P4

November 23, 2024

[1]:
```
from math import pi, sqrt, copysign
import matplotlib.pyplot as plt

result = pi**4 / 90
n_digits = 20
```

0.1 Ejercicio 1

(a) Why the associative property is not fulfilled?

Porque cuando estamos sumando o restando números con coma flotante, el resultado no es exacto, ya que el número está compuesta por la base 2 y se utiliza como mucho 23 bits para representar la parte decimal. Por lo tanto, al sumar o restar números con coma flotante, mientras que ambos números no está al mismo rango, se pierde precisión y el resultado no es exacto.

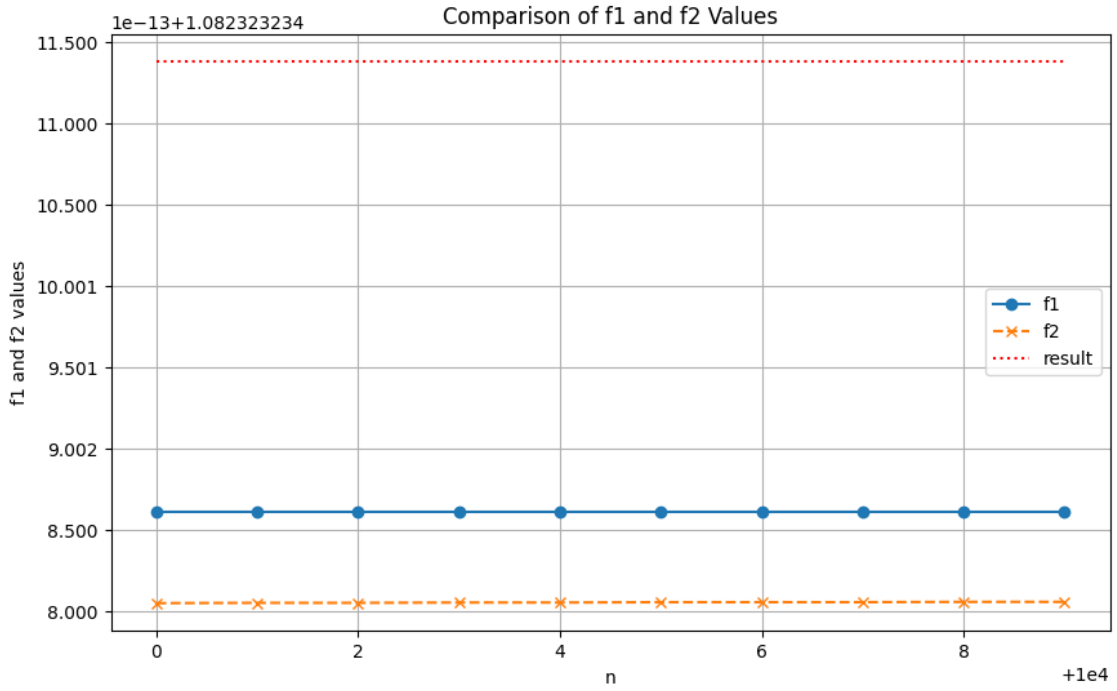
[2]:
```
# (c) Write a function called sumSeriesA with the number of terms to be taken␣
 ↪into account as theargument.
# The output of the function should be the first n terms summed in ascending␣
 ↪order.
def sumSeriesA(n):
    all_values = [1 / i**4 for i in range(1, n+1)]
    results = [all_values[0]]
    for i in range(1, n):
        results.append(all_values[i] + results[i-1])
    return results[-1]

# (d) Write a function called sumSeriesD but in this cases summing terms in␣
 ↪descending order.
def sumSeriesD(n):
    all_values = [1 / i**4 for i in range(n, 0, -1)]
    results = [all_values[0]]
    for i in range(1, n):
        results.append(all_values[i] + results[i-1])
    return results[-1]

# (e) Write a program sumSeries in python using both functions to compare the␣
 ↪results
def sumSeries(f1, f2, n, n_digits):
```

```
    results1 = f1(n)
    results2 = f2(n)
    print(f'n-{n}: f1: {results1:.{n_digits}f} vs f2: {results2:.{n_digits}f}')
    return results1, results2


def plot_sumSeries(n, n_digits, result):
    results1 = []
    results2 = []
    n_values = []
    for i in range(n, n+10):
        value1, value2 = sumSeries(sumSeriesA, sumSeriesD, i, n_digits)
        results1.append(value1)
        results2.append(value2)
        n_values.append(i)

    print(result)

    line = [result] * len(n_values)
    plt.figure(figsize=(10, 6))
    plt.plot(n_values, results1, label="f1", marker="o", linestyle="-")
    plt.plot(n_values, results2, label="f2", marker="x", linestyle="--")
    plt.plot(n_values, line, label="result", color="red", linestyle=":")
    plt.xlabel("n")
    plt.ylabel("f1 and f2 values")
    plt.title("Comparison of f1 and f2 Values")
    plt.legend()
    plt.grid()
    plt.show()
```

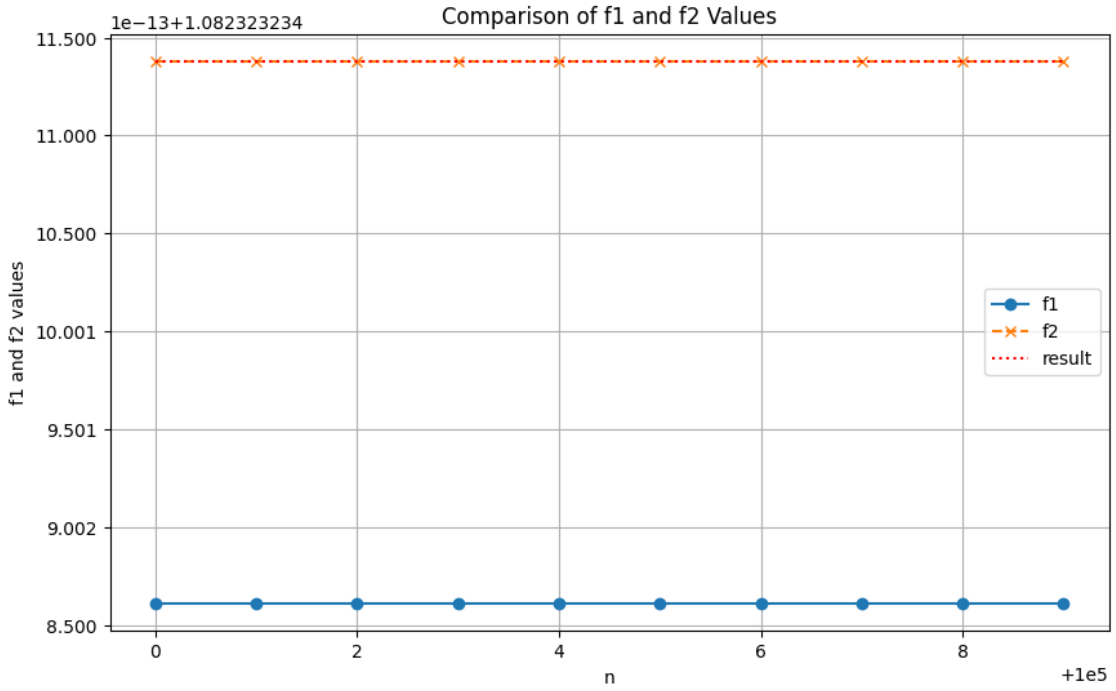
[3]:
```
result = pi**4 / 90
n_digits = 20

n = 10000
plot_sumSeries(n, n_digits, result)

n = 100000
plot_sumSeries(n, n_digits, result)
```

```
n-10000: f1: 1.08232323371086103236 vs f2: 1.08232323371080485508
n-10001: f1: 1.08232323371086103236 vs f2: 1.08232323371080507712
n-10002: f1: 1.08232323371086103236 vs f2: 1.08232323371080507712
n-10003: f1: 1.08232323371086103236 vs f2: 1.08232323371080529917
n-10004: f1: 1.08232323371086103236 vs f2: 1.08232323371080529917
n-10005: f1: 1.08232323371086103236 vs f2: 1.08232323371080552121
n-10006: f1: 1.08232323371086103236 vs f2: 1.08232323371080552121
n-10007: f1: 1.08232323371086103236 vs f2: 1.08232323371080552121
n-10008: f1: 1.08232323371086103236 vs f2: 1.08232323371080574326
```

```
n-10009: f1: 1.08232323371086103236 vs f2: 1.08232323371080574326
1.082323233711138
```




```
n-100000: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100001: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100002: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100003: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100004: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100005: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100006: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100007: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100008: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
n-100009: f1: 1.08232323371086103236 vs f2: 1.08232323371113792199
1.082323233711138
```



(b) What order is more accurate?. why?

Tras ejecutar ambos funciones con valores muy altos, se puede ver que la suma ascendente deja de ser precisa mientras que la suma descente, todavía se puede aproximar al valor real. Este se debe que en la suma descente, al ser 1 / i^4, estamos empezando a sumar con números muy pequeños, donde todos los números están al mismo rango y poco a poco se va subiendo ese rango, por lo tanto, siempre intenta mantener esa dimensionalidad, mientras que la suma ascendente, como empezamos a sumar con números muy "grandes", cuando llega a 1 / 1000^2, la representación de la parte decimal ya no dispone de bits suficientes y se pierde la suma, generando el mismo resueltado en valores muy grandes.

Entonces, la respuesta de cuál es la más precisa sería la función de suma descendente aunque para números bajos, como la suma ascendente se va acumulando los errores, en este caso se acerca más al valor real.

[4]:
```
n = 20
for i in range(n, n+10):
    sumSeries(sumSeriesA, sumSeriesD, i, n_digits)

result
```

```
n-20: f1: 1.08228458800758153835 vs f2: 1.08228458800758153835
n-21: f1: 1.08228972989804894667 vs f2: 1.08228972989804894667
n-22: f1: 1.08229399873214493510 vs f2: 1.08229399873214515715
n-23: f1: 1.08229757218992994616 vs f2: 1.08229757218992994616
n-24: f1: 1.08230058627172009977 vs f2: 1.08230058627172009977
```

```
n-25: f1: 1.08230314627172008457 vs f2: 1.08230314627172008457
n-26: f1: 1.08230533457044919565 vs f2: 1.08230533457044919565
n-27: f1: 1.08230721624687231674 vs f2: 1.08230721624687231674
n-28: f1: 1.08230884317315312337 vs f2: 1.08230884317315312337
n-29: f1: 1.08231025703836358787 vs f2: 1.08231025703836358787
```

[4]:
```
1.082323233711138
```

[5]:
```
n = 73
for i in range(n, n+10):
    sumSeries(sumSeriesA, sumSeriesD, i, n_digits)

result
```

```
n-73: f1: 1.08232239429650278772 vs f2: 1.08232239429650278772
n-74: f1: 1.08232242764475827812 vs f2: 1.08232242764475827812
n-75: f1: 1.08232245924969650019 vs f2: 1.08232245924969672224
n-76: f1: 1.08232248922376061984 vs f2: 1.08232248922376061984
n-77: f1: 1.08232251767080156490 vs f2: 1.08232251767080178695
n-78: f1: 1.08232254468683519733 vs f2: 1.08232254468683541937
n-79: f1: 1.08232257036072265777 vs f2: 1.08232257036072287981
n-80: f1: 1.08232259477478520715 vs f2: 1.08232259477478542919
n-81: f1: 1.08232261800535822793 vs f2: 1.08232261800535867202
n-82: f1: 1.08232264012329393310 vs f2: 1.08232264012329415515
```

[5]:
```
1.082323233711138
```

(f) Find out the n value from which different results are obtained. Explain why this n value is obtained.

Hemos probado diferentes valores de n para detectar a partir de qué momento, la suma ascendente deja de tener una presición aceptable. El primer caso es n = 22, donde la suma ascendente y descendente devuelve 2 valores diferentes, pero apartir de n = 23, vuelve a retornar el mismo valor. Esto se debe posiblemente de que n = 22, existe algunos decimales muy pequeños donde la parte ascendente no ha podido tener en cuenta, pero a partir de n = 23, como existe decimales más significativos, hizo que la suma ascendente autocorrigiera el error cometido en n = 22.

Luego, ya apartir de n = 74, todos los valores empieza a ser diferentes porque la suma ascendente ha llegado un punto donde no puede tener en cuenta todos los decimales ya que para n muy grandes, los decimales son muy pequeños y no se pueden representar con la precisión necesaria.

0.2 Ejercicio 2

(a) Show the equivalence between the three previous expressions

Primero demostramos siguiente expresión:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{y} \quad \frac{2c}{-b \pm \sqrt{b^2 - 4ac}}$$

Como cada función devuelve 2 valores, una con suma y otra con resta, en total se podría general 4 casos, resultado de func1 con la suma + resultado de func2 con la suma, suma + resta, resta + suma, resta + resta, pero si realizamos un cálculo matemático, o incluso sin hacerlo, la única manera de conseguir los mismos resultados es suma + resta o resta + suma, mientras que para suma + suma o resta + resta, el resultado no será el mismo. Por lo tanto si realizamos una equivalencia, obtenemos siguiente expresión:

$$(-b + \sqrt{b^2 - 4ac}) \cdot (-b - \sqrt{b^2 - 4ac}) = 4ac$$

$$b^2 - (b^2 - 4ac) = 4ac$$

Aunque no hemos terminado la demostración, se puede ver que la expresión final sería $0 = 0$, por lo tanto, se puede demostrar que el resultado de la suma de func1 será el resultado de la resta de func2 y viceversa.

Ahora sólo nos queda demostrar la func3, que es la siguiente:

$$q = -\frac{1}{2}\left(b + \text{sgn}(b)\sqrt{b^2 - 4ac}\right), \quad x_1 = \frac{q}{a}, \quad x_2 = \frac{c}{q}$$

Realmente esta demostración es más, sólo tenemos que sustituir q en x1 y x2 y ver si obtenemos la misma expresión que la func1 y func2:

$$x_1 = \frac{-\frac{1}{2}\left(b + \text{sgn}(b)\sqrt{b^2 - 4ac}\right)}{a} = \frac{-b - \text{sgn}(b)\sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{c}{-\frac{1}{2}\left(b + \text{sgn}(b)\sqrt{b^2 - 4ac}\right)} = \frac{2c}{-b - \text{sgn}(b)\sqrt{b^2 - 4ac}}$$

Tras sustituir q en x1 y x2, se puede ver que la expresión de func3 es equivalente a la de func1 y func2 pero con signo negativo, por lo tanto, se puede decir que las 3 expresiones son equivalentes, ya que desde un comienzo, ya se ha demostrado que el resultado con la suma del func1 es equivalente a la resta del func2 y viceversa, por lo que x1, que es equivalente a la resta del func1, también será equivalente a la suma del func2, y x2 que es equivalente a la resta del func2, también será equivalente a la suma del func1.

[6]:
```
# (b) Write functions quadratic1, quadratic2 y quadratic3 with parameters a, b␣
 ↪and c as arguments and
# roots using the different expressions as output.

def quadratic1(a, b, c):
    root = sqrt(b**2 - 4*a*c)
    return (-b + root) / (2 * a), (-b - root) / (2 * a)

def quadratic2(a, b, c):
    root = sqrt(b**2 - 4*a*c)
    return (2 * c) / (-b + root), (2 * c) / (-b - root)
```

```
def quadratic3(a, b, c):
    root = sqrt(b**2 - 4*a*c)
    q = -0.5 * (b + copysign(root, b))
    return q / a, c / q

def print_quadratic(a, b, c):
    print(f'quadratic1: {quadratic1(a, b, c)}')
    print(f'quadratic2: {quadratic2(a, b, c)}')
    print(f'quadratic3: {quadratic3(a, b, c)}')
    print()
```

[7]:
```
print_quadratic(1, 2, 1)
print_quadratic(1, 3, 1)
print_quadratic(1, 100, 1)
print_quadratic(10, 100, 10)
print_quadratic(1, 1000, 1)
```

```
quadratic1: (-1.0, -1.0)
quadratic2: (-1.0, -1.0)
quadratic3: (-1.0, -1.0)

quadratic1: (-0.3819660112501051, -2.618033988749895)
quadratic2: (-2.6180339887498953, -0.38196601125010515)
quadratic3: (-2.618033988749895, -0.38196601125010515)

quadratic1: (-0.010001000200048793, -99.98999899979995)
quadratic2: (-99.98999899981216, -0.010001000200050014)
quadratic3: (-99.98999899979995, -0.010001000200050014)

quadratic1: (-0.10102051443364388, -9.898979485566356)
quadratic2: (-9.898979485566349, -0.10102051443364381)
quadratic3: (-9.898979485566356, -0.10102051443364381)

quadratic1: (-0.0010000010000226212, -999.998999999)
quadratic2: (-999.9989999783788, -0.001000001000002)
quadratic3: (-999.998999999, -0.001000001000002)
```

(c) Find out and example where the results are different. Explain the result.

Antes de entrar con ejemplos con diferentes resultados usando |ac| « b^2, simplemente poniendo a = 1, b = 3 y c = 1 ya genera unos resultados donde uno de los resultados de la función tiene más decimales que el otro, esto del momento no nos garantiza nada y vamos a ver ejemplos con valores más grandes.

Cuando usamos ejemplos estilo b = 1000 con valores a y c muy bajos como a = c = 1, se puede ver de nuevo que los resultados realmente son muy similares, pero no exactamente iguales, donde la diferencia es mínima con algunas decimales extras. También podemos ver un efecto bastante interesante donde el resultado de la resta del func1 es exactamente igual que el x1 del func3 y el

resultado de la resta del func2 es exactamente igual que el x2 del func3, este fenómeno ocurre con cualquier tipo de valores a, b y c.

(d) Which is the correct result?. Why?

Realizando una comparativa de los datos, se concluye que la func3 es la función correcta, porque tanto a nivel de ejecución como a nivel de demostración matemática, se puede demostrar que x1 del func3 es equivalente a la resta del func1 que a su vez es equivalente a la suma del func2(con menos exactitud), y x2 del func3 es equivalente a la resta del func2 que a su vez es equivalente a la suma del func1(con menos exactitud).

El motivo de por qué func1 y func2 no ha podido tener un resultado exacto es por el tema del rango de los números, si nos fijamos un poco más sobre los resultados no exactos, siempre ocurre cuando quisiesemos realizar una suma del -b + la raiz, esto es porque cuando b es un número muy grande y además |ac| « b^2, la raiz del b^2 - 4ac es practicamente igual a b, por lo que -b + b tiene un rango de valores muy grande, por ejemplo, si b = 100000, estaríamos sumando -100000 + 100000, donde la representación de ambos valores en potencia de 2 es totalmente diferente por los signos, y por lo tanto, empezamos a perder la precisión. Mientras por otra lado, si realizamos una resta, -b - b está en la misma escala de dimensiones, por lo que la representación de ambos valores en potencia de 2 es muy similar y por lo tanto, no perdemos precisión y el resultado será más exacto. Además, como se ha demostrado matemáticamente que ambos resultados del func1, func2 y func3 son equivalentes, se debe utilizar la func3 para que todas las operaciones realizen en el mismo rango.