Tarea #1B
Dos Métodos Numéricos para Aproximar el Valor de π
Informe Escrito

Christian Guerra
18.640.812-8
Lunes 8 de Abril

La estrategia general adoptada para resolver este problema fue utilizar una hoja en blanco e intentar deducir las formulas requeridas para cada función antes de comenzar a generar código, las funciones después fueron utilizadas como parte de otras funciones para facilitar el proceso y finalmente el programa principal fue programado.

El programa principal no tuvo mayores dificultades, pero en las funciones hubo algunas complicaciones, como la deducción de formulas que debían usarse para llegar al resultado requerido de la función, si los valores debían ponerse por primera vez dentro de la función o fuera y como hacer que los números aleatorios cambien de valor en cada iteración.

El programa principal básicamente consiste de un "input" que pide el numero de iteraciones, un ciclo para verificar si un numero es positivo y en el caso de no serlo que vuelva a preguntar por las iteraciones de nuevo. Si el valor entregado es mayor o igual que uno, hay un ciclo más que tiene un contador y desarrolla una tabla con tantas iteraciones como fueron pedidas utilizando las funciones que fueron creadas.

Las formulas específicas y formas en que fueron desarrolladas cada función son las siguientes:

1. Aproximación por la serie de Leibniz:
    a. **termino** (i): esta función recibe un numero i y devuelve el i-ésimo termino de la serie que se encuentra entre -0.333… a 1 lo cual esta definido por un algoritmo que se divide en dos partes, la primera devuelve fracciones positivas o negativas dependiendo si el numero es par o impar (si i es impar devuelve un valor positivo, si es par devuelve un valor negativo), y la segunda parte consiste en determinar el valor del denominador que es igual a el doble del termino i menos 1. El algoritmo utilizado entonces fue: ((-1.0)**(i+1))/((i*2)-1).
        i. Ejemplos de prueba:
            1. Si i = 3 entonces devuelve 1/5 = 0.2
            2. Si i= 4 entonces devuelve -1/7 = -0.14285714
            3. Si i=1000 entonces devuelve -1/1999 = -0.00050025013
    b. **suma** (n): esta función recibe un número n y devuelve la suma del n-ésimo término más todos los términos anteriores. Mientras mayor sea n, mas cercano será el termino devuelto al valor deseado, los valores varían entre 0.6666… a 1. El algoritmo utilizado esta basado en la función anterior (**termino**) y utiliza un ciclo (**while**) que tiene un contador (nsec), el cual es

igual al termino n al principio pero por cada repetición del ciclo se resta 1 a su valor, si llega a valer 1, el ciclo se rompe. El valor devuelto se basa en sumar **termino** (n) con **termino** (n-1), **termino** (n-2), **término** (n-…..) hasta sumar termino (1), así sumando todos los valores desde el primer termino hasta el termino del numero n.

i. Ejemplos de prueba:
    1. Si n = 3 entonces devuelve 0.86666…
    2. Si n = 4 entonces devuelve 0.72380952381
    3. Si n =1000 entonces devuelve 0.78514816346

c. **leibniz** (n): esta función recibe un numero n y devuelve una aproximación del numero pi., los valores varían entre 2.6666… a 4 el algoritmo utilizado esta basado en utilizar la función **suma** (n) y multiplicar el valor devuelto por 4, ya que la función suma equivale a ¼ de pi.

i. Ejemplos de prueba:
    1. Si n = 3 entonces devuelve 3.46666666667
    2. Si n = 4 entonces devuelve 2.89523809524
    3. Si n =1000 entonces devuelve 3.14059265384

2. Aproximación utilizando un algoritmo de tipo Montecarlo:
    a. **estaDentro** (x, y): esta función recibe dos valores, x e y, los cuales representan las coordenadas en un plano cartesiano (ambos valores varían entre 0 y 2), y devuelve dos valores dependiendo si las coordinadas dadas se encuentran dentro de el circulo circunscrito en el cuadrado, devolviendo True si se encuentra adentro del circulo y False si se encuentra fuera. Esta función fue creada utilizando una formula entregada en la tarea que es la raíz de $(x-1)^2 + (y-1)^2$. En el caso de que la formula utilizada de un valor menor o igual a 1 entonces devuelve True, o sino, devuelve False.
        i. Ejemplos de prueba: (los valores pueden variar ya que son generados aleatoriamente),
            1. (x, y) = (1,1) devuelve True
            2. (x, y) = (2,2) devuelve False
            3. (x, y) = (0,0) devuelve False
            4. (x, y) = (0.456 , 0.1) devuelve False
    b. **montecarlo** (n): esta función recibe un numero n, que es el numero de casos que debe verse si **estaDentro** (x, y) es True y formar una fracción entre los casos que es True dividido por n y después multiplicar por 4, por el mismo motivo que multiplicamos por 4 en **leibniz** (el algoritmo calcula pi/4, al multiplicar por 4 calculamos pi.). El valor puede variar entre 0 y 4 (si todos los casos son False o True respectivamente). Los dos puntos del plano cartesiano son generados por un operador aleatorio y después se usa un ciclo, el cual resta uno a n al repetir el ciclo, hasta que n sea igual a 0, rompiendo el ciclo. Dentro del ciclo si las coordenadas son True aparte de restar n-1, también suma un número a otro contador que es la cantidad de casos True.

i. Ejemplos de prueba:
    1. n = 3 devuelve 4
    2. n = 3 (de nuevo) devuelve 2.66666666667
    3. n = 1000 devuelve 3.18
    4. n = 1000 (de nuevo) devuelve 3.148

3. Análisis comparativo de los métodos de aproximación:
    a. **errorPorcentual** (valor): esta función recibe un valor denominado (valor), que puede ser tanto la función **montecarlo** (n) o **leibniz** (n) y devuelve el porcentaje de error en comparación con el numero pi, que puede variar entre 0 y 100. El algoritmo requerido es bien simple ya que es sacar porcentaje, es decir se calcula 100- (100*valor/pi).
        i. Ejemplos de prueba:
            1. Valor= leibniz (3), devuelve 89.6525727896
            2. Valor= montecarlo (3), devuelve 72,6760455265
            3. Valor= leibniz (100), devuelve 0.318301929431
            4. Valor= montecarlo (99), devuelve 0.97025763171
            5. Valor= montecarlo (100), devuelve 9.49860084722

Ya que todas las funciones creadas para el programa ahora usaremos 3 ejemplos de prueba para ver si el programa funciona correctamente:

1. Ejemplo de prueba con números positivos:
    a. 2 iteraciones:

Ingrese el número de iteraciones: 2

N	Leibniz	Error	Montecarlo	Error
1	4.0	27.3239544735	4.0	27.3239544735
2	2.66666666667	15.1173636843	2.0	27.3239544735

    b. 5 iteraciones:

Ingrese el número de iteraciones: 5

N	Leibniz	Error	Montecarlo	Error
1	4.0	27.3239544735	4.0	27.3239544735
2	2.66666666667	15.1173636843	0.0	27.3239544735
3	3.46666666667	10.3474272104	2.666666667	15.1173636843
4	2.89523809524	7.84170914298	3.0	4.50703414486
5	3.33968253968	6.30539690963	2.4	23.6056273159

2. Ejemplo de prueba con números negativos y 0:
    a. (-1) iteraciones:

Ingrese el número de iteraciones: -1
Por favor, ingrese un número positivo!!
Ingrese el número de iteraciones:

    b. 0 iteraciones:

Ingrese el número de iteraciones: 0
Por favor, ingrese un número positivo!!
Ingrese el número de iteraciones:

3. Ejemplo de prueba con números decimales:
    a. **2.83 iteraciones:**

Ingrese el número de iteraciones: 2.83

N	Leibniz	Error	Montecarlo	Error
1	4.0	27.3239544735	4.0	27.3239544735
2	2.66666666667	15.1173636843	4.0	27.3239544735

    b. **(-2.83) iteraciones:**

Ingrese el número de iteraciones: -2.83
Por favor, ingrese un número positivo!!
Ingrese el número de iteraciones:

En los ejemplos se puede ver que el programa funciona sin problema dentro de los valores nominales (los enteros positivos), los números decimales ignoran la posición decimal, solo consideran el numero entero (el programa es una secuencia, debe funcionar con números enteros) y en los valores negativos que no son posibles en la secuencia, el programa pide el ingreso de un numero positivo (incluso si es un negativo decimal). Se puede ver que el programa es suficiente robusto para el objetivo que se requiere de este.

En esta tarea principalmente aprendí como utilizar ciclos y funciones, la tarea de base exige el uso de 5 funciones distintas, ejercitando el uso de estas y requiriendo el conocimiento de como utilizarlas (es necesario crear funciones y utilizar funciones dentro de otras funciones) y los ciclos son primordiales para el correcto funcionamiento de las funciones y el programa final, el cual requiere imprimir en pantalla tantas iteraciones como sean pedidas. La tarea es un buen ejercicio para entender la materia pasada hasta el momento (print, input, if, def, while, etc.…) y ayuda a formar una buena base para lo que es programación utilizando variables (numéricas), pensar en formulas matemáticas que son necesarias para los ciclos, recursión y ecuaciones en general desde una perspectiva como programador. También es satisfactorio ver los resultados una vez que funciona el programa generando un mayor gusto por la programación ya que requiere un desafío y una vez completo genera un resultado positivo (satisfacción al ver el funcionamiento del programa).