95.13

ANALISIS NUMERICO

TPM1: ERRORES.

Nicolas Facciano 105859
Juan Biancuzzo 106005

1er.Cuatrimestre de 2022

Docentes a cargo del curso:

- Rodriguez Daniel.F
- Machiunas Valeria

Objetivo:
Estudiar errores de truncamiento, la propagación de errores inherentes y de redondeo.

Desarrollo del práctico:

1) Unidad de máquina:

Determine la unidad de máquina (en notación decimal) (μ) para simple y doble precisión.
Indicar además la cantidad de dígitos binarios con los que trabaja la máquina (t) en simple y doble precisión y la unidad de máquina (μ) correspondiente.

2) Errores:

Se quiere aproximar la suma infinita,

$$S = x\ \left[1 - \frac{x^2}{2\ 1!\ 3} + \frac{x^4}{2^2\ 2!\ 5} - \frac{x^6}{2^3 3!\ 7} + \ldots\right], \text{ en } x = A,$$

donde A es igual $a^1 = \frac{P_1 + P_2}{10^6}$

a) Determine (analíticamente) la cantidad de términos necesaria que hay que sumar para que el error de truncamiento sea menor a 10^{-14}
b) Halle (en forma experimental) la condición del problema y el término de estabilidad.
c) Suponiendo que A no tiene error, halle una cota para el error total.
d) Ahora suponga que A tiene un error relativo menor al 0,01%, halle una cota para el error total.
e) Repita a, b, c y d con x = 15 * A.
f) Extraiga conclusiones

Juan Biancuzzo 106005
Nicolas Facciano 105859

Introducción:

En este Tp los principales objetivos son estudiar y analizar los distintos errores que aparecen a la hora de calcular en este caso, la serie infinita que se nos presenta.

Primeramente, debemos tomar en cuenta nuestra unidad de máquina para poder observar y determinar el error de redondeo que ocurre en dicho calculo, que es debido a la cantidad finita de dígitos que nos limita trabajar en una computadora.

Para analizar y determinar la unidad de maquina pensamos en calcularla a través de un programa hecho en Python que se mostrara en el **anexo 1**

Consecuentemente a la hora de trabajar con la serie infinita nos deberíamos encontrar con diferentes problemáticas que nombraremos a continuación:

- En el cálculo de la unidad de maquina debemos averiguar cuantas iteraciones hay hasta el error relativo máximo de representación.
- Plantear el pasaje de un problema matemático a un problema numérico (de una serie infinita a una serie parcial).
- Cálculo de errores experimentales.
- Analizar las distintas relaciones y sus repercusiones entre los distintos errores que aparecen.

Estos distintos pasos que fuimos tomando van a ser explicados con mayor claridad en el **desarrollo** y el código fuente expresado en el **anexo 1**

Desarrollo:

Como primer paso, calculamos la unidad de maquina mediante el uso de un seudocódigo encontrado en el libro Análisis Numérico primer curso[1], que mediante la deducción que podemos generalizar la unidad de redondeo de la siguiente forma:

$$\mu = \begin{cases} B^{1-t} & para\ corte \\ \frac{1}{2}B^{1-t} & para\ redondeo\ simetrico \end{cases}$$

Y mediante dicha deducción, el libro nos presenta el seudocódigo siguiente para el cálculo de “t”

Variables: $s = 1;$
$t = 1;$
$x = 2;$
Mientras $x > 1$ hacer
Escribir x, t
$t = t + 1;$
$s = \frac{s}{10};$
$x = 1 + s;$
Finmientras
Fin.

Juan Biancuzzo 106005
Nicolas Facciano 105859

Este algoritmo, lo que hace es ir escribiendo sucesivos valores de x y de t para hacer más grafica la situación; el ultimo valor de t que escribe nos indica aproximadamente a cuantos dígitos equivalen las cifras que utiliza la máquina para guardar las mantisas en la base por ella utilizada. Lo único que tuvimos que hacer fue entender cómo funcionaba el algoritmo y llevarlo al lenguaje de Python para hacer el código del programa.

Por otra parte, se aproximó la suma infinita a una serie y es aquí donde se ve como hay un problema matemático que se puede transformar en un problema numérico, en el cual se aprecia un error de truncamiento.

La siguiente suma $S = x\ [1-\frac{x^2}{2\ 1!\ 3}+\frac{x^4}{2^2\ 2!\ 5}-\frac{x^6}{2^3 3!\ 7}+\ldots]$, en x = A, la pudimos pensar mediante la serie parcial a continuación:

$$S = x\sum_{k=0}^{N}(-1)^k * \frac{x^{2k}}{2^k * k! * (2k+1)} = \sum_{k=0}^{N}(-1)^k * \frac{x^{2k+1}}{2^k * k! * (2k+1)}$$

Luego de comprobar que es una sucesión decreciente, para poder acotarla como una serie alternada mediante el criterio de Leibniz que investigamos en la página que nombramos en la bibliografía usada; de la siguiente manera:

- Comprobación de decrecimiento de la sucesión:

$a_k = \frac{x^{2k+1}}{2^k * k! * (2k+1)}$ $\qquad$ $a_{k+1} = \frac{x^{2k+3}}{2^{k+1} * (k+1)! * (2k+3)}$

$$\frac{a_{k+1}}{a_k} = \frac{x^{2k+3}}{2^{k+1} * (k+1)! * (2k+3)} \cdot \frac{2^k * k! * (2k+1)}{x^{2k+1}} =$$

$$= \frac{x^{2k} \cdot x^3}{2^k \cdot 2^1 * (k+1) k! * (2k+3)} \cdot \frac{2^k \cdot k! \cdot (2k+1)}{x^{2k} \cdot x^1} =$$

$$= \frac{x^2 \cdot (2k+1)}{2 * (k+1) * (2k+3)} =$$

$$\lim_{k \to \infty} \frac{x^2 \cdot (2k+1)}{2 * (k+1) * (2k+3)} = 0$$

Como se puede ver es una sucesión que da 0 cuando k tiende a infinito por lo que se puede decir que es decreciente y queda demostrado que el criterio es válido para esta serie y se puede acotar de la siguiente forma:

$$|L - S_k| = e_k \leq a_{k+1} = \frac{x^{2k+3}}{2^{k+1} * (k+1)! * (2k+3)} \leq 10^{-14}$$

Juan Biancuzzo 106005
Nicolas Facciano 105859

Donde $X = A = (P_1 + P_2) * 10^{-6}$. También vamos a ver el caso donde $X = A * 15$.

Como primer intento, se buscó una expresión matemática más fácil de computar, pero llegamos a la conclusión de que la más accesible era, al menos desde nuestro punto de vista, la siguiente

$$\frac{x^{2k+3}}{2^{k+1} * (k+1)! * (2k+3)}$$

A la hora de hacer el código para analizar el ítem a del trabajo practico, no tuvimos inconvenientes al plantear la ecuación últimamente expresada, e hicimos que esta se repita (itere) con diferentes valores de k (k representa la cantidad de iteraciones) tantas veces como sea necesario hasta llegar al último valor menor a 10^{-14} siendo este el error mínimo que queremos tener.

Por otro lado, si tuvimos inconvenientes al calcular la condición de problema de forma experimental. Quisimos afrontar el problema directamente mediante el computo de los valores del CP para diferentes variaciones del valor (X) y encontramos discrepancias con lo teóricamente esperado.

Mediante el método de cálculo de la Condición del Problema de forma experimental dada por:

$$CP \approx \frac{|f(x \cdot (1 + \delta_x)) - f(x)|}{|f(x)| \cdot |\delta_x|}$$

Donde δ_x es la variación de x mencionada anteriormente.

Lo esperado siendo que para valores de la variación cercanos a 0 el cálculo del CP experimental aumente de manera apresurada, para valores más alejados del 0 el cálculo del CP experimental de nuevo aumenta de manera apresurada, por último, nos quedan valores para los cuales el CP nos da valores aproximados al CP real creando de esta forma una curva semejante a una parábola donde el vértice de esta sería la mejor aproximación de la Condición del Problema.

Finalmente, no era un problema en la implementación sino un error conceptual donde estábamos tomando valores de variación demasiado grandes para un estudio del CP.

Al reducirlos a una escala más acorde en el rango de 10^{-8} y 10^{-11} sí pudimos observar valores del CP más adecuados a la representación teórica que teníamos de ellos.

Para calcular el Termino de Estabilidad de forma experimental tuvimos que hacer una función que haga el cálculo de la factorial (con un ciclo for que itere el valor y lo multiplique por siguiente sucesivamente tantas veces como sea el valor del número), para asegurarnos que el valor de la serie sea completamente en simple y en doble precisión.

Después si se nos hizo más fácil ya que solo tuvimos que programar la función del Termino de Estabilidad de forma experimental que ya sabíamos mediante la teoría; y le cargamos los datos ya obtenidos en el punto anterior donde calculamos la unidad de máquina.

Para el cálculo de la cota del error total, la formula teórica del mismo es:

$$|e_T| \leq CP \cdot r + TE \cdot \mu + e_t$$

Donde r es el error inherente, μ es el error relativo máximo de representación y e_t es el error de truncamiento.

Suponiendo que X no tiene error, el cálculo de la cota del error inherente nos queda de la siguiente forma:

$$|e_T| \leq TE \cdot \mu + e_t$$

Como no tuvimos problemas con el Termino de Estabilidad este cálculo lo pudimos hacer fácilmente en Python. Mientras que al suponer que si tiene un error inherente del 0.01%, si tuvimos dificultades por los problemas que tuvimos al calcular el CP y estos problemas se solucionaron cuando pudimos establecer bien el CP experimental mediante la función factorial para poder guardar los valores en simple o doble.

Conclusiones:

- Luego de haber realizado las operaciones correspondientes para obtener la cantidad de iteraciones necesarias para que el valor sea menor a la cota de error dada como consigna, pudimos observar que decrece de una forma casi lineal el valor por iteración hasta llegar a las 5 iteraciones máximas que nos permiten que el valor sea menor a 10^{-14}.

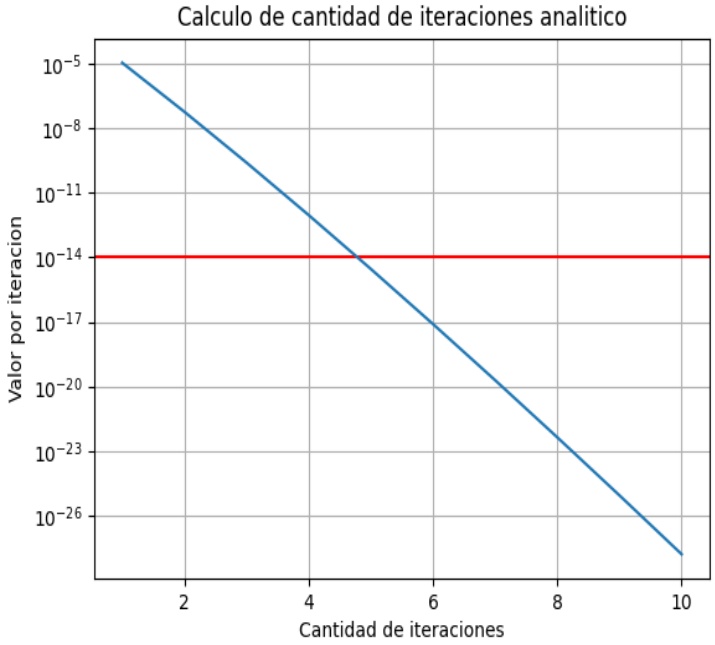

Figura 1.
Grafico de los Valores por iteración de acuerdo con la cantidad de iteraciones

Juan Biancuzzo 106005
Nicolas Facciano 105859

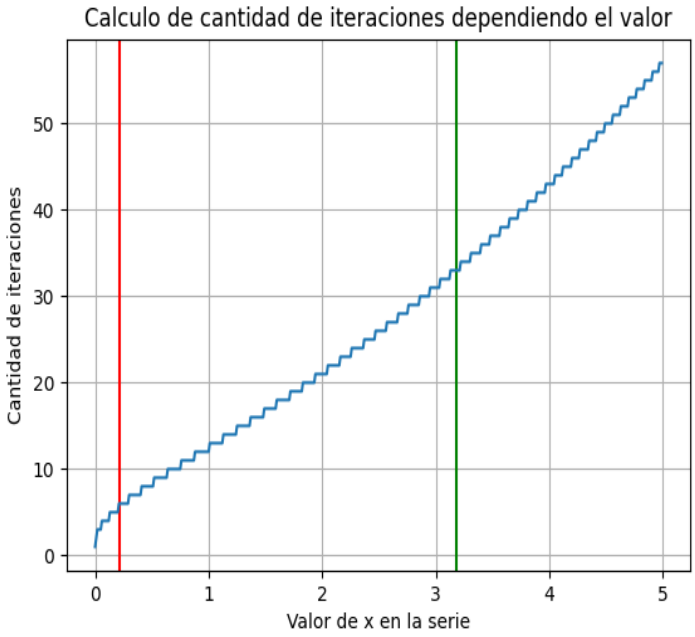

Figura 2.
Cálculo de la cantidad de iteraciones según el valor tomado.
La line roja indica el valor cuando $X = A$
La línea verde indica el valor cuando $X = A * 15$

En la figura 2 se muestra la dependencia de la cantidad de iteraciones necesarias para conseguir la precisión pedida, dependiendo del valor dado a la serie.
Como se puede observar es una representación semejante a una recta, por lo que es evidente que mientras mayor sea el valor se dificulta más conseguir la precisión pedida.

Otras de las deducciones que pudimos observar fue como en cálculo del CP, con perturbaciones muy chicas hay mayor error. Esto es debido a que se está trabajando con valores muy cercanos al 0 y hay una división entre ellos, por lo que generan valores suficientemente grandes tales que no pueden ser representados con precisión en la computadora; Introduciendo así un error en el cálculo con imprecisiones en el mismo resultado. Se muestra esta observación en el gráfico de la **Figura 3.**

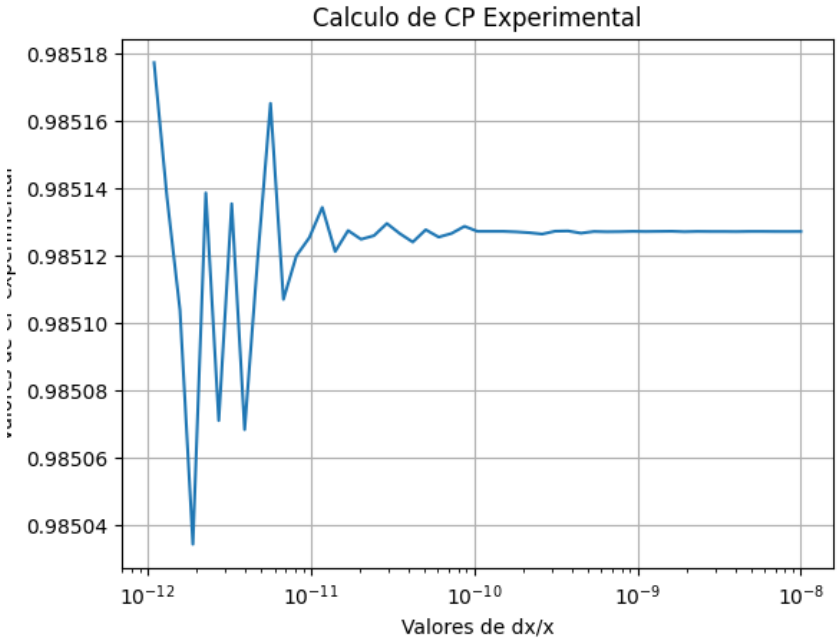

Figura 3.

Representación gráfica de las perturbaciones en el cálculo experimental del CP

- **Influencia del error inherente en el error total**

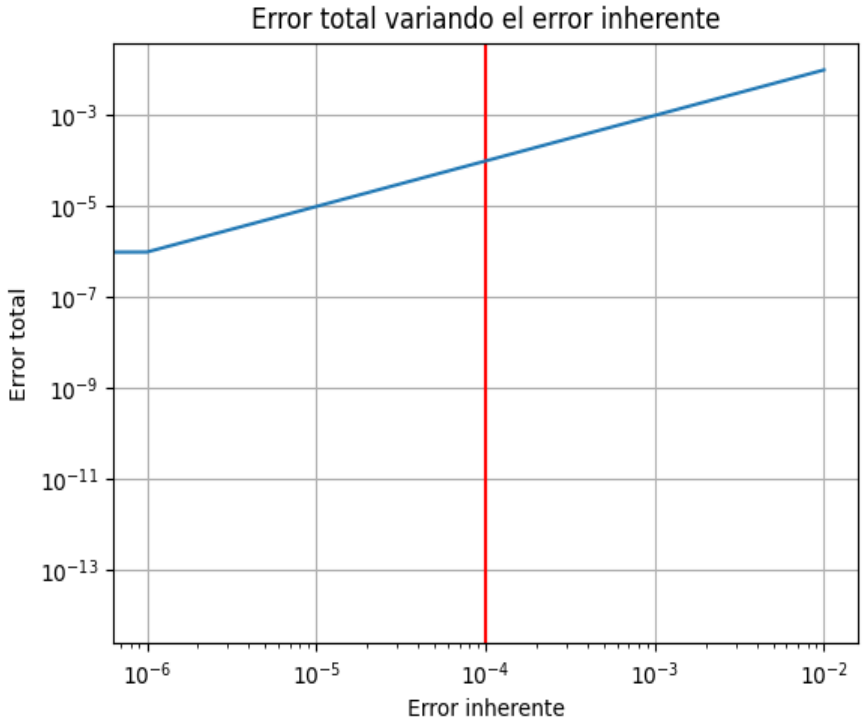

Figura 4.
Error total sobre error inherente

En la diferencia entre cada iteración pudimos observar que siempre de una iteración a la siguiente el resultado es 2 órdenes de magnitud más preciso.

Juan Biancuzzo 106005
Nicolas Facciano 105859

En nuestro modelo los errores inherentes influencian mucho el error total, esto se puede ver como con valores cercanos a 0, es decir sin error inherente, el error total es aproximadamente el error de redondeo

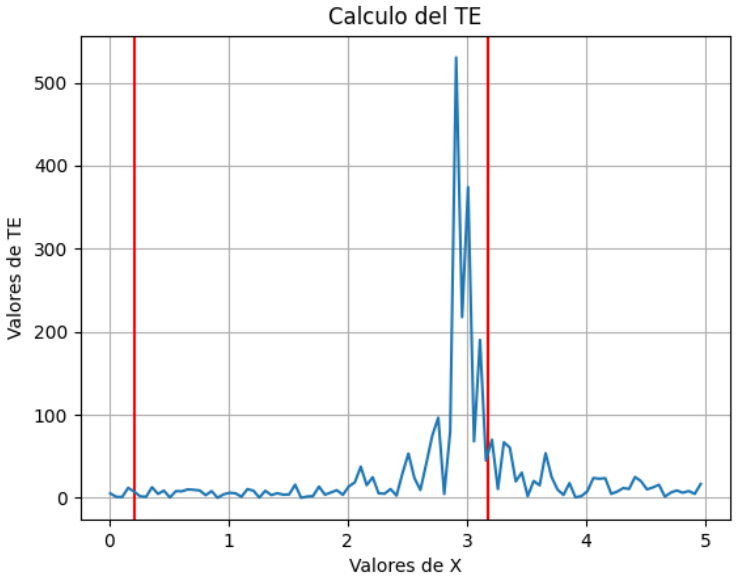

Figura 5.
Representación gráfica del TE experimental

En cuanto al desarrollo del calculo del Termino de Estabilidad se puede observar en la **Figura 5** mediante la representación de las líneas rojas , los valores del TE de acuerdo al valor de X que nos toca evaluar.

Al incrementar nuestro error inherente se observa como opaca todos otros los errores.

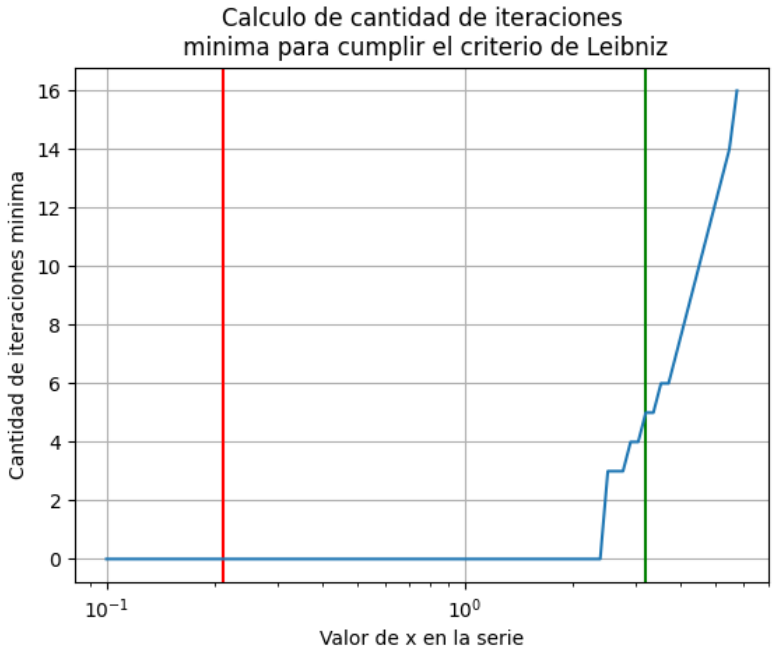

Figura 6.
Representación gráfica de la cantidad de iteraciones mínimas que permiten cumplir con el criterio de Leibniz

Juan Biancuzzo 106005
Nicolas Facciano 105859

Como pudimos observar en el grafico representativo **(Figura 6)** de los valores que podemos usar para comprobar la validez del criterio de Leibniz; de acuerdo al valor que se tome para la variable X va a determinar una cantidad mínima de iteraciones para que se cumplan las hipótesis del criterio mencionado.

Resultados con X = A:

Unidad de maquina en base 2 (float32)	24
Unidad de maquina en base 2 (float64)	53
Unidad de maquina en base 10 (float32)	8
Unidad de maquina en base 10 (float64)	16
Cantidad de iteraciones	5
Resultado de la serie (error inherente=0, bien redondeado, y en float64)	0.2102896476313
Resultado de la serie (error inherente=0.01%, bien redondeado, y en float64)	0.210
Cota de error total (error inherente=0)	$2 \cdot 10^{-14}$
Cota de error total (error inherente=0.01%)	$1 \cdot 10^{-4}$
Termino de estabilidad	5.151492380701273
Condición de problema	0.9825725755953436

Resultados con X = A * 15

Unidad de maquina en base 2 (float32)	24
Unidad de maquina en base 2 (float64)	53
Unidad de maquina en base 10 (float32)	8
Unidad de maquina en base 10 (float64)	16
Cantidad de iteraciones	31
Resultado de la serie (error inherente=0, bien redondeado, y en float64)	1.2514552818081
Resultado de la serie (error inherente=0.01%, bien redondeado, y en float64)	1.25146
Cota de error total (error inherente=0)	$2 \cdot 10^{-14}$
Cota de error total (error inherente=0.01%)	$6 \cdot 10^{-7}$
Termino de estabilidad	23.078419084896854
Condición de problema	0.00591430388159044

Bibliografía:

- González, Hernán. Análisis Numérico primer curso[1]. Editorial: Nueva librería. Impreso en Estados Unidos 301, Ciudad Autónoma de Buenos Aires, Argentina. En enero del 2011.

- https://es.wikipedia.org/wiki/Criterio_de_Leibniz#:~:text=En%20análisis%20matemático%20el%20criterio,con%20an%20≥%200.&text=La%20inversa%20en%20general%20no%20es%20cierta.

Juan Biancuzzo 106005

Nicolas Facciano 105859

Main:

```
from numpy import float32, float64


from UnidadDeMaquina import UnidadDeMaquina

from Serie import FuncionSerie, CantidadIteraciones

from CalculoExperimentales import CalcularCPExperimental,
CalcularTEExperimental

from ErrorTotal import ErrorTotal


valorPrueba = (105859 + 106005) / 10 ** 6

errorMinimo = 10 ** (-14)

precisionDeCalculo = float64

precisionesUsadas = [float32, float64]

baseDeCalculo = 10

basesUsadas = [2, 10]

# iteracion para las bases y precisiones

def UnidadDeMaquicaSegunBasesYPrecisiones(bases, precisiones):

    for base in bases:

        print(f"Con la base {base}, tenemos las unidades de maquina: ")

        for precision in precisiones:

            unidad = UnidadDeMaquina(base, precision)

            #__name__ para que no apresca el nombre de la clase(+ bonito)

            print(f"\tCon precision {precision.__name__}: {unidad}")

        print()



# imprime los resultados de la serie segun la precision (32 o 64 bits en los
floats)

def ResultadosSegunPrecisiones(valor : float, iteracion : int, precisiones):

    for precision in precisiones:

        print(f"Con una cantidad de iteraciones {iteracion}\n\tSe logra con
{precision.__name__} de precision: ", end = "")

        resultado = FuncionSerie(valor, iteracion, precision)

        print("{0:.55f}\n".format(float64(resultado)))
```

Juan Biancuzzo 106005
Nicolas Facciano 105859

```
def CalculoDeCPYTEPorBases(valor : float, iteracion : int, presicionMayor,
presicionMenor, bases):
    # El calculo del cp solo varia segun la precision
    for precision in [presicionMayor]:
        print(f"Con la precision {precision.__name__}:")
        cp = CalcularCPExperimental(FuncionSerie, valor, iteracion,
precision)
        print(f"\tLa Condicion del problema: {cp}")
    # el calculo experimental del te es |y_d - y_s|/ mu_s |y_d|
    for base in bases:
        print(f"Con la base {base}: ")
        te = CalcularTEExperimental(FuncionSerie, valor, iteracion,
presicionMayor, presicionMenor, base)
        print(f"\tEl termino de estabilidad: {te}")


def MostrarCalculoErrorTotal(valor , errorInherente, iteracion, bases):
    for base in bases:
        print(f"Con base {base}")
        print(f"\t El error inherente relativo es : {errorInherente} ")
        error = ErrorTotal(FuncionSerie, valor, iteracion, float64, float32,
base, errorMinimo, errorInherente)
        print(f"\t El error total es : {error}\n")


def Main():
    UnidadDeMaquicaSegunBasesYPrecisiones(basesUsadas, precisionesUsadas)
    for valor in [valorPrueba, valorPrueba * 15]:


        print(f"Con valor: {valor}")


        iteracionesNecesarias = CantidadIteraciones(valor, errorMinimo,
precisionDeCalculo)
        ResultadosSegunPrecisiones(valor, iteracionesNecesarias,
precisionesUsadas)


        CalculoDeCPYTEPorBases(valor, iteracionesNecesarias, float64,
float32, basesUsadas)
        MostrarCalculoErrorTotal(valor, 0, iteracionesNecesarias,
basesUsadas)
```

Juan Biancuzzo 106005
Nicolas Facciano 105859

```
        MostrarCalculoErrorTotal(valor, 0.0001, iteracionesNecesarias,
basesUsadas)
if __name__ == "__main__":
    Main()
```

Serie:

```
def FuncionFactorial(valor:int, precision) -> float:
    resultado = 1
    for i in range(valor):
        resultado = precision( resultado * (i + 1) )
    return precision( resultado )


# la variable iteracion seria la "k" en la serie
def ValorEnIteracion(valor : float, iteracion : int, precision) -> float:
    signo = (-1) ** iteracion
    dividendo = precision( signo * ( valor ** ( 2 * iteracion + 1 ) ) )
    divisor = precision( ( 2 ** iteracion ) * FuncionFactorial(iteracion,
precision) * ( 2 * iteracion + 1 ) )

    return precision( dividendo / divisor )



# La funcion serie itera el calculo en ValorEnIteracion (sumatoria)
def FuncionSerie(valor : float, iteraciones : int, precision) -> float:
    resultado = 0
    for iteracion in range(iteraciones + 1):
        resultado += ValorEnIteracion(valor, iteracion, precision)
    return precision(resultado)

# Segun el criterio de Leibniz (a_(n+1)) = Cota de error
def CotaDeErrorEnIteraccion(valor : float, n : int, precision) -> float:
    return abs(ValorEnIteracion(valor, n + 1, precision))



def CantidadIteraciones(valor : float, errorMinimo : float, precision) ->
int:
    iteracion = 0
```

```
    while CotaDeErrorEnIteraccion(valor, iteracion, precision) >
errorMinimo:

        iteracion += 1

    return iteracion


def CondicionMinima(valor : float, iteracion : int, precision) -> bool:

    valorConNIteraciones = abs(ValorEnIteracion(valor, iteracion,
precision))

    valorConNMas1Iteraciones = abs(ValorEnIteracion(valor, iteracion + 1,
precision))

    return valorConNMas1Iteraciones < valorConNIteraciones



def CantidadMinimaIteracciones(valor : float, precision) -> int:

    iteracion = 0

    while not CondicionMinima(valor, iteracion, precision):

        iteracion += 1

    return iteracion
```

Unidad de Maquina:

```
def UnidadDeMaquina(base : int, precision) -> int:
    division = 1              #definimos div=1
    valor = 1 + division      #definimos valor=1+div
    unidad = 0                #unidad =0

    #itera hasta que no se cumpla que la precision en valor sea igual a la
           precision en valor=1
    while not precision(valor) == precision(1):
        division /= base       #div=div/base(10)
        valor = 1 + division
        unidad += 1            #unidad=unidad+1

    #devolvemos unidad que fue sumandose de a uno hasta que dio(un contador)
    return unidad
```

Calculo Experimental:

```
from UnidadDeMaquina import UnidadDeMaquina


def CPExperimentalSegunDeltaX(serieIterable, valor : float, iteraciones :
int, precision, deltaX : float) -> float:
```

Juan Biancuzzo 106005
Nicolas Facciano 105859

```
    valorIteracion = serieIterable(valor, iteraciones, precision)
    valorCorrida = serieIterable(valor * (1 + deltaX), iteraciones,
precision)


    return abs(valorCorrida - valorIteracion) / (abs(valorIteracion) *
abs(deltaX))


def CalcularCPExperimental(serieIterable, valor : float, iteraciones : int,
precision) -> float:
      # un numero suficientemente a la derecha del valor del cp
    deltaValor = 10 ** -13      decrecimiento = 10 ** -14
    calculoCP = lambda delta : CPExperimentalSegunDeltaX(serieIterable,
valor, iteraciones, precision, delta)


    CPInicial = calculoCP(deltaValor)
    deltaValor -= decrecimiento
    CPSiguiente = calculoCP(deltaValor)

    while CPInicial > CPSiguiente:
        CPInicial = CPSiguiente
        deltaValor -= decrecimiento
        CPSiguiente = calculoCP(deltaValor)
    return CPInicial


# el calculo experimental del te es |y_d - y_s|/ mu_s |y_d|
def CalcularTEExperimental(serieIterable, valor : float, iteracion : int,
precisionMayor, precisionMenor, base : int) -> float:
    valorMayor = serieIterable(valor, iteracion, precisionMayor)
    valorMenor = serieIterable(valor, iteracion, precisionMenor)
    unidad = UnidadDeMaquina(base, precisionMenor)
    return abs(valorMayor - valorMenor) / ((0.5 * (base ** -unidad)) *
abs(valorMayor))
```

Error total:

```
from CalculoExperimentales import
CalcularCPExperimental,CalcularTEExperimental
from UnidadDeMaquina import UnidadDeMaquina


def CalculoDeError(cp : float, te : float, mu : float, errorInherente :
float, errorTruncamiento : float) -> float:
```

```
    return (cp * errorInherente) + (te * mu) + errorTruncamiento


def ErrorTotal(serieIterable, valor, iteracion, precisionMayor,
precisionMenor, base, errorTruncamiento, errorInherente):
    cp = CalcularCPExperimental(serieIterable, valor, iteracion,
precisionMayor)
    te = CalcularTEExperimental(serieIterable, valor, iteracion,
precisionMayor, precisionMenor, base)


    unidad = UnidadDeMaquina(base, precisionMayor)
    mu = 0.5* (base**-unidad)


    return CalculoDeError(cp, te, mu, errorInherente, errorTruncamiento)
```

Gráficos (plots):

```
from matplotlib import pyplot
from numpy import arange, float32, float64


from CalculoExperimentales import CalcularCPExperimental,
CalcularTEExperimental
from UnidadDeMaquina import UnidadDeMaquina
from CalculoExperimentales import CPExperimentalSegunDeltaX
from Serie import CotaDeErrorEnIteraccion, CantidadMinimaIteracciones
from ErrorTotal import CalculoDeError
from Main import valorPrueba, errorMinimo, precisionDeCalculo, baseDeCalculo
from Serie import FuncionSerie, CantidadIteraciones



def MostrarTableDeValoresDeIteracion(valor : float, errorMinimo : float,
precision):
    iteracionMasBaja = 1
    iteracionMasAlta = 10
    resultados = []
    rango = range(iteracionMasBaja, iteracionMasAlta + 1)
    for i in rango:
        resultado = CotaDeErrorEnIteraccion(valor, i, precision)
        resultados.append(resultado)
    fig = pyplot.figure()
```

```
    ax = fig.add_subplot(1, 1, 1)
    ax.set_yscale('log')
    pyplot.axhline(y=errorMinimo, color='r', linestyle='-')
    pyplot.title("Calculo de cantidad de iteraciones analitico")
    pyplot.xlabel("Cantidad de iteraciones")
    pyplot.ylabel("Valor de la cota de error")
    pyplot.plot(rango, resultados)
    pyplot.grid(True)
    pyplot.show()


def MotrarTablaDeCacluloDeCP(serieIterable, valor : float, iteraciones :
int, precision):
    deltaMasAlto = 10**-8
    rango = [deltaMasAlto]
    for i in range(50):
        rango.append(rango[i] / 1.2)
    resultados = []
    for i in rango:
        resultado = CPExperimentalSegunDeltaX(serieIterable, valor,
iteraciones, precision, i)
        resultados.append(resultado)

    fig = pyplot.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.set_xscale('log')
    pyplot.title("Calculo de CP Experimental")
    pyplot.xlabel("Valores de dx/x")
    pyplot.ylabel("Valores de CP experimental")
    pyplot.plot(rango, resultados)
    pyplot.grid(True)
    pyplot.show()


# por ahora no pude hacer el plot bien
def MostrarTablaDeValoresDeFuncion(serieIterable, valor : float, iteraciones
: int, precision):
    resultados = []
```

```
    rango = range(0, iteraciones)
    for i in rango:
        resultado = serieIterable(valor, i, precision)
        resultados.append(resultado)

    pyplot.title("Valores de la serie")
    pyplot.xlabel("Numero de iteraciones")
    pyplot.ylabel("Valor de la serie")
    pyplot.plot(rango, resultados)
    pyplot.grid(True)
    pyplot.show()



def MostrarTablaDeErrorTotal(serieIterable, valor : float, iteraciones :
int, precisionMayor, precisionMenor, base, errorTruncamiento):
    resultados = []
    rango = arange(0, 0.01, 0.000001)
    cp = CalcularCPExperimental(serieIterable, valor, iteraciones,
precisionMayor)
    te = CalcularTEExperimental(serieIterable, valor, iteraciones,
precisionMayor, precisionMenor, base)
    unidad = UnidadDeMaquina(base, precisionMayor)
    mu = 0.5* (base**-unidad)
    for i in rango:
        resultado = CalculoDeError(cp, te, mu, i, errorTruncamiento)
        resultados.append(resultado)


    fig = pyplot.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.set_yscale('log')
    ax.set_xscale('log')
    pyplot.axvline(x=0, color='r', linestyle='-')
    pyplot.axvline(x=0.01 / 100, color='r', linestyle='-')
    pyplot.title("Error total variando el error inherente")
    pyplot.xlabel("Error inherente")
    pyplot.ylabel("Error total")
    pyplot.plot(rango, resultados)
    pyplot.grid(True)
```

```
    pyplot.show()


def MostrarTablaDeCantidadIteracionesSegunValor():
    resultados = []
    valorMinimo = 10 ** -4
    valorMaximo = 5
    valorSalto = 10 ** -2
    rango = arange(valorMinimo, valorMaximo, valorSalto)
    for i in rango:
        resultado = CantidadIteraciones(i, errorMinimo, precisionDeCalculo)
        resultados.append(resultado)

    pyplot.axvline(x=valorPrueba, color='r', linestyle='-')
    pyplot.axvline(x=valorPrueba * 15, color='g', linestyle='-')
    pyplot.title("Calculo de cantidad de iteraciones dependiendo el valor")
    pyplot.xlabel("Valor de x en la serie")
    pyplot.ylabel("Cantidad de iteraciones")
    pyplot.plot(rango, resultados)
    pyplot.grid(True)
    pyplot.show()


def MostrarTablaDeCantidadIteracionesMinimasSegunValor():
    resultados = []
    valorMinimo = 0.1
    rango = [valorMinimo]
    for i in range(83):
        rango.append(rango[i] * 1.05)
    for i in rango:
        resultado = CantidadMinimaIteracciones(i, precisionDeCalculo)
        resultados.append(resultado)

    fig = pyplot.figure()
    ax = fig.add_subplot(1, 1, 1)
    ax.set_xscale('log')
    pyplot.axvline(x=valorPrueba, color='r', linestyle='-')
```

```
    pyplot.axvline(x=valorPrueba * 15, color='g', linestyle='-')
    pyplot.title("Calculo de cantidad de iteraciones\n minima para cumplir el criterio de Leibniz")
    pyplot.xlabel("Valor de x en la serie")
    pyplot.ylabel("Cantidad de iteraciones minima")
    pyplot.plot(rango, resultados)
    pyplot.grid(True)
    pyplot.show()


def MostrarTablaDeCacluloDeTE(serieIterable, valor : float, iteraciones : int, precisionMayor, precisionMenor, base):
    resultados = []
    rango = arange(0.01, 5, 0.05)
    for i in rango:
        resultado = CalcularTEExperimental(serieIterable, i, iteraciones, precisionMayor, precisionMenor, base)
        resultados.append(resultado)

    pyplot.axvline(x=valorPrueba, color='r', linestyle='-')
    pyplot.axvline(x=valorPrueba * 15, color='r', linestyle='-')
    pyplot.title("Calculo del TE")
    pyplot.xlabel("Valores de X")
    pyplot.ylabel("Valores de TE")
    pyplot.plot(rango, resultados)
    pyplot.grid(True)
    pyplot.show()


def main():
    MostrarTableDeValoresDeIteracion(valorPrueba, errorMinimo, precisionDeCalculo)
    MostrarTablaDeCantidadIteracionesSegunValor()
    iteracionesNecesarias = CantidadIteraciones(valorPrueba, errorMinimo, precisionDeCalculo)
    MostrarTablaDeErrorTotal(FuncionSerie, valorPrueba, iteracionesNecesarias, float64, float32, baseDeCalculo, errorMinimo)
    MotrarTablaDeCacluloDeCP(FuncionSerie, valorPrueba, iteracionesNecesarias, precisionDeCalculo)
    MostrarTablaDeCantidadIteracionesMinimasSegunValor()
```

```
    MostrarTablaDeCacluloDeTE(FuncionSerie, valorPrueba,
iteracionesNecesarias, float64, float32, baseDeCalculo)



if __name__ == "__main__":
    main()
```

Juan Biancuzzo 106005
Nicolas Facciano 105859