timeframe
N =n*t #Total compounding periods
c = 30
Pnew = P0*(1+(rate/n))**N+c #New monthly balance

#Show the new balance
Pnew
```

```
3324.01899538056
```

```
#Function to get the monthly payment nedded
def P(c):
  P=P0
  for k in range(1,N+1):
    P=P*(1+rate/n)+c
  return P

#Setting the while loop to run until it hits our goal of 20000
c = 1
while P(c) <20000:
  #Increment by 1 cent
  c+=0.01
print(c)
```

```
107.59000000001814
```

Problem 5 (10 points)

Four fair dice are rolled together. Use Monte Carlo simulations to estimate the probability of getting a dice sum that is divisible by 4.

Problem 5 Solution

```
def rolls():
  #Get the sum of 4 random die
  sum =  np.random.randint(1,7,size=4).sum()
  #Use % to see if divisible by 4
  if sum % 4 == 0:
    #Return 1 for the MC simulation
    return 1
  else:
    return 0

#Run and print the MC simulation
result = [rolls() for i in range(10000)]
np.mean(result)
```

```
0.2446
```

Problem 6 (15 points)

Use Monte Carlo simulations to estimate the area enclosed by the closed curve $x^4+y^4 = 1$. The idea is to simulate random (x, y) points inside a square of side 2 units centered at $(0, 0)$.

a. (2 points) What is the range of the x-and-y coordinates?
b. (4 points) Simulate 10,000 random x and y coordinates inside the square, by using `np.random.rand(N)` and scaling it appropriately to fit the ranges for the coordinates.
c. (5 points) Compute the fraction of points that lie inside the closed curve, out of the total number of simulated random points.
d. (4 points) Use the fraction above to estimate the area enclosed by the closed curve.

Problem 6 Solution

```
points=10000 #Generate 1000 points
inside=0 #Initiate variable for points that land inside the desired area

#For loop that runs for the amount of desired points, generating random x, y variabl
for i in range(points):
  x=np.random.rand()*2
  y=np.random.rand()*2
  #If the random points land inside the area below, increase the 'inside' tracker by
  if y <= (x**4) - (y**4):
    inside+=1

#Divide the amount of points that landed inside the area by the total number of poir
inside/points
```

```
0.3916
```

Problem 7 (15 points)

The population of country X is divided into two groups: city-folk and country-folk. Currently, there are 10,000,000 country-folks, that is, people living in the country, and 5,000,000 city-folk, that is, people living in cities. Suppose each year 75% of the people living in the country remain there, 20% move to cities and 5% die. Suppose each year 90% of the people living in cities remain in cities, 5% move to the country and 5% die. Suppose each year 20,000 people immigrate into country X and all move into the country side. Build a model for this situation and describe the population for country X after 10 years. Express your model with two recursion equations and plot the time evolution of the two populations.

Problem 7 Solution

```
#Set the parameter sgiven in the question
t = 10
city_folk = np.empty(t)
country_folk = np.empty(t)
city_folk[0] =5000000
```

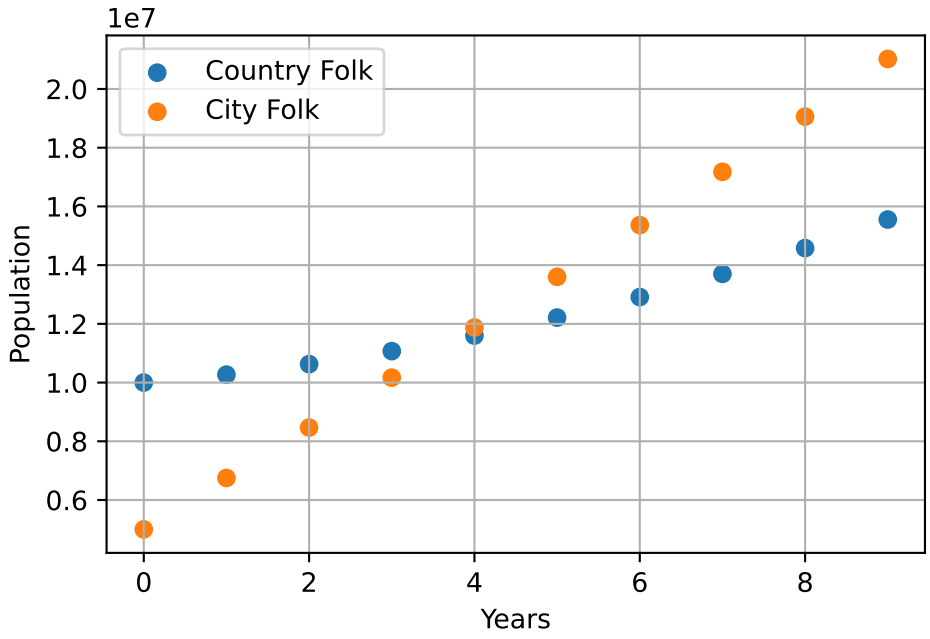
```
country_folk[0] = 10000000

#Use for loop to model out what will happen over time to each population
for i in range(t-1):
  country_folk[i + 1] = 0.75*country_folk[i] + 0.25*country_folk[i] + 0.05*city_folk
  city_folk[i + 1] = 0.2*country_folk[i] + 0.9*city_folk[i] + 0.1*city_folk[i] - 0.0

#Get the final sum of how many people will be in the city and country after 10 years
sum(country_folk)
sum(city_folk)
```

```
128481654.66439337
```

```
#Show the plot of how many people will be in the city and country
plt.clf()
plt.scatter(range(len(country_folk)),country_folk,label='Country Folk')
plt.scatter(range(len(city_folk)),city_folk,label='City Folk')
plt.legend()
plt.grid()
plt.ylabel('Population')
plt.xlabel('Years')
plt.show()
```



Bonus Problems for Extra Credit

Bonus Problem 1 (5 points)

You flip a fair coin repeatedly. Using simulation, estimate the average number of coin tosses to get 4 consecutive heads.

Bonus Problem 1 Solution

```
#Create function
def consecutive():
  #intiate variables for number of tosses and times it lands on heads
  tosses=0
  heads=0
  while heads < 4: #Flip coins until 4 heads in a row
    outcome = np.random.choice([1,0])
    tosses += 1 #Add 1 to your tosses tracker for every flip
    if outcome == 1:
      heads += 1 #If heads, add 1 to tracker
```

```
    else:
      heads = 0
  return tosses

#Return MC simulation
sample = [consecutive() for i in range(10000)]
np.mean(sample)
```

```
29.9576
```

Bonus Problem 2 (10 points)

A country has 270 missiles stored in different locations. The US military is considering an attack aimed at destroying all missiles. However, for the attack to succeed, every one of the missiles must be destroyed. Assume that each attacking warhead has a probability of 0.75 of hitting its target. Use Monte Carlo simulations to estimate the average number of warheads needed to destroy all 270 missiles.

Bonus Problem 2 Solution