

ÁLGEBRA LINEAL COMPUTACIONAL

2do Cuatrimestre 2025

Laboratorio N° 1: Números de máquina.

1 Introducción

En este laboratorio exploraremos el efecto de aproximar números reales. Tengan a mano el conversor [IEEE 754 precisión simple](#) para ganar intuición de lo que ocurre cuando lo necesiten.

Ejercicio 1 (Una inocente suma).

- Comparen el resultado de hacer $0.3 + 0.25$ con el de hacer $0.3 - 0.25$. ¿En ambos casos obtienen el resultado esperado? ¿Por qué?
- Escriban el número 0.25 en base 2. ¿Cómo queda expresado en términos de su mantisa y exponente?
- Escriban el número 0.3 en base 2. ¿Qué dificultades aparecen al escribir 0.3 en binario? ¿Se puede escribir exactamente con una mantisa finita?

Ejercicio 2 (No tan distintos). En este punto exploraremos expresiones que son aparentemente iguales.

- ¿Cuánto da $(\sqrt{2})^2 - 2$?
Simbólicamente sabemos que el resultado es 0, pero ¿qué ocurre en python? Importen la librería `numpy` (`import numpy as np`) para emplear la función `np.sqrt` y calculen `np.sqrt(2)**2-2`
- Para 100 valores de equiespaciados x en el intervalo de $[0, 5 \times 10^{-8}]$, evaluar las siguientes 2 expresiones que son matemáticamente equivalentes (*pruebenlo*) y graficarlas usando `matplotlib.pyplot`. En base al gráfico obtenido identificar la opción que mejor resiste la pérdida de valores significativos.

a. $y = \sqrt{2x^2 + 1} - 1$

b. $y = \frac{2x^2}{\sqrt{2x^2+1}+1}$

Ejercicio 3 (Acumulación del error). Calculen algebraicamente el límite cuando $n \rightarrow \infty$ de esta sucesión

$$x_1 = \sqrt{2}$$
$$x_{n+1} = \frac{x_n \cdot x_n}{\sqrt{2}}$$

Implementen una rutina que calcule el valor de x_i para $i = 1, \dots, 100$ y grafiquen sus valores. ¿En qué punto se desestabiliza la sucesión? Tip: pueden guardar los elementos de la sucesión en una lista `l=[]` usando `l.append(xi)` dentro de un loop `for`. Luego importar `matplotlib.pyplot` as `plt` y graficarlos con `plt.plot(l)`.

Ejercicio 4 (Series). Comparen el resultado de calcular

$$\sum_{i=1}^{10^n} \frac{1}{i} \quad y \quad \sum_{i=1}^{5 \cdot 10^n} \frac{1}{i}$$

para $n = 6, 7$, usando precisión de 64 y 32 bits. Para eso, aprovechen la siguiente pieza de código:

```
1 import numpy as np
2
3 n = 7
4 s = np.float32(0)
5 for i in range(1, 10**n+1):
6     s = s + np.float32(1/i)
7 print("suma = ", s)
8
9 s = np.float32(0)
10 for i in range(1, 5*10**n+1):
11     s = s + np.float32(1/i)
12 print("suma = ", s)
```

Para pensar:

- ¿Cuánto vale $1/i$ en precisión simple cuando $i = 2 \cdot 10^7$?
- Si escribimos $1/10^7$ usando el mismo exponente que el necesario para representar a $\sum_{i=1}^{5 \cdot 10^6} 1/i$, ¿a cuánto equivale $1/i$?
- ¿Por qué la siguiente modificación cambia el resultado?

```
1 s = np.float32(0)
2 for i in range(2*10**n, 0, -1):
3     s = s + np.float32(1/i)
4 print("suma = ", s)
```

Extra: Usen lo aprendido para estimar el número de Euler e mediante la serie

$$e \approx \sum_{n=0}^{10} \frac{1}{n!}.$$

Comparen con el valor provisto por `numpy`, `numpy.e`

Ejercicio 5 (Arrastre de error: Descomposición LU). Desarrollar una función `matricesIguales(A, B)` que devuelve `True` si ambas matrices son iguales.

La siguiente matriz A

$$A = \begin{pmatrix} 4. & 2. & 1. \\ 2. & 7. & 9. \\ 0. & 5. & \frac{22}{3} \end{pmatrix}$$

tienen una descomposición LU siguiente:

$$L = \begin{pmatrix} 1. & 0. & 0. \\ 0.5 & 1. & 0. \\ 0. & \frac{5}{6} & 1. \end{pmatrix}, \quad U = \begin{pmatrix} 4. & 2. & 1. \\ 0. & 6. & 8.5 \\ 0. & 0. & 0.25 \end{pmatrix}$$

Verificar que la función desarrollada `matricesIguales(A, L@U)` devuelva `True`.

Ejercicio 6 (Arrastre de error: función `esSimetrica`). Verificar que devuelve la función `esSimetrica()` desarrollada para el laboratorio pasado en los siguientes casos. Para una $A = \text{np.array}(\text{np.random.rand}(4,4))$

- `esSimetrica(A.T@A)`
- `esSimetrica(A.T@ $\frac{(A*0.25)}{0.25}$)`
- `esSimetrica(A.T@ $\frac{(A*0.2)}{0.2}$)`

Ejercicios extra

Ejercicio 7. *Ángulos mínimos* Haciendo uso de los epsilon de máquina ¹ de cada tipo de variable, estimar el error en el ángulo que pueden formar el vector E_1 (canónico con todos ceros y un 1 en la primera posición, $E_1 = (1, 0, \dots, 0)$) y el vector de norma 1 $v_\gamma = \sqrt{1 - (n-1)\gamma^2} E_1 + \gamma \sum_{i=2}^n E_i$ (E_i el canónico que tiene un 1 en la posición i). Para calcular el ángulo formado entre dos vectores a y b , pueden usar la expresión:

$$\cos(\theta) = \frac{a^T b}{\sqrt{a^T a} \sqrt{b^T b}}$$

con θ el ángulo que forman a y b . Explorar:

- Fijando $\gamma = \epsilon$ y variando $n = 1, \dots, 1000$
- Fijando $n = 2$ y variando $\gamma = k\epsilon$ con $k = 1, \dots, 1000$

Comparar en todos los casos con el ángulo exacto. *Tip: el ángulo exacto puede calcularse como*

$$\tan \theta = \sqrt{\frac{(n-1)\gamma^2}{1 - (n-1)\gamma^2}}$$

Ejercicio 8. *Eso no es la identidad* Partiendo de la matriz

$$A = \begin{pmatrix} 10^{-1} & 1 \\ 0 & 1 \end{pmatrix}$$

- Calcular analíticamente A^{-1} .
- Comprobar que $AA^{-1} = I$, la matriz identidad.
- Calcular $A^n(A^{-1})^n$ para valores de $n = 1, \dots, 100$. ¿Qué problemas de estabilidad aparecen?

Módulo ALC

Para el módulo ALC, deben programar:

¹El epsilon de cada tipo de variable puede obtenerse con `numpy.finfo(x).eps` para x en `['float16', 'float32', 'float64']`

```

1 def error(x,y):
2     """
3     Recibe dos numeros x e y, y calcula el error de aproximar x usando y en float64
4     """
5
6 def error_relativo(x,y):
7     """
8     Recibe dos numeros x e y, y calcula el error relativo de aproximar x usando y en float64
9     """
10
11 def matricesIguales(A,B):
12     """
13     Devuelve True si ambas matrices son iguales y False en otro caso.
14     Considerar que las matrices pueden tener distintas dimensiones, ademas de distintos valores.
15     """

```

Se aportan una serie de tests utilizando la función `assert`.

```

1 def sonIguales(x,y,atol=1e-08):
2     return np.allclose(error(x,y),0,atol=atol)
3
4 assert(not sonIguales(1,1.1))
5 assert(sonIguales(1,1 + np.finfo('float64').eps))
6 assert(not sonIguales(1,1 + np.finfo('float32').eps))
7 assert(not sonIguales(np.float16(1),np.float16(1) + np.finfo('float32').eps))
8 assert(sonIguales(np.float16(1),np.float16(1) + np.finfo('float16').eps,atol=1e-3))
9
10 assert(np.allclose(error_relativo(1,1.1),0.1))
11 assert(np.allclose(error_relativo(2,1),0.5))
12 assert(np.allclose(error_relativo(-1,-1),0))
13 assert(np.allclose(error_relativo(1,-1),2))
14
15 assert(matricesIguales(np.diag([1,1]),np.eye(2)))
16 assert(matricesIguales(np.linalg.inv(np.array([[1,2],[3,4]]))@np.array([[1,2],[3,4]]),np.eye(2)))
17 assert(not matricesIguales(np.array([[1,2],[3,4]]).T,np.array([[1,2],[3,4]])))

```