MFE Math Bootcamp: Harshin Pathak

hapath1224@gmail.com

Question 1: Gradient Descent

a) Write a function that implements gradient descent algorithm in python. Set arguements for 5 variables

In [1]:
```
# Initialize Variables
h = 10**-2
nu = 10**-3
alpha = 10**-6
beta = 10**6
N = 100000

# Initialize x0 (start point)
x0 = 0

# Create function f
f = lambda x: (3*x**4)-(16*x**3)+(6*x**2)+72*x+5

# Define Gradient Descent Function with input parameters
def grad_desc(f, x0, h, nu, alpha, beta, N):
    def g(x0, h):
        g_x0 = (f(x0 + h) - f(x0 - h)) / (2 * h)
        return g_x0

    for i in range(N):
        g_x0 = g(x0, h)
        x = x0 - (nu * g_x0)
        if abs(x - x0) < alpha:
            return x
        elif abs(x - x0) > beta:
            return None
        else:
            x0 = x

    return x0

result = grad_desc(f, x0, h, nu, alpha, beta, N)
print(result)
```

```
-0.999975437086363
```

b) Pick x0 values and use function from a) to find local minima of f

In [3]:
```
# Pick 4 x values to test: -1,0,1,2
x_vals = [-1,0,1]
rx = []
ry = []
```

```
# Run the x values to find the corresponding local minima
for i in range(len(x_vals)):
    x0 = x_vals[i]
    x_lm = grad_desc(f, x0, h, nu, alpha, beta, N)
    rx.append(x_lm)
    ry.append(f(x_lm))

print('The local minima corresponding to x values of [-1,0,1] are', ry,)
```

```
The local minima corresponding to x values of [-1,0,1] are [-41.999999986210184, -4
1.999999956560174, -41.99999995404039]
```

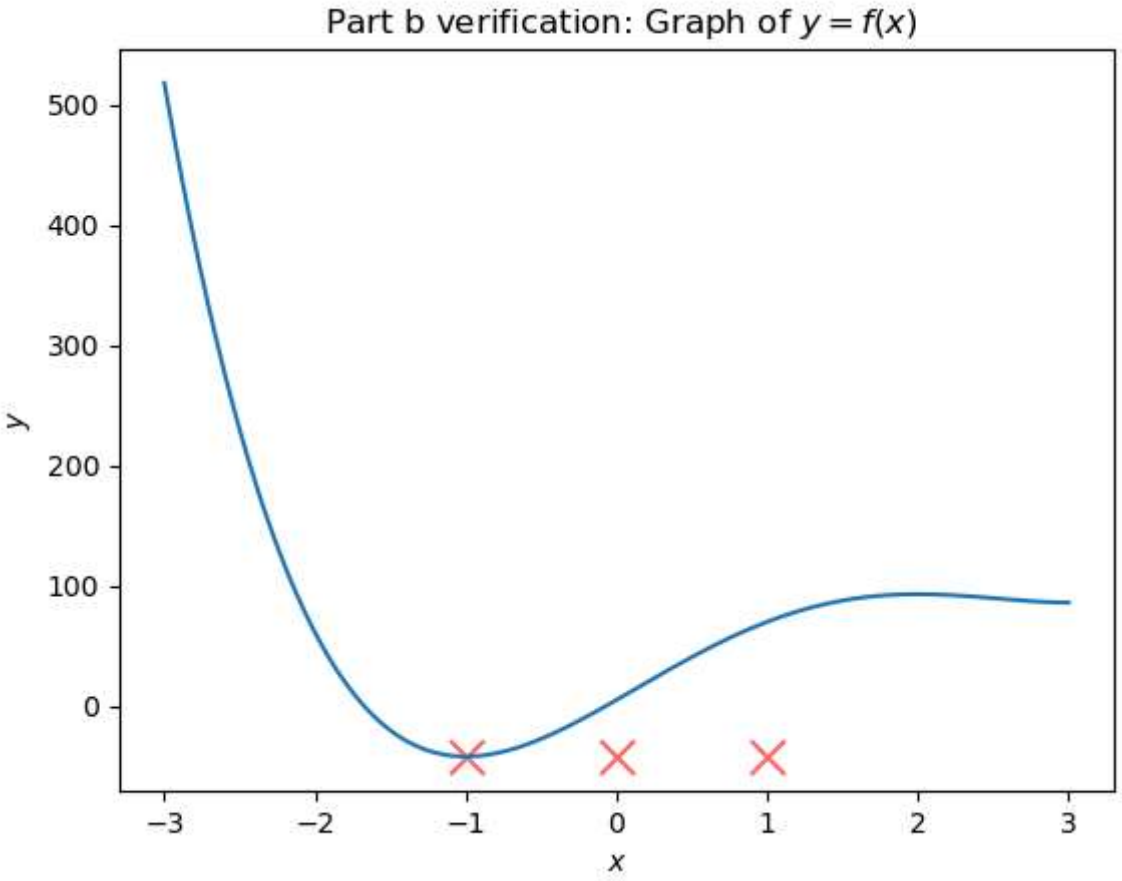
c) Repeating part b with -f (instead of f) to find local maxima

In [5]:
```
# Pick 4 x values to test: -1,0,1,2
x_vals = [-1,0,1]
rx_m = []
ry_m = []

# Create the new f
f2 = lambda x: -3*x**4 + 16*x**3 - 6*x**2 - 72*x - 5

def grad_desc2(f, x0, h, nu, alpha, beta, N):
    def g(x0, h):
        g_x0 = (f(x0 + h) - f(x0 - h)) / (2 * h)
        return g_x0

    for i in range(N):
        g_x0 = g(x0, h)
        x = x0 + (nu * g_x0) #add to move in gradient direction
        if abs(x - x0) < alpha:
            return x
        elif abs(x - x0) > beta:
            return None
        else:
            x0 = x

    return x0

# Run the x values to find the corresponding local maxima
for i in range(len(x_vals)):
    x0 = x_vals[i]
    x_lmax = grad_desc2(f2, x0, h, nu, alpha, beta, N)
    if x_lmax is not None:
        rx_m.append(x_lmax)
        ry_m.append(f2(x_lmax))

print('The local maxima corresponding to x values of [-1,0,1] are', ry_m,)
```

```
The local maxima corresponding to x values of [-1,0,1] are [41.999999986210184, 41.9
99999956560174, 41.99999995404039]
```

d) Verify Work in Parts b and c

In [7]:
```
#Part B Verification
import matplotlib.pyplot as plt
import numpy as np

#domain of our function f(x)
x = np.linspace(-3,3,2000)
y = f(x)

plt.plot(x, y)
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r' Part b verification: Graph of $y = f(x)$')

#plot the random x values from part b and their corresponding local minima
plt.scatter(x_vals,ry,color='red', s=150, marker='x', alpha=0.6)
plt.show()
```

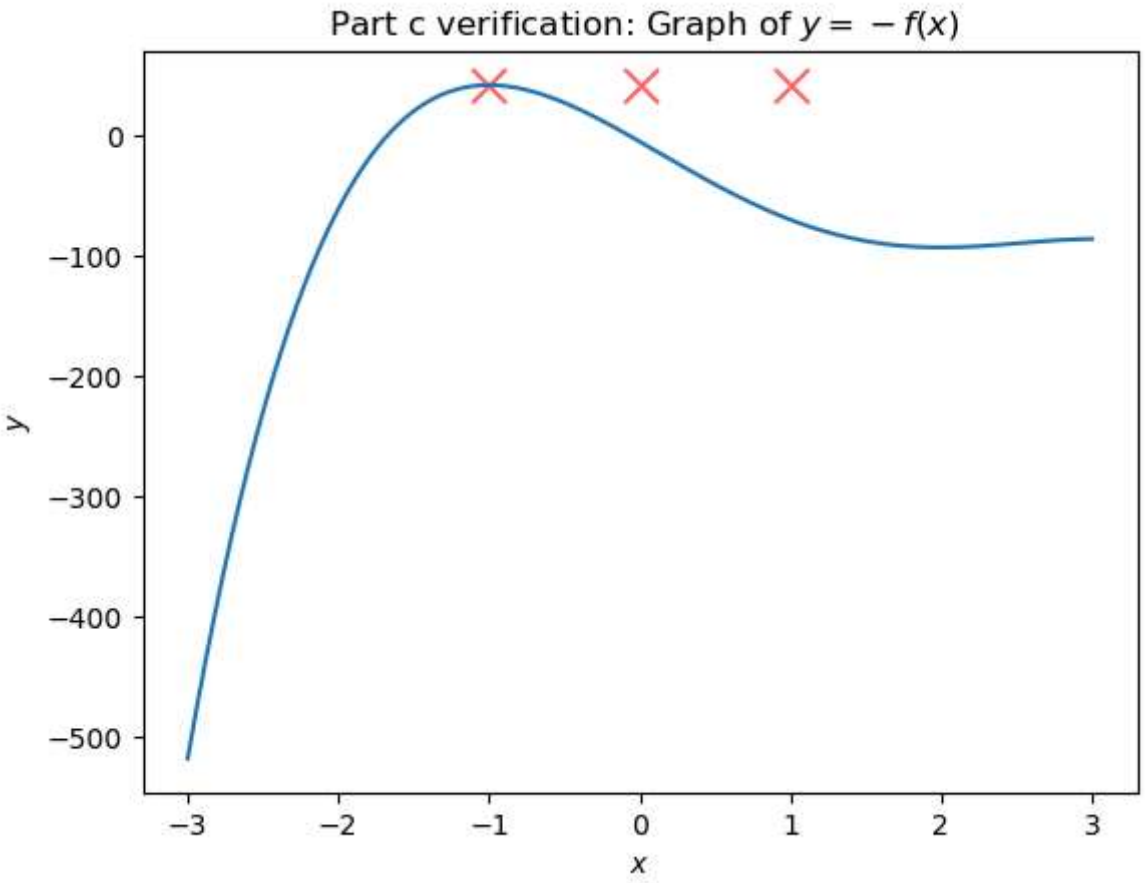


The x markers show the 3 initial x values to start gradient descent algorithm. The x marker's y values are all -42: corresponding to the closest local minima value at x= -1.

In other words, the plot above corresponds to HW question 1b) where 3 random gradient descent inititon points of x =-1,1,0 correspond to the same local mimina at x = -1 which is -42

In [9]:
```
#Part C Verification

#domain of our function f(x)
x = np.linspace(-3,3,2000)
y = f2(x)
```

```
plt.plot(x, y)
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
plt.title(r' Part c verification: Graph of $y = -f(x)$')

#plot the random x values from part b and their corresponding local minima
plt.scatter(x_vals,ry_m,color='red', s=150, marker='x', alpha=0.6)
plt.show()
```



The x markers show the 3 initial x values to start the "reverse" gradient descent algorithm. The x marker's y values are all 42: corresponding to the closest local maxima value at x= -1.

In other words, the plot above corresponds to HW question 1c) where 3 random gradient descent initition points of x =-1,1,0 correspond to the same local maxima at x = -1 which is 42

Question 2: area under curve of y = e^-sqrt(x) from x = 1, x = 100

a) Approximate Area using Riemann Sum with defined P

In [73]:
```
import numpy as np


def RS(f,s,e,n):
    P1 = np.linspace(s,e,n+1)
    dx1 = np.diff(P1)

    A_LS = sum(dx1*f(P1[0:n]))

    T=[0.5*P1[i]+0.5*P1[i+1] for i in range(0, n)]
```

```
    A_M= sum([f(T[i])*dx1[i] for i in range(0, n)])


    A_R = sum(dx1*f(P1[1:n+1]))
    #A_LS2 = sum(dx2*f(P2[0:n2]))
    return A_LS, A_M, A_R, dx1,P1, T


f = lambda x: np.exp(-np.sqrt(x))

output = RS(f,1,100,100)

A_LS, A_M, A_R, dx1, P1, T = output

print('The LRS is', A_LS,'The MRS is', A_M,'The RRS is', A_R,)
```

```
The LRS is 1.667254920627027 The MRS is 1.463321336207172 The RRS is 1.3030992197977
636
```

In [15]:
```
import pandas as pd

n_vals =[10, 25, 50, 75,100]
results = pd.DataFrame(index=n_vals, columns=['left', 'mid', 'right'])
f = lambda x: np.exp(-np.sqrt(x))


for n in n_vals:
    output = RS(f,1,100,n)
    A_LS, A_M, A_R, dx1, P1 = output
    results.loc[n,'left'] = A_LS
    results.loc[n,'mid'] = A_M
    results.loc[n,'right'] = A_R

results
```

Out[15]:

	left	mid	right
10	4.182437	1.159157	0.54088
25	2.395829	1.384797	0.939206
50	1.890313	1.444197	1.162002
75	1.738941	1.458065	1.2534
100	1.667255	1.463321	1.303099

b) Use a new partition method

In [17]:
```
def calculate_powers(start, end, n):
    results = []
    for i in range(n + 1):  # Iterate from 0 to n (inclusive)
        value = end ** (i / n)
        results.append(value)
```

```
    return np.array(results)  # Convert to a NumPy array for convenience

# Example usage:
start = 1  # This variable is not actually used in the loop
end = 10
n = 10

result_array = calculate_powers(start, end, n)
print(result_array)
```

```
[ 1.          1.25892541  1.58489319  1.99526231  2.51188643  3.16227766
  3.98107171  5.01187234  6.30957344  7.94328235 10.        ]
```

In [111…

```
#Make an edit to the riemann sum function to account for new partition method

def RS2(f,s2,e2,n2):
    P2 = calculate_powers(s2, e2, n2)
    dx2 = np.diff(P2)

    A_LS2 = sum(dx2*f(P2[0:n2]))

    T2=[0.5*P2[i]+0.5*P2[i+1] for i in range(0, n2)]
    A_M2= sum([f(T2[i])*dx2[i] for i in range(0, n2)])


    A_R2 = sum(dx2*f(P2[1:n2+1]))

    return A_LS2, A_M2, A_R2, dx2, P2, T2


f = lambda x: np.exp(-np.sqrt(x))

n_vals2 =[10, 25, 50, 75,100]
results2 = pd.DataFrame(index=n_vals2, columns=['left', 'mid', 'right'])
f = lambda x: np.exp(-np.sqrt(x))


for n2 in n_vals2:
    output2 = RS2(f,1,100,n2)
    A_LS2, A_M2, A_R2, dx2, P2, T2 = output2
    results2.loc[n2,'left'] = A_LS2
    results2.loc[n2,'mid'] = A_M2
    results2.loc[n2,'right'] = A_R2

results2
```

Out[111…

	left	mid	right
10	1.969399	1.437359	1.108519
25	1.65079	1.465123	1.311941
50	1.557744	1.469167	1.38871
75	1.528043	1.469918	1.415401
100	1.51343	1.470181	1.428961

The table above is the Riemann Sums without a uniform partition

c) Use analytic technique to find exact area

In [21]:
```
# Integrating this integral analytically we get the solution

e= 2.71828
sol = (4/e)-(22/(e**10))
sol
```

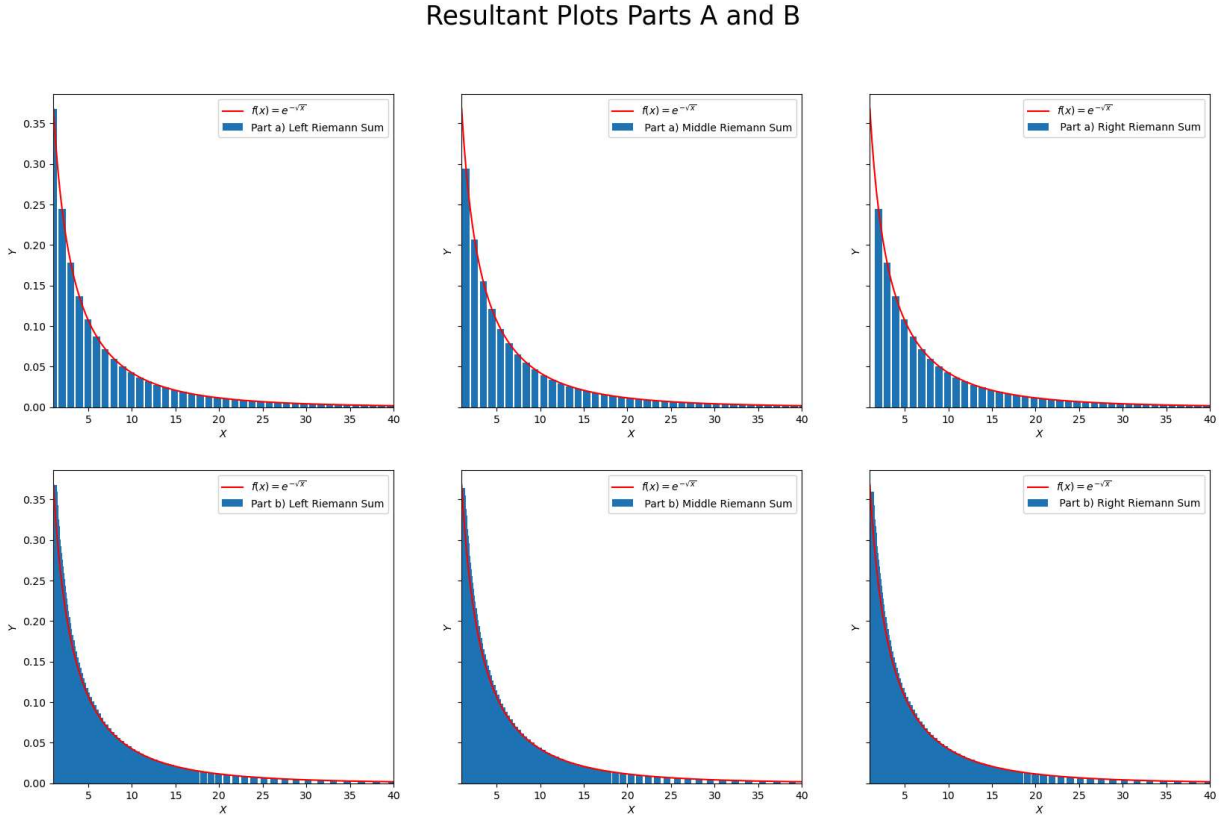
Out[21]: 1.4705199493332841

d) Use the Matplotlib and subplots to graph results from part a and b

In [194…
```
import numpy as np
import matplotlib.pyplot as plt


#Subplots Part A
fig, ax = plt.subplots(2, 3, sharey = True, figsize = (20, 12))
n=100
#For n=100, Part a Left Riemann Sum
ax[0, 0].bar((P1[0:n]), f(P1[0:n]), label='Part a) Left Riemann Sum')
ax[0, 0].plot(x, y, label=r'$f(x) = e^{-\sqrt{x}}$', color='red')
ax[0,0].set_xlim([1, 40])
ax[0, 0].set_xlabel(r'$X$')
ax[0, 0].set_ylabel(r'$Y$')
ax[0, 0].legend( loc="upper right")

#For n=100, Part a Middle Riemann Sum
ax[0, 1].bar(T[0:n], f(T[0:n]), label=' Part a) Middle Riemann Sum')
ax[0, 1].plot(x, y, label=r'$f(x) = e^{-\sqrt{x}}$', color='red')
ax[0,1].set_xlim([1, 40])
ax[0, 1].set_xlabel(r'$X$')
ax[0, 1].set_ylabel(r'$Y$')
ax[0, 1].legend( loc="upper right")

#For n=100, Part a Right Riemann Sum
ax[0, 2].bar((P1[1:n+1]), f(P1[1:n+1]), label=' Part a) Right Riemann Sum')
ax[0, 2].plot(x, y, label=r'$f(x) = e^{-\sqrt{x}}$', color='red')
ax[0,2].set_xlim([1, 40])
ax[0, 2].set_xlabel(r'$X$')
ax[0, 2].set_ylabel(r'$Y$')
```

```
ax[0, 2].legend( loc="upper right")


#Subplots Part B
#For n=100, Part b Left Riemann Sum
ax[1, 0].bar((P2[0:n2]), f(P2[0:n2]), label='Part b) Left Riemann Sum')
ax[1, 0].plot(x, y, label=r'$f(x) = e^{-\sqrt{x}}$', color='red')
ax[1,0].set_xlim([1, 40])
ax[1, 0].set_xlabel(r'$X$')
ax[1, 0].set_ylabel(r'$Y$')
ax[1, 0].legend( loc="upper right")

#For n=100, Part b Middle Riemann Sum
ax[1, 1].bar(T2[0:n2], f(T2[0:n2]), label=' Part b) Middle Riemann Sum')
ax[1, 1].plot(x, y, label=r'$f(x) = e^{-\sqrt{x}}$', color='red')
ax[1,1].set_xlim([1, 40])
ax[1, 1].set_xlabel(r'$X$')
ax[1, 1].set_ylabel(r'$Y$')
ax[1, 1].legend( loc="upper right")

#For n=100, Part b Right Riemann Sum
ax[1, 2].bar((P2[1:n2+1]), f(P2[1:n2+1]), label=' Part b) Right Riemann Sum')
ax[1, 2].plot(x, y, label=r'$f(x) = e^{-\sqrt{x}}$', color='red')
ax[1,2].set_xlim([1, 40])
ax[1, 2].set_xlabel(r'$X$')
ax[1, 2].set_ylabel(r'$Y$')
ax[1, 2].legend( loc="upper right")
plt.suptitle(r'Resultant Plots Parts A and B', size=25)
```

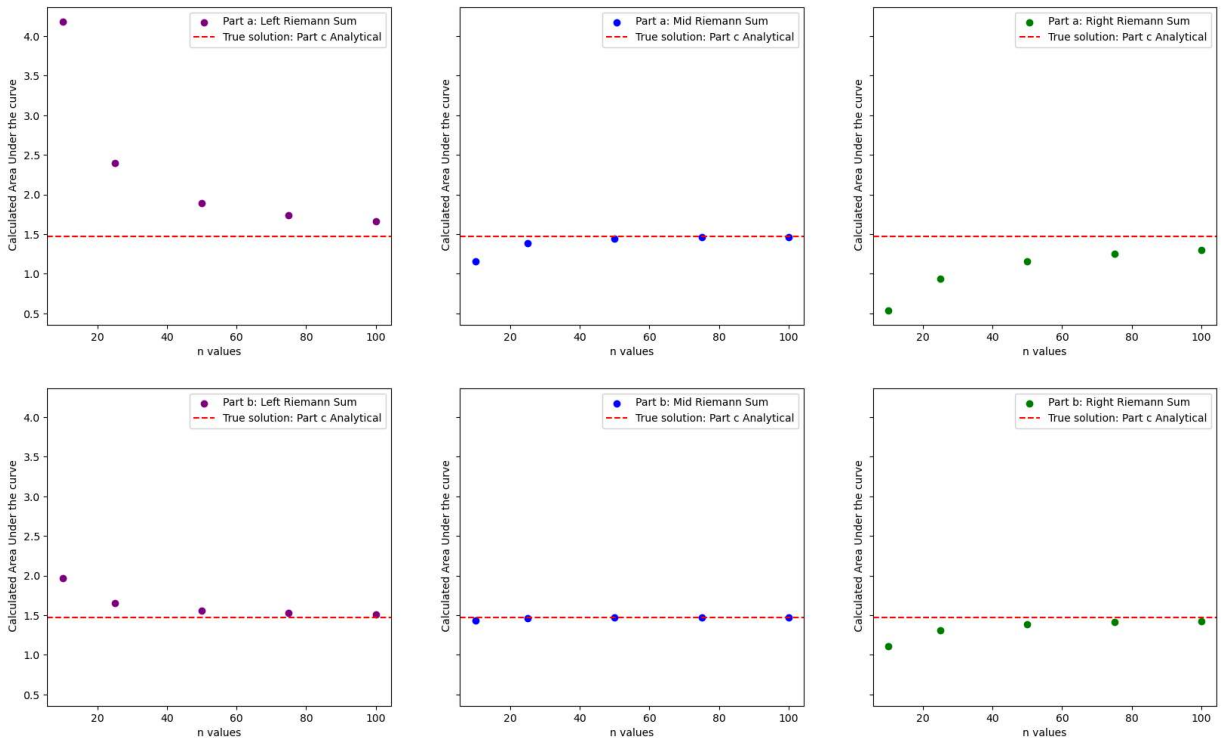
Out[194… Text(0.5, 0.98, 'Resultant Plots Parts A and B')



In [ ]:

In [197…
```
df=results
df2=results2



#Subplots Part A
fig, ax = plt.subplots(2, 3, sharey = True, figsize = (20, 12))

ax[0, 0].scatter(df.index, df["left"], color="purple", label="Part a: Left Riemann
ax[0, 0].axhline(y=sol, color = 'red', linestyle='--', label='True solution: Part c
ax[0, 0].set_xlabel(r'n values')
ax[0, 0].set_ylabel(r' Calculated Area Under the curve')
ax[0, 0].legend( loc="upper right")


ax[0, 1].scatter(df.index, df["mid"], color="blue", label="Part a: Mid Riemann Sum"
ax[0, 1].axhline(y=sol, color = 'red', linestyle='--', label='True solution: Part c
ax[0, 1].set_xlabel(r'n values')
ax[0, 1].set_ylabel(r' Calculated Area Under the curve')
ax[0, 1].legend( loc="upper right")


ax[0, 2].scatter(df.index, df["right"], color="green", label="Part a: Right Riemann
ax[0, 2].axhline(y=sol, color = 'red', linestyle='--', label='True solution: Part c
ax[0, 2].set_xlabel(r'n values')
ax[0, 2].set_ylabel(r' Calculated Area Under the curve')
ax[0, 2].legend( loc="upper right")


#Subplots Part B

ax[1, 0].scatter(df2.index, df2["left"], color="purple", label="Part b: Left Rieman
ax[1, 0].axhline(y=sol, color = 'red', linestyle='--', label='True solution: Part c
ax[1, 0].set_xlabel(r'n values')
ax[1, 0].set_ylabel(r' Calculated Area Under the curve')
ax[1, 0].legend( loc="upper right")


ax[1, 1].scatter(df2.index, df2["mid"], color="blue", label="Part b: Mid Riemann Su
ax[1, 1].axhline(y=sol, color = 'red', linestyle='--', label='True solution: Part c
ax[1, 1].set_xlabel(r'n values')
ax[1, 1].set_ylabel(r' Calculated Area Under the curve')
ax[1, 1].legend( loc="upper right")


ax[1, 2].scatter(df2.index, df2["right"], color="green", label="Part b: Right Riema
ax[1, 2].axhline(y=sol, color = 'red', linestyle='--', label='True solution: Part c
ax[1, 2].set_xlabel(r'n values')
ax[1, 2].set_ylabel(r' Calculated Area Under the curve')
ax[1, 2].legend( loc="upper right")

plt.suptitle(r'Numerical Values for Integral Visualized: Parts a, b and c', size=25
```

Out[197… Text(0.5, 0.98, 'Numerical Values for Integral Visualized: Parts a, b and c')



Question 3: Pricing Annuities

In [ ]:

In [ ]:

In [ ]:

Harshin Pahwa

MFE MATH Boutcamp HW #1

3) • perpetuities
- find price or prove that results would not converge
- Payment end of year r = 0.07

1) $10 payment

$$NPV = \frac{10}{1.07} + \frac{10}{(1.07)^2} + \frac{10}{(1.07)^3} \cdots \frac{10}{(1.07)^n}$$

⇒ geometric series

$$10 \sum_{k=1}^{\infty} \left(\frac{1}{1.07}\right)^k$$

* $\sum_{k=1}^{\infty} x^k = \frac{x}{1-x}$

⇒ $x = \left(\frac{1}{1.07}\right)$

$$10 \left(\frac{\frac{1}{1.07}}{1 - \frac{1}{1.07}} \right) \frac{1.07}{1.07}$$

$$\Rightarrow 10 \cdot \left[\frac{\frac{1.07}{1.07}}{1.07 - \frac{1.07}{1.07}} \right] \Rightarrow 10 \cdot \left[\frac{1}{1.07-1} \right]$$

$$\Rightarrow \frac{10}{0.07} \Rightarrow \boxed{NPV = \$142.86}$$

b) Payments of numerical value of the year

$$NPV = \frac{2023+1}{(1+0.07)} + \frac{2023+2}{(1+0.07)^2} + \frac{2023+3}{(1+0.07)^3} + \dots + \frac{2023+T}{(1+0.07)^T}$$

$$NPV = \sum_{k=1}^{\infty} \frac{2023+k}{(1.07)^k} \Rightarrow \underset{A)}{\sum_{k=1}^{\infty} \frac{2023}{(1.07)^k}} + \underset{B)}{\sum_{k=1}^{\infty} \frac{k}{(1.07)^k}}$$

A) $2023 \sum_{k=1}^{\infty} \left(\frac{1}{1.07}\right)^k \Rightarrow$ Geometric series $\Rightarrow 2023 \left(\frac{\frac{1}{1.07}}{1-\frac{1}{1.07}}\right) \cdot \frac{1.07}{1.07}$

$$\sum_{k=1}^{\infty} x^k = \frac{x}{1-x}$$

$$x = 1/1.07$$

$$\Rightarrow 2023 \cdot \left(\frac{1}{1.07-1}\right) \Rightarrow A) = 28{,}900$$

B) $\sum_{k=1}^{\infty} \frac{k}{(1.07)^k}$

$$NPV_T = NPV_A + NPV_B$$

$$28{,}900 + \sum_{k=1}^{\infty} \frac{k}{(1.07)^k}$$

(1) payment 1$ this year and +10% in 10% for next 4 years, each 5% after that

Soln $1, 1.10, 1.10(1.10)^2, 1.10(1.10)^3, (1.10)(1.10)^4, 1.46\cdot(1.05), 1.46\cdot(1.05)^2$

1 2 3 4 5 6 7

- $NPV_{1-5} = \frac{1}{1.07} + \frac{1.10}{(1.07)^2} + \frac{(1.10)^3}{(1.07)^3} + \frac{(1.10)^4}{(1.07)^4} + \frac{(1.10)^5}{(1.07)^5}$
- formula $\Rightarrow \frac{1-\left(\frac{1+r}{1+i}\right)^n}{i-r} \Rightarrow \frac{1-\left(\frac{1.10}{1.07}\right)^n}{-0.03} \Rightarrow \frac{1-\left(\frac{1.10}{1.07}\right)^5}{-0.03}$

$NPV_{1-5} = 4.94$

- $NPV_{6\to\infty} = \sum_{t=1}^{\infty} (1.46)\cdot(1.05)^t$

$= (1.46)\sum_{t=1}^{\infty}(1.05)^t \Rightarrow 1.46\cdot\left(\frac{1.05}{i-1.05}\right)$

use $\Rightarrow \frac{x}{1-x} = \sum_{k=1}^{\infty} x^k$

$\Rightarrow \frac{1-\left(\frac{1.05}{1.07}\right)^n}{0.02} \Rightarrow \frac{1.46}{0.02}\left(1-\left(\frac{1.05}{1.07}\right)^n\right)$

$\sum_{k=1}^{\infty}\left(\frac{1.05}{1.07}\right)^k = \frac{1.05}{1.07}\cdot\frac{1}{1-\frac{1.05}{1.07}}\cdot\frac{1.07}{1.05} \Rightarrow \frac{1}{1.05}$

Total $= NPV_{1-5} + NPS_{6\to\infty}$

$= 4.94 + \left(\frac{1-\left(\frac{1.05}{1.07}\right)^n}{0.02}\right)$

d) payments of k^k in kth year where 2024 = 1st

exp $\Rightarrow$ $\frac{1^1}{(1.07)^1}, \frac{2^2}{(1.07)^2}, \frac{3^3}{(1.07)^3} \cdots \frac{k^k}{(1.07)^k}$

$NPV = \sum_{k=1}^{\infty} \frac{k^k}{(1.07)^k}$ * looks to diverge

ratio test: $\lim_{k\to\infty} \left| \frac{\left(\frac{k+1}{1.07}\right)^{k+1}}{\left(\frac{k}{1.07}\right)^k} \right|$

$\Rightarrow \lim_{k\to\infty} \left| \frac{(k+1)^{k+1}}{1.07^{k+1}} \cdot \frac{1.07^k}{k^k} \right|$

$\lim_{k\to\infty} \left| \frac{(k+1)^{k+1}}{1.07 \cdot k^k} \right|$

$\lim_{k\to\infty} \left| \frac{(k+1)(k+1)^k}{(1.07)k \cdot k^{k-1}} \right|$ * L'hopital

$\Rightarrow \lim_{k\to\infty} \frac{(k+1)^{k+1}}{(1.07)^k}$

$\Rightarrow$ * L'hopital

$\lim_{k\to\infty} \left| \frac{(k+1)\,k\,(k+1)^{k-1}(k+1)^k}{(1.07)k^{k-1} \cdot (1.07)k\,(k-1)k^{k-2}} \right|$

$\frac{(1)(1)(k-1)(k)}{k^{k-1}\,k^{k-2}}$

$\lim_{k\to\infty}$ seems to be $\to 1$

diverges