Assignment 1

Stochastic Methods

Justin Self

AERO 500 Aerospace Modeling and Simulation

May 2, 2025

Problem 1

a) Use the Monte Carlo Integration Method to estimate the value of π.

b) Submit your Python code and a plot of your results

c) Approximately how many iterations does your code take to converge to a solution?

d) How many trials did your simulation need to converge within 1% of the known value of π?

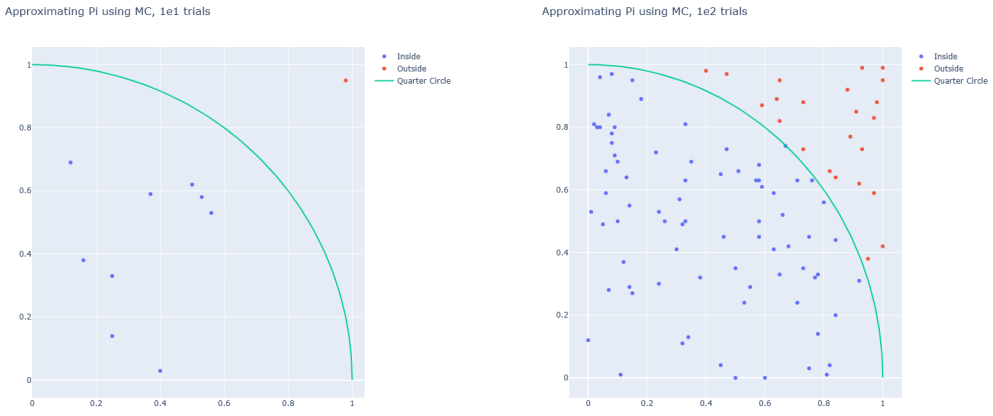
Solution. Four iteration count values were explored for plotting convergence: 10, 100, 1000, and 10000 counts. The solution converged withing 1% of the known value of π between 100 and 1000 iterations. The output solutions are given below and are shown in Figure 1a - Figure 1d. The full code is included in Section 1.

```
------ numTrials = 1e1 ------
----------------------------
sumIn is:
9
sumOut is:
1
>>>>> Approximation of pi <<<<<
Number of trials: 10
3.6
Percent off is: -14.591559026164646

------ numTrials = 1e2 ------
----------------------------
sumIn is:
77
sumOut is:
23
>>>>> Approximation of pi <<<<<
Number of trials: 100
3.08
Percent off is: 1.9605550553924616

------ numTrials = 1e3 ------
----------------------------
sumIn is:
779
sumOut is:
221
>>>>> Approximation of pi <<<<<
```

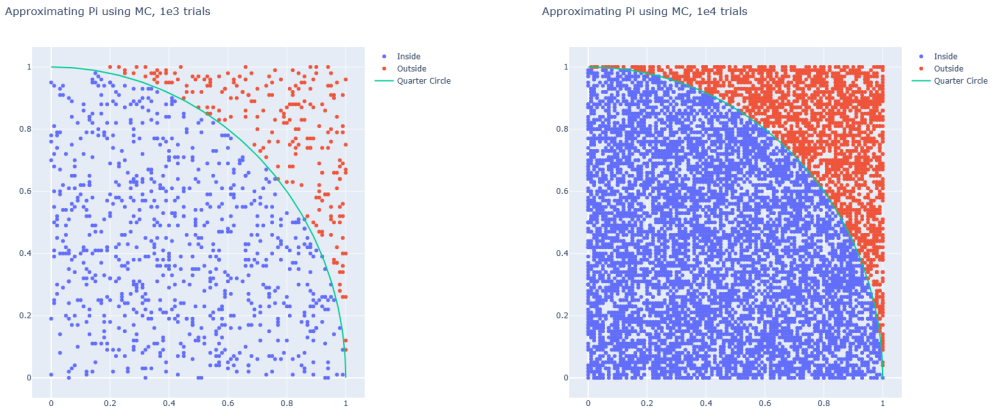
```
Number of trials: 1000
3.116
Percent off is: 0.8146394651308242

------ numTrials = 1e4 ------
----------------------------
sumIn is:
7834
sumOut is:
2166
>>>>> Approximation of pi <<<<<
Number of trials: 10000
3.1336
Percent off is: 0.25441406544734946
```

1 Full Code: Task 1

```python
# Justin Self
# AERO 500
# Spring 2025


# Task 1: Estimate pi using Monte Carlo Integration Method

# Draw an imaginary quarter circle in a box
# box height = length = 1
print(" ")
print("------ TASK 1 ------")
import random
import decimal
import math
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

numTrials =10**4 #                      <<<<<< user changes this only
R = 1 # radius of circle
sumIn = 0
sumOut = 0
xPoint = []
yPoint = []

xPointsIn = []
yPointsIn = []
xPointsOut = []
yPointsOut = []
```



(a) (b)



(c) (d)

Figure 1: Approximating π using Monte Carlo Simulation

```
31
32 for trial in range(numTrials):
33     xPoint.append(random.randint(0,100)/100)
34     yPoint.append(random.randint(0,100)/100)
35     z = xPoint[trial]**2 + yPoint[trial]**2
36     #print("------------Length is:")
37     #print(z)
38
39     if (z <= R):
40         #print("Inside")
41         xPointsIn.append(xPoint[trial])
42         yPointsIn.append(yPoint[trial])
43         sumIn += 1
44
45     else:
46         #print("Outside")
47         xPointsOut.append(xPoint[trial])
48         yPointsOut.append(yPoint[trial])
49         sumOut += 1
50
51 print("_____________________________")
52 print("sumIn is: ")
53 print(sumIn)
54 print("sumOut is: ")
55 print(sumOut)
56
57 print(">>>>> Approximation of pi <<<<<")
58 piApprox = 4.0* sumIn / numTrials
59 print("Number of trials: " + str(numTrials))
60 print(piApprox)
61
62 # Create figure
63 #fig = go.Figure()
64 fig = go.Figure(layout_title_text="Approximating Pi using MC, 1e4 trials")
65 #fig = go.Figure(layout=go.Layout(title=go.layout.Title(text="Number of iterations
       : " + str(numTrials) + "; approximation of pi is: " + str(piApprox))))
66 # Add trace for inside points
67 fig.add_trace(go.Scatter(x = xPointsIn,y = yPointsIn, mode="markers",name="Inside"
       ))
68 # Add trace for outside points
69 fig.add_trace(go.Scatter(x=xPointsOut,y=yPointsOut, mode="markers",name="Outside")
       )
70
71 #print("X points in:")
72 #print(xPointsIn)
73 #print("X points out:")
74 #print(xPointsOut)
75
76 # Add trace to plot the quarter circle
77 centerX,centerY = 0,0
78 theta = np.linspace(0,np.pi / 2, 100)
79 x =  centerX + R * np.cos(theta)
80 y = centerY + R * np.sin(theta)
81
```

```
# Quarter circle
fig.add_trace(go.Scatter(x=x,y=y, mode="lines",name="Quarter Circle"))

# Show plot
fig.show()

# Compute 1% convergence nTrials
onepercent = 0.01*math.pi
howclose = piApprox / math.pi
percentoff = 100-howclose*100
print("Percent off is: " + str(percentoff))
```

Algorithm 1: Task 1

Problem 2

a) Use the Monte Carlo Integration Method to integrate the function $f(x) = x^3 \sin(x)$, over the domain, $-2 \leq x \leq 3$.

b) Assuming $E = \int_a^b f(x)\, dx$, you can use the Mean Value Theorem to approximate the integral of $f(x)$ as $E_N = \frac{(b-a)}{N} \sum_{i=1}^{N} f(x_i)$

c) Submit your Python code and a plot of your results

d) Approximately how many iterations does your code take to converge to a solution?

Solution.

In this problem, we first estimate the function using the Mean Value Theorem, then we explore a geometric argument similar to Task 1. The approach here approximates the function $f(x) = x^3 \sin(x)$ for $x \in [-2, 3]$ using the Mean Value Theorem, with $f(x) \approx E_N = \frac{(b-a)}{N} \sum_{i=1}^{N} f(x_i)$, so the limiting factor was only the number of iterations in the Python loop. The solution converged to less than 1% (0.2%) of the known value of 15.66464 in less than 100 iterations, shown below.

Interestingly, after 1000 iterations, the solution fluctuates a bit (1.9%), then the error at 10000 iterations reduces to 1.4%. Further behavior of the error fluctuations were not studied. The *good enough* estimation was obtained near 100 iterations.

The plots shown in Figure 2 are provided to show how a geometric argument similar to Task 1 might be obtained. The area under the curve (AUTC) can be approximated (and thus the integral) by the following:

$$\text{AUTC} \approx \frac{\text{points under curve}}{\text{total points}} \cdot \text{ Total area}$$

Where the total area is simply $x \cdot y$ from the graph area of the plot. The geometric argument is better (but not significantly different) than the MVT for the $N = 1e + 04$ case, and does not improve for the $N = 1e + 05$ case.

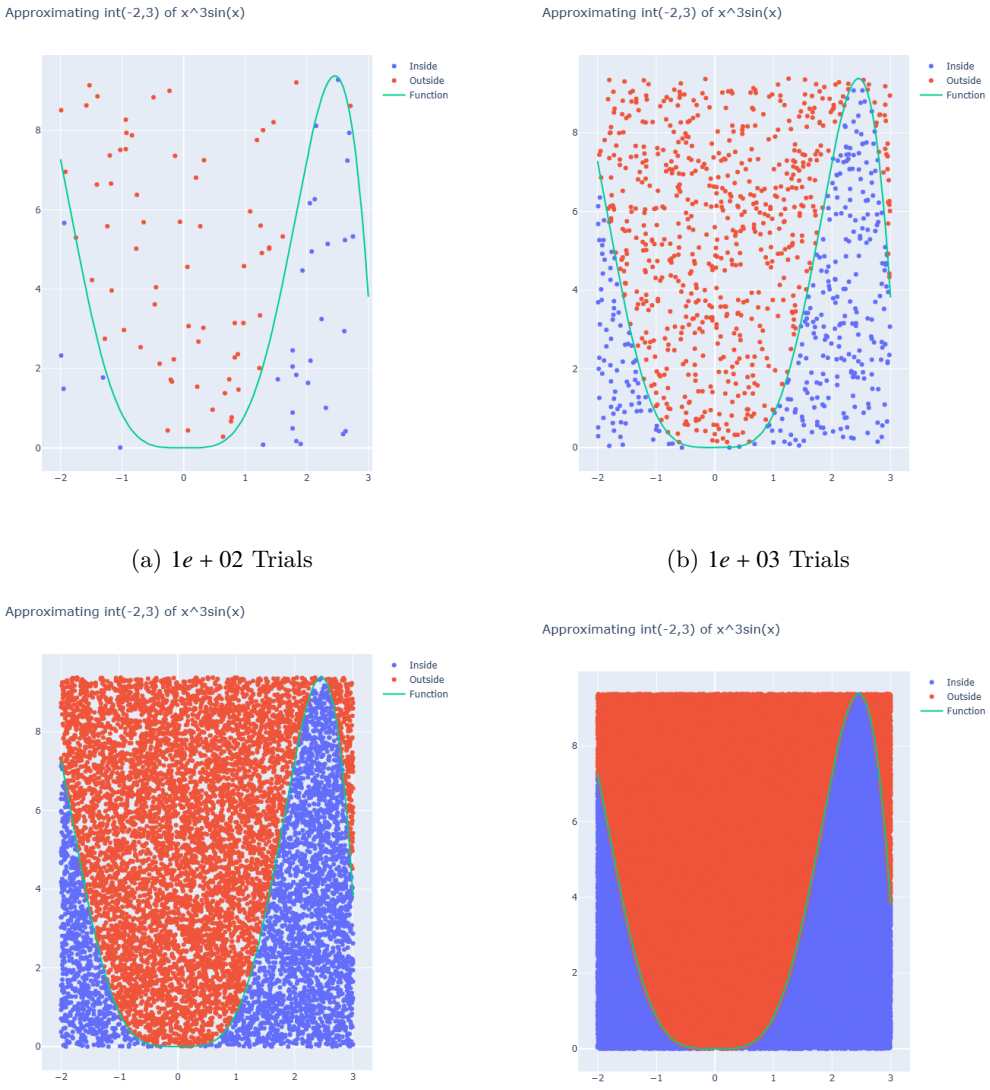
The full code is provided in Section 2.

```
------ N = 1e1 ------

----------------------------
sumIn is: 2
sumOut is: 8
Exact solution is: 15.66464
Approximation gives:
11.66235283616861
Geometric approximation
9.37

------ N = 1e2 ------
```

```
----------------------------
sumIn is: 40
sumOut is: 60
Exact solution is: 15.66464
Approximation gives:
17.782494919529398
Geometric approximation
18.74

------ N = 1e3 ------
----------------------------
Exact solution is: 15.66464
Approximation gives:
15.429241209101654
Geometric approximation
15.226249999999999


------ N = 1e4 ------
----------------------------
sumIn is: 3347
sumOut is: 6653
Exact solution is: 15.66464
Approximation gives:
15.805514417551322
Geometric approximation
15.680694999999998

------ N = 1e5 ------
----------------------------
sumIn is: 33574
sumOut is: 66426
Exact solution is: 15.66464
Approximation gives:
15.616718561237205
Geometric approximation
15.729418999999996
```

2 Full Code: Task 2

```
'''
Justin Self
AERO 500 (Aerospace Modeling and Simulation)
```



(a) $1e+02$ Trials

(b) $1e+03$ Trials

(c) $1e+04$ Trials

(d) $1e+05$ Trials

Figure 2: Approximating $f(x) = x^3 \sin(x)$, over the domain, $-2 \leq x \leq 3$ using Monte Carlo Simulation (geometric) and Mean Value Theorem (iterative) Methods

```
Spring 2025
Assignment 1: Stochastic Methods

Part 2: use MSC to approximate: int from -2 to 3 of x^3 sinx
'''
# import modules
import random
import math
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

# Task 2: Monte Carlo Integration Method estimations
print(" ")
print("------ TASK 2 ------")
a = -2
b = 3
N = 100000 # number of iterations <<<<<<<<<<<<< user changes this

xPoint = []
yPoint = []
xPointsIn = []
yPointsIn = []
xPointsOut = []
yPointsOut = []
sumIn = 0
sumOut = 0
summation = 0

for trial in range(N):
    xPoint.append(random.randint(-200,300)/100)
    yPoint.append(random.randint(0,937)/100)
    fx = xPoint[trial]**3 * math.sin(xPoint[trial])
    summation += fx
    #En = (b-a)/N * fx.append(fx[trial])

    if (yPoint[trial] <= fx):
        #print("Inside")
        xPointsIn.append(xPoint[trial])
        yPointsIn.append(yPoint[trial])
        sumIn += 1

    else:
        #print("Outside")
        xPointsOut.append(xPoint[trial])
        yPointsOut.append(yPoint[trial])
        sumOut += 1

print("_____________________________")
print("sumIn is: " + str(sumIn))
#print(sumIn)
print("sumOut is: " + str(sumOut))
#print(sumOut)
```

```
58
59 fig = go.Figure(layout_title_text="Approximating int(-2,3) of x^3sin(x)")
60
61 # Add trace for inside points
62 fig.add_trace(go.Scatter(x = xPointsIn,y = yPointsIn, mode="markers",name="Inside"
       ))
63 # Add trace for outside points
64 fig.add_trace(go.Scatter(x=xPointsOut,y=yPointsOut, mode="markers",name="Outside")
       )
65
66 # Add function to integrate
67 xVals = np.linspace(a,b,100)
68 for i in xVals:
69     yVals =  xVals**3 * np.sin(xVals)
70 fig.add_trace(go.Scatter(x = xVals, y = yVals, mode="lines",name="Function"))
71
72 # plot it
73 fig.show()
74
75 # Approximate the integral
76 exSol = 15.66464
77
78 EN = (b-a) / N * summation
79 approx = EN
80 print("Exact solution is: " + str(exSol))
81 print("Approximation gives: ")
82 print(approx)
83
84 print("Geometric approximation")
85 wholearea = 5 * 9.37 # whole area
86 approximateGeo = sumIn/N * wholearea #geo approx
87 print(approximateGeo)
```

Algorithm 2: Task 2

Problem 3

a) In this task, you are going to analytically and numerically propagate the error for the ideal gas law, $P = \rho R_{air} T$, with $R_{air} = 287.053 \frac{J}{kgK}$

b) Assume you are measuring temperature and pressure of a fixed volume. Initially everything is at standard room temperature, $15°C$, and the air density is $\rho = 1.225 \frac{kg}{m^3}$.

c) Now you heat the volume of gas so the air temperature rises to $T = 25 \pm 0.2°C$ and the pressure is measured at $P = 104847 \pm 52$ Pa. Each of these measurements is given as the mean value plus or minus one standard deviation.

d) Solve for the density of the heated volume of gas and estimate the error in the calculation both analytically, and using a Monte Carlo Simulation.

e) Keeping the error in temperature constant, vary the error in pressure from ± 0 Pa to ± 800 Pa in steps of 50 Pa. For each of these error values, run a Monte Carlo Simulation and compute the associated error in the density.

f) Plot the varying error in pressure versus the computed error in density. This plot is the sensitivity of the error in density with respect to the error in pressure for constant error in Temperature.

g) Submit your Python code and plots as a pdf as separate files.

Solution. Computing the value of ρ by hand (analytically) and using a Monte Carlo simulation, we see the error is lower when utilizing a Monte Carlo simulation at 1000 iterations per run. Table 1 gives the summary of this analysis. The error in the analytical solution is given by Equation (2). This equation uses the uncertainties of all variables used in calculating the function q, where $\frac{\partial q}{\partial x}\delta x$ is the uncertainty in q from the variable x. In this case, the function being analyzed is Equation (1).

$$\rho = \frac{P}{RT} \tag{1}$$

The error reported in the MC trials is one standard deviation, σ.

$$\partial q = \left(\left(\frac{\partial q}{\partial x} \delta x \right)^2 + \cdots + \left(\frac{\partial q}{\partial z} \delta z \right)^2 \right)^{1/2} \tag{2}$$

The second deliverable in this section requires keeping the error in temperature constant and varying the pressure measurement error from ± 0 Pa to ± 800 Pa in increments of 50 Pa. The results of this simulation are shown as a sensitivity plot (Figure 3), where the $y-$axis shows the estimated value for ρ, and the $x-$axis shows the pressure error. Even with zero error in pressure, there is still a random error present in temperature, which is why the plot doesn't converge to the exact solution at zero pressure error. As the number of iterations increases, the fidelity increases. Since the trials are random (but uniform) distributions

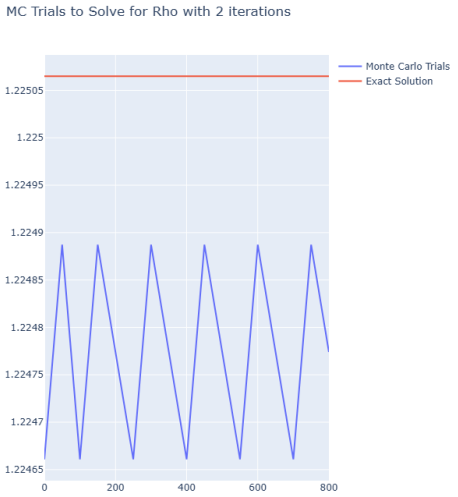
Method	Value of ρ, kg/m^3	Number of Trials	Error
Analytical	1.22506	n/a	0.0010
Monte Carlo	1.22588	2	0.0003
Monte Carlo	1.22498	10	0.0007
Monte Carlo	1.22508	100	0.0006
Monte Carlo	1.22509	1000	0.0006

Table 1: Computing Density from the Ideal Gas Law using Monte Carlo Simulation

about zero, the solutions end up converging near the exact solution, even though the error range may be large. The code for this deliverable is in Section 3.

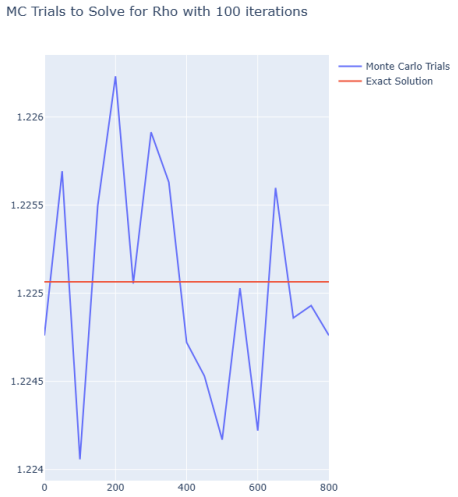
3 Full Code: Task 3

```
'''
Justin Self
AERO 500 (Aerospace Modeling and Simulation)
Spring 2025
Assignment 1: Stochastic Methods

Part 3: Ideal Gas Law
'''
# import modules
import random
import math
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd
import statistics as stats

# Task 3: Ideal Gas Law
print(" ")
print("------ TASK 3 ------")

# initial conditions
# known: rho0 = 1.225  kg/m3
T0 = 25 # C
T0 = T0 + 273.15 # in K now
P0 = 104847 # Pa
R = 287.053 # J/kg.K

# analytically
rho1 = P0/(R*T0)
print("By hand, rho is: " + str(rho1) + " kg/m3")

# need to compute delta (error) in this.
# Monte Carlo Trials
T = []
P = []
#global rho
```

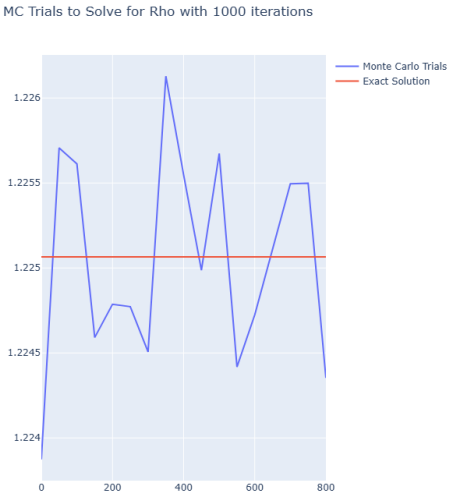


(a) 2 Trials

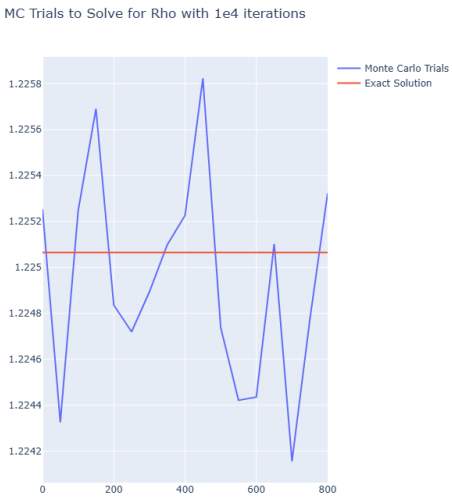
(b) 10 Trials



(c) 100 Trials



(d) 1000 Trials



(e) $1e+04$ Trials

Figure 3: Approximating ρ with MC Trials

```
rho = []
numTrials = 10000

for i in range(numTrials):
    Terr = random.uniform(-0.2,0.2)     # C
    Perr = random.uniform(-52,52)       # Pa
    T.append(T0 + Terr)
    P.append(P0 + Perr)
    rho.append(P[i] / (R * T[i]) )

# find average val of rho
rhoMean = stats.mean(rho)

# NOW USE stats stdev() and compare w hand/analytic.
print("Using MC, rho is: " + str(rhoMean) + " kg/m3")

# Error (sigma; standard dev)
# experimental error
delP = 52                   # error in P is \pm 52 Pa
delR = 0.001                # error in R is last sig fig; or 0.001
delT = 0.2                  # error in T is: 0.2 C
a = 1/(R*T0)*delP
b = -(P0/T0)*(1/R**2)*delR
c = -(P0/R)*(1/T0**2) * delT

z = a**2 + b**2 + c**2
delRho = math.sqrt(z)
mcSig = stats.stdev(rho)

# print errors
print("For a Monte Carlo sim with " + str(numTrials) + " runs:")
print("Error in analytical solution is: " + str(delRho))
print("Error (stdDev) in MC sim is: " + str(round(mcSig,4)))

print(" ")
print("Next section: KEEP Terr constant; vary pressure error.")
print(" ")
# Next part: keep error in T constant, but vary the error in Pressure by:
    (0,800,50)
# make function

# make function
def idealGasMC(P0,T0,Perr):
    n = numTrials
    for x in range(n):
        Terr = random.uniform(-0.2,0.2) + 273.15     # K
        PerrNew = random.uniform(-Perr,Perr)
        T.append(T0 + Terr)
        P.append(P0 + Perr)
        #print("P is: " + str(P0) + " + " + str(round(Perr,3)) + " Pa")
        #print("T is: " + str(T0) + " + " + str(round(Terr,3)) + " K")
        rho.append(P[x] / (R * T[x]) )
        #print("MC iteration count is: " + str(x) + " ~~~~~~~~~~~~~~~~~~")
        #print("MC Run Perr is: " + str(PerrNew))
```

```
    rhomean = stats.mean(rho)

    return rhomean

# prepare to call function
PerrRange = np.linspace(0,800,17)
rhoList = []

# call function once
Perr = PerrRange
for i in range(17):
    print("Run # is: " + str(i))
    rho.append(idealGasMC(P0,T0,PerrRange[i]))
    print("Perr is within (-" + str(PerrRange[i]) + ", +" + str(PerrRange[i]) + ")
    ")
    print("Rho mean is: " + str(rho[i]))
    print("------------")

# seems to be working now. Let's plot it.

# Add trace for inside points
exactSoln = []
for x in range(len(PerrRange)):
    exactSoln.append(rho1)

#print(exactSoln)

fig = go.Figure(layout_title_text="MC Trials to Solve for Rho with 1e4 iterations"
    )
fig.add_trace(go.Scatter(x = PerrRange,y = rho,mode='lines',name='Monte Carlo
    Trials'))
fig.add_trace(go.Scatter(x = PerrRange, y = exactSoln, mode='lines',name='Exact
    Solution'))
fig.write_html('first_figure.html', auto_open=True)
#fig.show()
```

Algorithm 3: Task 3

Parameter	Value
Mean	73396.6
Median	72250.4
Standard Deviation	16427.0
Variance	269846329.0
Minimum	26141.9
Maximum	131470.3

Table 2: Earnings Summary Statistics

Problem 4

a) Write Python code to solve Example 5.2 from the book, Sokolowski, John A., and Banks, Catherine M., MODELING AND SIMULATION FUNDAMENTALS - Theoretical Underpinnings and Practical Domains, Wiley, ISBN 978-0-470-48674-0, 2010.

b) Complete all steps in the example including finding the 95% confidence interval and the sigma normalized sensitivity derivatives.

c) You only need to plot the data associated with the sensitivity analysis (no histograms or probability distributions).

d) Submit your Python code and plots as a pdf as separate files.

Solution. This exercise was helpful in getting comfortale with math and plotting operators in Python. The solutions presented here closely resemble the solutions in the textbook [1], being on the same order of magnitude and numerically very similar. Small differences are due to the variability inherent in the Monte Carlo scheme.

The summary statistics for the earnings computation using a Monte Carlo simulation is shown in Table 2. The 95% confidence interval is:

$$[L(Y), U(Y)] = [73074.6, 73718.6]$$

The sensitivity analysis plots are shown in Figure 4 and match the expected plots well. One method of analyzing sensitivity is simply visualizing how the input variable affects the output variable (earnings, in this case). This method employs a scatter plot where each input variable in the Monte Carlo simulation is individually plotted against the output variable and the resulting pattern is analyzed. The more structured the output pattern, the more sensitive is the output variable to that input variable [1]. The variable and fixed costs show a fairly structured (normally distributed) spread, and the multiplicative plot (unit sales * unit price) trends upward, as expected. The qualitative assessment in these plots indicates that this model is more or less robust to input variables through Monte Carlo simulations.

The code for this deliverable is included in Section 4.

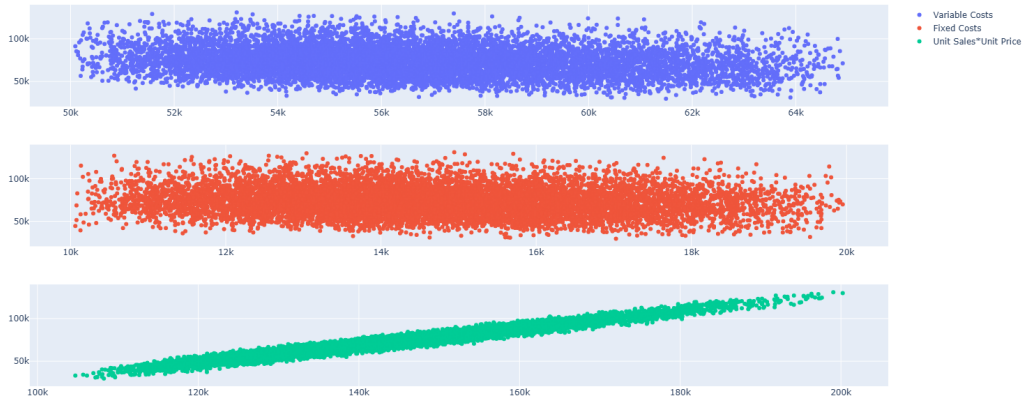


Figure 4: Earnings by Input Variable

4 Full Code: Task 4

```
'''
Justin Self
AERO 500 (Aerospace Modeling and Simulation)
Spring 2025
Assignment 1: Stochastic Methods

Part 4: Example 5.2 (Sokolowski, Banks)
'''
# import modules
import random
import math
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd
import statistics as stats
from plotly.subplots import make_subplots

# Task 4: Example 5.2
print(" ")
print("------ TASK 4 ------")
# Solve Ex 5.2
# Complete all steps including finding the 95% CI
# and sigma normalized sensitivity derivatives
# Plot only data associated w sensitivity analysis
# no prob dist; histograms

# Example 2: Computing Product Earnings (pps 136 and following)

earnings = []
vc = []
fc = []
```

```
usup = []

for i in range(10000):
    unitPrice = random.triangular(50,70,55)
    unitSales = random.triangular(2000,3000,2440)
    varCosts = random.triangular(50000,65000,55200)
    fixedCosts = random.triangular(10000,20000,14000)
    vc.append(varCosts)
    fc.append(fixedCosts)
    usup.append(unitSales * unitPrice)
    earnings.append(unitPrice * unitSales - (varCosts + fixedCosts))

#print(f'length of earnings is: {len(earnings)}')

# Print statistics (Table 5.3)
mean = round(stats.mean(earnings),1)
median = round(stats.median(earnings),1)
sig = round(stats.stdev(earnings),1)
variance = round(sig**2,1)
minimum = round(min(earnings),1)
maximum = round(max(earnings),1)

print(f'mean: {mean}')
print(f'median: {median}')
print(f'standard dev: {sig}')
print(f'variance: {variance}')
print(f'min: {minimum}')
print(f'max: {maximum}')

# Good; these check out with the book (Table 5.3) order of magnitude.
# CI
z = 1.96 # given; for alpha = 0.05
n = len(earnings)
sigmaN = sig
thetaList = []
theta = 0
for i in range(n):
    thetaList.append(earnings[i]/ n)
    theta += thetaList[i]

lower = round(theta - z * sigmaN / math.sqrt(n),1)
upper = round(theta + z * sigmaN / math.sqrt(n),1)
print("95% CI is: (" + str(lower) + ", " + str(upper) + ")")
print(" ")

# seems fair

# Sigma normalized Sens Analysis #
# scatter plot to show input and output vars
fig = make_subplots(rows=3,cols=1)
# var costs
fig.add_trace(
    go.Scatter(x=vc,y=earnings,mode="markers",name="Variable Costs"),row=1,col=1
)
```

```
# fixed costs
fig.add_trace(
    go.Scatter(x=fc,y=earnings,mode="markers",name="Fixed Costs"),row=2,col=1
)
fig.add_trace(
    go.Scatter(x=usup,y=earnings,mode="markers",name="Unit Sales*Unit Price"),row
    =3,col=1
)
# unit sales * unit price

fig.write_html('Figtitle.html', auto_open=True)
```

Algorithm 4: Task 4

References

[1] John A Sokolowski and Catherine M Banks. *Modeling and simulation fundamentals: theoretical underpinnings and practical domains*. John Wiley & Sons, 2010.