Ejercicio 1

(Representación de Números, Errores, Aritmética Punto Flotante, Convergencia)

Entrega hasta el **12 de septiembre** a las 23:59 en U-Campus. $nota(x) = \min(7, \max(1, 7x/13))$
Los ejercicios son **trabajos individuales**. Plagio en alguna parte de la tarea resulta en nota 1.0 para todo el ejercicio y se reportara a la escuela. Cada programa entregado debe contener la **información del autor** en forma de dos variables __autor__ y __RUT__.

P1 (4 puntos) Escribir un programa Python que permite dibujar un conjunto de polinomios en un intervalo dado (por ejemplo *Fig. 1*). Al dibujar un polinomio hay que evaluarlo y queremos usar aritmética anidada. Los polinomios a dibujar se definen en las primeras lineas del programa de la siguiente forma:

```python
import ...

__author__ = "NOMBRE"
__RUT__ = "RUT-SIN-PUNTOS-CON-GUIÓN"

# 2x^3 - 6x^2 + 2x - 1
p1 = [2, -6, 2, -1]
# 2x^3 + 3x + 1
p2 = [2, 0, 3, 1]
# -x^4
p3 = [-1, 0, 0, 0, 0]
# 10
p4 = [10]

polynomials = p1, p2, p3, p4
interval = np.linspace(-3, 3, 100)

# AQUÍ IMPLEMENTAR EL CÓDIGO
```

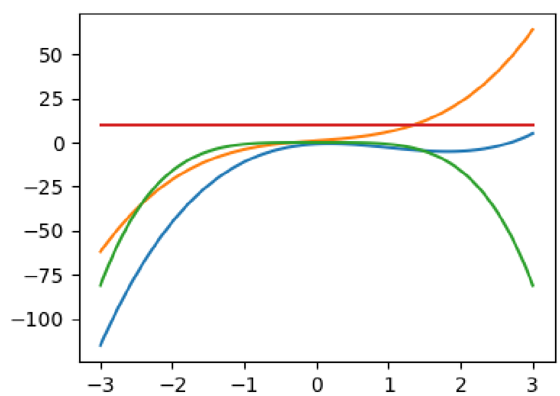

Fig. 1: Ejemplo de 4 polinomios en el intervalo $[-3, 3]$.

El programa debe ser capaz de evaluar cualquier conjunto de polinomios que quedan dentro de los limites del sistema punto flotante de Python (IEEE 754). El proposito del ejercicio es **implementar la aritmética anidada**. (No se debe usar NumPy u otra libraría para evaluar los polinomios.)

Pista: Implementar una función que permite evaluar un polinomio en un punto dado. Hay varias posibilidades para implementar la aritmética anidada. Por ejemplo, se puede utilizar un bucle o la función `reduce`.

Punto extra: Generar e imprimir una leyenda a partir de los polinomios dados.

P2 (3 puntos) Considerar la función

$$f(x) = 1.01e^{4x} - 4.62e^{3x} - 3.11e^{2x} + 12.2e^{x} - 1.99.$$

a) Usar aritmética de punto flotante de tres dígitos con redondeo para evaluar $f(1.53)$. Se supone que $e^{1.53} = 4.62$. Utilice multiplicación repetida para calcular $e^{nx} = (e^x)^n$.

b) Muestre que la técnica de la aritmética anidada también se puede aplicar a la evaluación de la función $f(x)$. Es decir, mostrar una forma anidada para $f(x)$.

c) Evaluar la forma anidada de la parte b) con el mismo sistema de punto flotante que en la parte a) y comparar los resultados con el valor verdadero de tres dígitos $f(1.53) = -7.61$.

P3 (2 puntos) Normalmente, en un sistema de punto flotante, el crecimiento lineal del error es inevitable. Lo que podemos intentar es reducir la cantidad de cálculos necesarios en un algoritmo. Consideramos la siguiente suma:

$$\sum_{i=1}^{n} \sum_{j=1}^{i} a_i b_j.$$

a) ¿Cuántas multiplicaciones y adiciones se requieren para calcular la suma?

b) Entregar una suma equivalente que reduzca el número de cálculos. ¿Cuantos son?

P4 (4 puntos) Consideramos la siguiente suma de términos:

$$\sum_{i=0}^{n} \frac{2^i x^{2^i-1} - 2^{i+1} x^{2^{i+1}-1}}{1 - x^{2^i} + x^{2^{i+1}}} = \frac{1-2x}{1-x+x^2} + \frac{2x-4x^3}{1-x^2+x^4} + \frac{4x^3-8x^7}{1-x^4+x^8} + \dots$$

En el intervalo $(-1, 1)$ la suma converge a $f(x) = \dfrac{1+2x}{1+x+x^2}$. Queremos visualizar que tan rápido converge la suma en este intervalo, es decir, para los valores $-1 < x < 1$. Escribir un programa Python que permite visualizar el **número de términos necesarios** para obtener una aproximación con error absoluto no mayor que un valor ε dado (por ejemplo *Fig. 2*).

```python
import ...

__author__ = "NOMBRE"
__RUT__ = "RUT-SIN-PUNTOS-CON-GUIÓN"

epsilon = 1e-8
interval = np.linspace(-1, 1, 102)[1:-1]

def f(x):
    return (1 + 2*x) / (1 + x + x**2)

# AQUÍ IMPLEMENTAR EL CÓDIGO
```

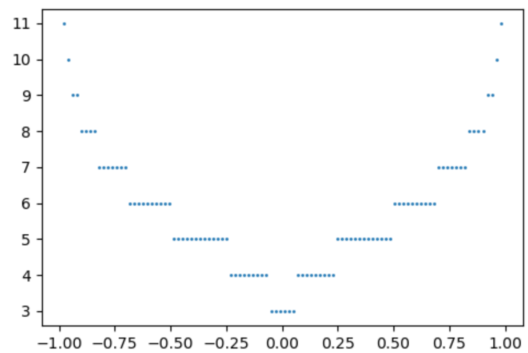

Fig. 2: Número de términos necesarios para $\varepsilon = 10^{-8}$.

Pista: Crear una función `termino(i, x)` para evaluar el i-ésimo término en un punto x dado.

Punto extra: ¿Que pasa en $x = 1$? Explíquelo.

Hay 15 puntos en total (13 + 2 extra). Para obtener un 7.0 se necesita 13 puntos.