CSE 20
Beginning Programming in Python
Programming Assignment 5

In this project you will write a Python program that computes the expected value of the determinant of a random matrix. You will do this by using a mathematical technique known as *Monte Carlo*, that uses random sampling to compute solutions to problems which would otherwise be difficult or impossible to access. You will also write a recursive function that computes the determinant of a square matrix, represented as a list of lists of numeric values.

Matrices and Determinants

A *matrix* is a rectangular array of objects (usually real numbers) arranged in rows and columns. A matrix is called *square* if the number of rows equals the number of columns. If a square matrix has n rows and n, columns we call it an $n \times n$ *square matrix*. We can represent such a matrix in Python as a list of length n, of lists of length n. The i^{th} element (list) in the outer list constitutes the i^{th} row ($1 \leq i \leq n$) of the matrix, and the j^{th} member of each interior list, taken together, constitute the j^{th} column ($1 \leq j \leq n$).

For example, the Python list $[[a,\ b],\ [c,\ d]]$ would represent the 2×2 square matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

and $[[a, b, c], [d, e, f], [g, h, i]]$ would represent the 3×3 matrix

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix},$$

while the 1×1 matrix (a) would be represented simply as $[[a]]$.

The *determinant* of a square matrix can be defined in a number of equivalent ways. For purposes of this assignment, we give a purely computational definition. Let A be an $n \times n$ matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

where a_{ij} denotes the element in the i^{th} row and j^{th} column ($1 \leq i \leq n, 1 \leq j \leq n$). We denote by $A\{i, j\}$ the $(n-1) \times (n-1)$ submatrix obtained by deleting the entire i^{th} row and the entire j^{th} column from A.

$$A\{i,j\} = \begin{pmatrix} a_{11} & \cdots & a_{1(j-1)} & a_{1(j+1)} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{(i-1)1} & \cdots & a_{(i-1)(j-1)} & a_{(i-1)(j+1)} & \cdots & a_{(i-1)n} \\ a_{(i+1)1} & \cdots & a_{(i+1)(j-1)} & a_{(i+1)(j+1)} & \cdots & a_{(i+1)n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{n(j-1)} & a_{n(j+1)} & \cdots & a_{nn} \end{pmatrix}$$

We now define $\mathrm{Det}(A)$ to be a_{11} if $n = 1$, and

$$\mathrm{Det}(A) = \sum_{j=1}^{n} (-1)^{j-1} a_{1j} \mathrm{Det}(A\{1, j\})$$

$$= a_{11}\mathrm{Det}(A\{1,1\}) - a_{12}\mathrm{Det}(A\{1,2\}) + \cdots \pm a_{1n}\mathrm{Det}(A\{1,n\})$$

if $n \geq 2$. This definition is not as complicated as it may at first appear. Observe that it is a *recursive* definition, i.e. you must first be able to compute the determinant of an $(n-1) \times (n-1)$ matrix, before you can compute that of an $n \times n$ matrix. Some examples will help clarify.

1×1: $\mathrm{Det}(a) = a$

2×2: $\mathrm{Det}\begin{pmatrix} a & b \\ c & d \end{pmatrix} = a\mathrm{Det}(d) - b\mathrm{Det}(c) = ad - bc$

3×3: $\mathrm{Det}\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = a\mathrm{Det}\begin{pmatrix} e & f \\ h & i \end{pmatrix} - b\mathrm{Det}\begin{pmatrix} d & f \\ g & i \end{pmatrix} + c\mathrm{Det}\begin{pmatrix} d & e \\ g & h \end{pmatrix}$

$$= a(ei - fh) - b(di - fg) + c(dh - eg)$$
$$= aei - afh - bdi + bfg + cdh - ceg$$

4×4: $\mathrm{Det}\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & p & q \end{pmatrix}$

$$= a\mathrm{Det}\begin{pmatrix} f & g & h \\ j & k & l \\ n & p & q \end{pmatrix} - b\mathrm{Det}\begin{pmatrix} e & g & h \\ i & k & l \\ m & p & q \end{pmatrix} + c\mathrm{Det}\begin{pmatrix} e & f & h \\ i & j & l \\ m & n & q \end{pmatrix} - d\mathrm{Det}\begin{pmatrix} e & f & g \\ i & j & k \\ m & n & p \end{pmatrix}$$

$$= a(\cdots\cdots\cdots\cdots\cdots\cdots) - b(\cdots\cdots\cdots\cdots\cdots\cdots) + c(\cdots\cdots\cdots\cdots\cdots\cdots) - d(\cdots\cdots\cdots\cdots\cdots\cdots)$$

We leave it as an exercise for the reader to fill in the dots in the last expression. Obviously, computing a determinant by hand using this definition is very tedious for $n \geq 4$. One of your tasks in this project is to write a recursive function with the following heading and doc string.

```
def Det(M):
    """
    Returns the determinant of a square matrix M.
    """

# end Det()
```

This function will take as input a list M of lists of numbers, representing an $n \times n$ square matrix. It will then compute the determinant of M using the above recursive procedure, and return the computed value. One subtlety is that fact that Python lists always begin at index 0 and end at $n - 1$, where n is the length of the list, while the above description has row and column indices going from 1 to n. Your problem is to perform the necessary index gymnastics to reconcile these two facts.

The Monte Carlo Method

The Monte Carlo method was invented by scientists working on the atomic bomb in the 1940s. They named their technique for the city in Monaco famed for its casinos. The core idea is to use randomly chosen inputs to explore the behavior of a complex dynamical system. These scientists faced difficult problems in mathematical physics, such as neutron diffusion, that were too complex for direct analytical solution, and must therefore be evaluated numerically. They had access to one of the earliest computers (ENIAC), but their mathematical models involved so many parameters that exhaustive numerical evaluation was prohibitively slow. Monte Carlo simulation proved to be surprisingly effective at finding solutions to these problems. Since that time, Monte Carlo methods have been applied to an incredibly diverse range of problems in science, engineering, and finance.

In this project we will attempt to answer the following question. Given a random $n \times n$ matrix, what is the expected absolute value of its determinant? Believe it or not, it is possible to find this number by purely analytical means, but since this is not a class in probability theory, we will use a Monte Carlo approach to compute it. To that end, you will produce (for each $n = 1, 2, 3, 4, 5$) a sequence of 10,000 random $n \times n$ matrices, whose entries are uniformly distributed on the interval $(-5, 5)$. You will compute the sequence of determinants that results, take the absolute value of each determinant, then compute the average of this quantity over all 10,000 matrices. (Note that these random determinants are equally likely to be positve or negative, so on average they would cancel to zero, which is why we first take the absolute value.)

You can find a very interesting history of early computing machines, Monte Carlo methods, and the development of the atomic bomb in the book *Turing's Cathedral* by George Dyson. Follow the link

https://en.wikipedia.org/wiki/Monte_Carlo_method

for an article on Monte Carlo methods.

Program Specifications

Your source code file for this assignment will be called Determinant.py, and will follow the general template found in /Examples on the course webpage. You must import the random module to produce a sequence of random numbers. Your program will take no user or command line input, and will print a table similar to the one below.

```
$ python3 Determinant.py

 n          average of abs(det(M))
---------------------------------
 1                 2.47198
 2                 9.23084
 3                43.30913
 4               239.34575
 5              1483.19133

$
```

As usual, your program output must match the above formatting exactly to get full credit, including blank lines and spaces. Study the examples FormatNumbers1.py and FormatNumbers2.py posted on the class webpage to learn how to format your output correctly.

The values in the table body should be close to those printed above, but not exactly the same. (Question: why does the above value for $n = 1$ make perfect sense?) Below is a more detailed template for your

program, including some comments giving the basic logic of function `main()`. Observe that `main()` will take one argument, whose default value is `None`, and that this value is used to seed the random number generator. This will allow us to predetermine your sequence of random numbers by calling `main()` with a specific argument. Also note that if you call `main(3517)` in your trailing conditional, you should produce exactly the same numbers as in the above model output.

```
#-----------------------------------------------------------------
#  Determinant.py
#-----------------------------------------------------------------
import random

def Det(M):
   """
   Returns the determinant of a square matrix M.
   """

# end Det()

# main() ----------------------------------------------------------
def main(seed=None):

   rng = random.Random(seed)

   # print table heading

   # compute and print table body

   # for each n from 1 to 5

      # for each k from 1 to 10,000

         # construct a random nxn matrix M
         # using rng.uniform(-5,5) to produce its entries

         # compute the absolute value of the determinant
         # of M and add it to an accumulating sum

      # compute the average of the absolute value of the determinants

      # print the average, formatted for the body of the table

# end main() ------------------------------------------------------

#-----------------------------------------------------------------
if __name__=="__main__":

   main()

# end if ----------------------------------------------------------
```

Submit `Determinant.py` to the assignment pa5 on Gradescope before the due date. As always, start early and ask plenty of questions.