Errores Numéricos

Mateo Cumbal

2024-10-28

Tabla de Contenidos

1 Ejercicio 1

Calcule los errores absoluto y relativo en las aproximaciones de p por p∗.

a) $p = \pi, p^* = \frac{22}{7}$

```
a1 = error_absoluto(math.pi, 22/7)
```

```
e abs = |3.141592653589793 - 3.142857142857143|
e abs = 0.0012644892673496777
```

```
error_relativo(a1, math.pi)
```

b) $p = \pi, p^* = 3.1416$

```
b1 = error_absoluto(math.pi, 3.1416)
```

```
error_relativo(b1, math.pi)
```

c) $p = e, p^* = 2.718$

```
c1 = error_absoluto(math.e, 2.718)
```

```
error_relativo(c1, math.e)
```

d) $p = \sqrt{2}, p^* = 1.414$

```
d1 = error_absoluto(math.sqrt(2),1.414)
```

```
error_relativo(d1,math.sqrt(2))
```

2 Ejercicio 2

Calcule los errores absoluto y relativo en las aproximaciones de p por p∗.

a) $p = e^{10}, p^* = 22000$

```
a2 = error_absoluto(math.pow(math.e, 10), 22000)
```

```
error_relativo(a2, math.pow(math.e, 10))
```

b) $p = 10^{\pi}, p^* = 1400$

```
b2 = error_absoluto(math.pow(10, math.pi), 1400)
```

```
error_relativo(b2, math.pow(10, math.pi))
```

c) $p = 8!, p^* = 39900$

```
c2 = error_absoluto(math.factorial(8), 39900)
```

```
error_relativo(c1, math.factorial(8))
```

d) $p = 9!, p^* = \sqrt{18\pi}(\frac{9}{e})^9$

```
d2 = error_absoluto(math.factorial(9), math.sqrt(18*math.pi)*math.pow(9/math.e, 9))
```

```
error_relativo(d2, math.factorial(9))
```

3 Ejercicio 3

Encuentre el intervalo más largo en el que se debe encontrar p para aproximarse a $p*$ con error relativo máximo de 10^{-4} para cada valor de p.

a) π

$\pi - 10^{-4} \cdot \pi \leq p^* \leq \pi + 10^{-4} \cdot \pi$

```
error_relativo_max(math.pi, 10**-4)
```

b) e

$e - 10^{-4} \cdot e \leq p^* \leq e + 10^{-4} \cdot e$

```
error_relativo_max(math.e, 10**-4)
```

c) $\sqrt{2}$

$\sqrt{2} - 10^{-4} \cdot \sqrt{2} \leq p^* \leq \sqrt{2} + 10^{-4} \cdot \sqrt{2}$

```
error_relativo_max(math.sqrt(2), 10 ** -4)
```

d) $\sqrt[3]{7}$

$\sqrt[3]{7} - 10^{-4} \cdot \sqrt[3]{7} \leq p^* \leq \sqrt[3]{7} + 10^{-4} \cdot \sqrt[3]{7}$

```
error_relativo_max(7 ** (1/3), 10 ** -4)
```

4 Ejercicio 4

Use la aritmética de redondeo de tres dígitos para realizar lo siguiente. Calcule los errores absoluto y relativo con el valor exacto determinado para por lo menos cinco dígitos.

a) $\frac{\frac{13}{14}-\frac{5}{7}}{2e-5.4}$

```
real1 = (13/14 - 5/7) / (2*math.e - 5.4)
redondeado1 = red(red((red(13/14) - red(5/7)))/(2*red(math.e) - 5.4))
print(f'Valor Real: {real1}')
print(f'Valor Redondeado: {redondeado1}')
```

```
abs1 = error_absoluto(real1, redondeado1)
```

```
error_relativo(abs1, real1)
```

b) $-10\pi + 6e - \frac{3}{61}$

```
real2 = -10*math.pi + 6*math.e - 3/61
redondeado2 = red(-10*red(math.pi) + 6*red(math.e) - red(3/61))
print(f'Valor Real: {real2}')
print(f'Valor Redondeado: {redondeado2}')
```

```
abs2 = error_absoluto(real2, redondeado2)
```

```
error_relativo(abs2, real2)
```

c) $(\frac{2}{9})(\frac{9}{11})$

```
real3 = (2/9) * (9/11)
redondeado3 = red(red(2/9) * red(9/11))
print(f'Valor Real: {real3}')
print(f'Valor Redondeado: {redondeado3}')
```

```
abs3 = error_absoluto(real3, redondeado3)
```

```
error_relativo(abs3, real3)
```

d) $\frac{\sqrt{13}+\sqrt{11}}{\sqrt{13}-\sqrt{11}}$

```
real4 = (math.sqrt(13) + math.sqrt(11)) / (math.sqrt(13) - math.sqrt(11))
redondeado4 = red(red(red(math.sqrt(13)) + red(math.sqrt(11))) /
                  red(red(math.sqrt(13)) - red(math.sqrt(11))))
print(f'Valor Real: {real4}')
print(f'Valor Redondeado: {redondeado4}')
```

```
abs4 = error_absoluto(real4, redondeado4)
```

```
error_relativo(abs4, real4)
```

5 Ejercicio 5

Los primeros tres términos diferentes a cero de la serie de Maclaurin para la función arcotangente son: $x-(\frac{1}{3})x^3+(\frac{1}{5})x^5$. Calcule los errores absoluto y relativo en las siguientes aproximaciones de π mediante el polinomio en lugar del arcotangente:

a) $4[arctan(\frac{1}{2})+arctan(\frac{1}{3})]$

```
a = 4 * (serie_maclaurin_reducida(1 / 2) + serie_maclaurin_reducida(1 / 3))
print(f'Valor de pi aproximado: {a}')
```

```
error1 = error_absoluto(math.pi, a)
```

```
error_relativo(error1, math.pi)
```

b) $16acrtan(\frac{1}{5})-4arctan(\frac{1}{239})$

```
b = 16 * serie_maclaurin_reducida(1 / 5) - 4 * serie_maclaurin_reducida(1 / 239)
print(f'Valor de pi aproximado: {b}')
```

```
error2 = error_absoluto(math.pi, b)
```

```
error_relativo(error2, math.pi)
```

6 Ejercicio 6

El número e se puede definir por medio de $\sum_{n=0}^{\infty}(\frac{1}{n!})$, donde $n! = n(n-1)\cdots 2$ para $n \neq 0$ y $0! = 1$. Calcule los errores absoluto y relativo en la siguiente aproximación de e

a) $\sum_{n=0}^{5}(\frac{1}{n!})$

```
e1 = serie_e(5)
print(f'Valor de e aproximado: {e1}')
```

```
ea1 = error_absoluto(math.e, e1)
```

```
error_relativo(ea1, math.e)
```

b) $\sum_{n=0}^{10}(\frac{1}{n!})$

```
e2 = serie_e(10)
print(f'Valor de e aproximado: {e2}')
```

```
ea2 = error_absoluto(math.e, e2)
```

```
error_relativo(ea2, math.e)
```

7 Ejercicio 7

Suponga que dos puntos (x_0,y_0) y (x_1,y_1) se encuentran en línea racta con $y_1 \neq y_0$. Existen dos fórmulas para encontrar la intersección x de la línea:

$x = \frac{x_0 y_1 - x_1 y_0}{y_1 - y_0}$ y $x = x_0 - \frac{(x_1 - x_0) y_0}{y_1 - y_0}$

Use los datos (x_0, y_0)=(1.31,3.24) y (x_1, y_1) = (1.93,5.76) y la aritmética de redondeo de tres dígitos para calcular la intersección con x de ambas manera ¿Cuál método es mejor y por qué?

```
aprox1 = interseccion_1((1.31,3.24), (1.93,5.76))
print(aprox1)
```

```
aprox2 = interseccion_2((1.31,3.24), (1.93,5.76))
print(aprox2)
```

> No se puede determinar cual ecuación es más precisa porque se aplica un redondeo a cada operación, quitandoles su propia precisión a cada una.

8 Códigos

8.1 Error absoluto y error relativo

Para la solución del ejercicio 1 y 2 se considerará el siguiente código:

```
import math
```

```
def error_absoluto(valor_real: float, valor_aproximado: float) -> float:
    print(f'e abs = |{valor_real} - {valor_aproximado}|')
    e_abs = abs(valor_real - valor_aproximado)
    print(f'e abs = {e_abs:.19f}\n')
    return e_abs

def error_relativo(error_absoluto: float, valor_real: float) -> None:
    print(f'e rel = {error_absoluto} / |{valor_real}|')
    e_rel = error_absoluto / abs(valor_real)
    print(f'e rel = {e_rel:.19f}')
    print(f'e rel = {e_rel*100} %')
```

8.2 Error relativo máximo

En este caso, el error relativo máximo queda determinado de la siguiente manera:

$|\frac{p^{*}-p}{p}| \leq 10^{-4}$

De donde se obtiene que:

$p - 10^{-4} \cdot p \leq p^* \leq p + 10^{-4} \cdot p$

Y queda programado como:

```
def error_relativo_max(valor_real: float, error_maximo: float) -> None:
    min, max = valor_real * (1 - (10 ** -4)), valor_real * (1 + (10 ** -4))
    print(f'{min}   p*   {max}')
```

8.3 Aritmética de redondeo de 3 dígitos

Para realizar el redondeo de 3 digitos se implementó el siguiente código:

```
def red(numero: float) -> float:
    if - 1 < numero < 1:
        return round(numero, 3)
    else:
        return round(numero, 2)
```

8.4 Aproximación de π con serie de Maclaurin

Se define la siguiente función para calcular la aproximación de π a partir de los 3 primeros términos distintos de cero de la serie de Maclaurin

```
def serie_maclaurin_reducida(x: float) -> float:
    return x - ((1 / 3) * (x ** 3)) - ((1 / 5) * (x ** 5))
```

8.5 Serie de e^x evaluada en $x = 1$ para determinar e

El código para determinar la aproximación de e se establece por:

```
def factorial(x: int) -> int:
    if x == 0 or x == 1:
        return 1
    else:
        return x * factorial(x-1)

def serie_e(n: int) -> float:
    aprox_e = 0
    for i in range(n+1):
        aprox_e += (1/factorial(i))
    return aprox_e
```

8.6 Intersección x con la línea

Las funciones para determinar las intersecciones con ambos métodos son:

```
def interseccion_1(punto1: list[float], punto2: list[float]) -> float:
    x = red((red(punto1[0]*punto2[1]) - red(punto2[0]*punto1[1])) /
            (punto2[1] - punto1[1]))
    return x
```

```
def interseccion_2(punto1: list[float], punto2: list[float]) -> float:
    x = punto1[0] - red((red((punto2[0] - punto1[0]) * punto1[1]))/
                        (punto2[1] - punto1[1]))
    return x
```

GitHub: Métodos Númericos - @mateobtw18