

Tarea 1 Física Numérica

Isaias Garcia Lopez

26 de Septiembre del 2025

Punto 1: Límites de Underflow y Overflow en Python

Contexto

En computación, los números en punto flotante tienen límites finitos debido a la representación en memoria finita. El **underflow** ocurre cuando un número es demasiado pequeño y se redondea a cero, mientras que el **overflow** ocurre cuando un número es demasiado grande y se representa como infinito. En Python, los números float siguen el estándar IEEE 754 de 64 bits.

Método Numérico

Se utilizó un enfoque iterativo para encontrar los límites:

- Para el **overflow**: Se parte de 1.0 y se multiplica por 2 repetidamente hasta que el resultado sea `inf`. El último valor finito es el límite de overflow dentro de un factor de 2.
- Para el **underflow**: Se parte de 1.0 y se divide por 2 repetidamente hasta que el resultado sea 0.0. El último valor no nulo es el límite de underflow dentro de un factor de 2.

Resultados

Los límites encontrados experimentalmente en esta máquina son:

- Límite de overflow: `8.988e+307`
- Límite de underflow: `5e-324`

Discusión

Este método es simple y efectivo, pero depende de la arquitectura y la implementación de Python. Los valores pueden variar ligeramente entre sistemas, pero coinciden con los límites teóricos del estándar IEEE 754.

Punto 2: Determinación de la Precisión de Máquina

Contexto

La **precisión de máquina** (denotada como ϵ_m) es el número positivo más pequeño tal que, en aritmética de punto flotante, se cumple:

$$1.0 + \epsilon_m \neq 1.0$$

Este valor representa el error de redondeo inherente en las operaciones numéricas y depende del estándar de punto flotante utilizado (IEEE 754 para precisión doble en Python).

Método Numérico

Se implementó un algoritmo iterativo que comienza con $\epsilon = 1.0$ y lo divide sucesivamente por 2 hasta que la condición $1.0 + \epsilon = 1.0$ se cumpla. El último valor que aún producía una diferencia detectable es la precisión de máquina.

```
def precision_maquina():
    epsilon = 1.0
    while 1.0 + epsilon != 1.0:
        epsilon_prev = epsilon
        epsilon /= 2.0
    return epsilon_prev
```

Resultados

La precisión de máquina encontrada experimentalmente es:

$$\epsilon_m = 2.220446049250313 \times 10^{-16}$$

Este valor coincide con el teórico para precisión doble (2.22×10^{-16}).

Discusión

El método utilizado es simple pero efectivo, determinando ϵ_m dentro de un factor de 2 como se solicita.

Punto 3(a): Cálculo de $\sin(x)$ mediante series de Taylor

Contexto y Objetivo

El objetivo es calcular la función seno mediante su desarrollo en serie de Taylor:

$$\sin(x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!}$$

con un error absoluto menor a 10^{-8} , evitando el cálculo inefficiente de factoriales mediante una relación de recurrencia.

Deducción de la Relación de Recurrencia

Sea el término general de la serie:

$$t_n = (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!}$$

El término siguiente es:

$$t_{n+1} = (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

La relación entre términos consecutivos es:

$$\frac{t_{n+1}}{t_n} = \frac{(-1)^n \frac{x^{2n+1}}{(2n+1)!}}{(-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!}} = \frac{(-1)^n}{(-1)^{n-1}} \cdot \frac{x^{2n+1}}{x^{2n-1}} \cdot \frac{(2n-1)!}{(2n+1)!}$$

Simplificando:

$$\frac{t_{n+1}}{t_n} = (-1) \cdot x^2 \cdot \frac{1}{(2n)(2n+1)}$$

Obtenemos la relación de recurrencia:

$$t_{n+1} = -t_n \cdot \frac{x^2}{(2n)(2n+1)}$$

Comparación entre Error Absoluto y Relativo

El error absoluto se define como:

$$E_{abs} = |S_n - \sin(x)|$$

donde S_n es la suma parcial después de n términos.

El error relativo se define como:

$$E_{rel} = \frac{|S_n - \sin(x)|}{|\sin(x)|}, \quad \text{si } \sin(x) \neq 0$$

El criterio de parada se basa en el error absoluto ($< 10^{-8}$), pero el error relativo proporciona información adicional sobre la precisión relativa al tamaño del valor verdadero.

Resultados para $x = 0.5$

N	Suma	Error Absoluto	Error Relativo	$ t_n $
1	0.5000000000000000	2.05e-02	4.27e-02	5.00e-01
2	0.4791666666666667	2.59e-04	5.40e-04	2.08e-02
3	0.4794270833333333	1.54e-06	3.22e-06	2.60e-04
4	0.479425538604167	5.39e-09	1.12e-08	1.55e-06
5	0.479425538604203	1.11e-15	2.31e-15	5.39e-09

Análisis de Resultados

Para $x = 0.5$, se observa que:

- El error absoluto y relativo tienen magnitudes similares cuando $\sin(x) \approx 1$
- El criterio de parada basado en $|t_n| < 10^{-8}$ garantiza error absoluto $< 10^{-8}$
- Solo se necesitan 4 iteraciones para alcanzar la precisión requerida
- La relación de recurrencia hace el cálculo eficiente y evita problemas numéricos

El método es robusto y eficiente para el cálculo de $\sin(x)$ con alta precisión.

Punto 3(b): Cálculo de $\sin(x)$ para valores grandes usando periodicidad

Problema con valores grandes de x

Para valores de $x > 2\pi$, la serie de Taylor converge lentamente y requiere muchos términos. Además, pueden aparecer errores numéricos debido a la cancelación catastrófica cuando se suman términos grandes con signos alternos.

Solución mediante periodicidad

Se utiliza la propiedad de periodicidad de la función seno:

$$\sin(x + 2k\pi) = \sin(x) \quad \text{para cualquier entero } k$$

El algoritmo reduce x al rango $[0, 2\pi]$ mediante:

$$x_{\text{reducido}} = x \mod 2\pi$$

Luego calcula $\sin(x_{\text{reducido}})$ usando la serie de Taylor, que converge rápidamente para valores pequeños.

Implementación en Python

```
x_reducido = x % (2 * math.pi)
```

Nos asegura la periodicidad, solo colocamos dicho valor dentro de la definición de nuestra función para calcular el seno con la relación de recurrencia de Taylor

Resultados comparativos

Table 1: Comparación con y sin periodicidad

Directo	$x_{reducido}$	Real	Con periodicidad	Sin periodicidad	Error Periodic	Error
10.0	3.716815	-0.5440211109	-0.5440211109	-0.5440211107	1.14e-11	1.58e-10
20.0	1.150444	0.9129452507	0.9129452507	0.9129452532	6.23e-12	2.45e-09
30.0	4.867259	-0.9880316241	-0.9880316238	-0.9879731902	3.21e-10	5.84e-05
40.0	2.300888	0.7451131605	0.7451131605	-1.8155099229	6.10e-11	2.56e+00
50.0	6.017703	-0.2623748537	-0.2623748535	-39110.4584803935	1.71e-10	3.91e+04
100.0	5.752220	-0.5063656411	-0.5063656411	1.178e+26	4.23e-11	1.18e+26
200.0	5.221255	-0.8732972972	-0.8732972973	2.259e+69	-	2.26e+69

De modo que implementar la periodicidad nos da una convergencia rápida al valor real del calculo, además de que, debido a esto, se reduce significativamente el número de iteraciones y los números con los que se trabajan son de una magnitud más moderada.

Punto 3(c): Tolerancia menor que la precisión de máquina

Objetivo

Investigar cómo afecta al cálculo de $\sin(x)$ establecer una tolerancia menor que la precisión de máquina ϵ_m .

Precisión de Máquina

Del punto 2, se determinó que la precisión de máquina es:

$$\epsilon_m \approx 2.22 \times 10^{-16}$$

Para este experimento, se establece una tolerancia de:

$$\text{tolerancia} = 2.22 \times 10^{-17} = \frac{\epsilon_m}{10}$$

Cambio en el Algoritmo

El único cambio requerido en el algoritmo del punto 3(a) es modificar el criterio de parada:

```
# En lugar de:  
tolerancia = 1e-8  
  
# Se usa:  
tolerancia = 2.22e-17 # /10
```

El resto del algoritmo permanece idéntico.

Comportamiento Esperado

Para valores pequeños de x

- El algoritmo continuará iterando más allá del punto óptimo
- Los términos de la serie t_n eventualmente serán menores que ϵ_m
- Cuando $|t_n| < \epsilon_m$, al sumarlo a la acumulada, no habrá cambio debido a redondeo
- Se alcanzará un punto donde $t_{n+1} = t_n$ numéricamente (estancamiento)
- El error no mejorará a pesar de más iteraciones

Para valores moderados de x

- Se necesitarán más iteraciones para alcanzar la tolerancia artificial
- Posible aparición de errores de redondeo acumulados
- El error final puede ser mayor que con tolerancia razonable

Para valores grandes de x (usando periodicidad)

- El comportamiento será similar al caso de x pequeños
- La precisión final estará limitada por ϵ_m , no por la tolerancia
- Iteraciones adicionales no mejoran la precisión debido a límites numéricos

Interpretación Física

Establecer una tolerancia menor que ϵ_m es físicamente imposible de alcanzar en aritmética de punto flotante. La computadora no puede distinguir entre:

$$1.0 + \epsilon \quad \text{y} \quad 1.0 \quad \text{para } \epsilon < \epsilon_m$$

Análogamente, en el cálculo de la serie, cuando los términos son menores que ϵ_m veces la suma acumulada, su contribución se pierde por redondeo.

Conclusión

El experimento demostró que existe un límite fundamental a la precisión alcanzable, por lo que establecer tolerancias más allá de ϵ_m es inútil, pues se observó que los valores de aproximación eran idénticos a los obtenidos por la función `math.sin(x)`, lo cual también se observó al caer los errores relativo y absolutos a 0.00 para términos más allá de la precisión de la máquina, sin embargo, para términos para términos mayores a la tolerancia no se observa algún cambio significativo.

La precisión de máquina ϵ_m representa el límite absoluto de precisión en cálculos numéricos con aritmética de punto flotante.