Tarea 1: Métodos Numéricos

Rafael Alejandro García Ramírez

5 de septiembre de 2023

1. Como es bien sabido, la representación numérica binaria de tipo signo/exponente/mantisa, depende de la creatividad o precisión que desee el programador. Analicemos qué sucede con un sistema de numeración binario con n ¿0 (entero) bits. Para esto, elige un valor par entre 10 y 20, crea el sistema de numeración entero similar al estudiado en clase (la mantisa debe ser, al menos, la mitad del número elegido). Describe tu sistema de numeración. ¿Cuál es el menor y mayor de los números representados? Presenta un número que no se pueda representar en tu sistema seleccionado. Describe tu proceso. **(4 pts)** NOTA: punto extra si además se describe un sistema fraccionario que tenga sentido.

   Propondremos un sistema de representación de 16 bits con 1 bit de signo, 6 de exponente y 9 de mantisa. Ahora haremos la representación del número 105. Pasando el numero a binario teneos $105_{10} = 1101001_2$. Dado que no se necesita un valor exponencial para tomar este valor (aún queda espacio) y el signo es positivo, tenemos el valor:

Signo	Exponente						Mantisa								
0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1

   Ahora construyamos un número que requiera el uso del exponente, es decir en forma decimal. Proponemos el valor $662_{10} = 1010010110_2$ lo cuál utilizando la forma exponencial sería $662_{10} = 2^1 \times 101001011{,}0_2$ lo cual en forma de exponente toma la siguiente forma

Signo	Exponente						Mantisa								
0	0	0	0	0	0	1	1	0	1	0	0	1	0	1	1

   Ahora buscaremos el mayor número representable en este sistema. Este será el número 111111111111111_2

Signo	Exponente						Mantisa								
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

   Este número es, con un exponente 6 tenemos $2^{6-1} - 1 = 31$, así con la mantisa esto es $111111111 \times 2^{31} = 11111111100000000000000000000000000000000 = 2^{32}+2^{33}+2^{34}+2^{35}+2^{36}+2^{37}+2^{38}++2^{39}+2^{40} = 2,194,728,288,256$ que es nuestro máximo valor. Para nuestro menor valor, dado que reservamos un bit para el signo, tenemos que este es

Signo	Exponente						Mantisa								
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

   El cuál corresponde al valor $-2,194,728,288,256$. Para encontrar un valor que, cómo se hizo anteriormente para la parte decimal, no se pueda representar, buscaremos el máximo valor de la mantisa que sobrepase los 9 dígitos, usaremos entonces el número $1110110111_2 = 951_{10}$, así con la forma exponencial tenemos $951_{10} = 111011011{,}1_2 \times 2^1$, este número es

Signo	Exponente						Mantisa								
0	0	0	0	0	0	1	1	1	1	0	1	1	0	1	1

   Que corresponde al número 475, es decir que el número 951 no es representable, si incluimos el exponente (es decir, el 1 que nos falta lo pasamos a la siguiente posición) tenemos el valor $0000001111011011_2 = 987_{10}$.

2. Consideremos la siguiente sucesión $\{x_n\}_{n\in\mathbb{N}} \subset \mathbb{R}$ definida de manera recurrente como: **(4 pts)**

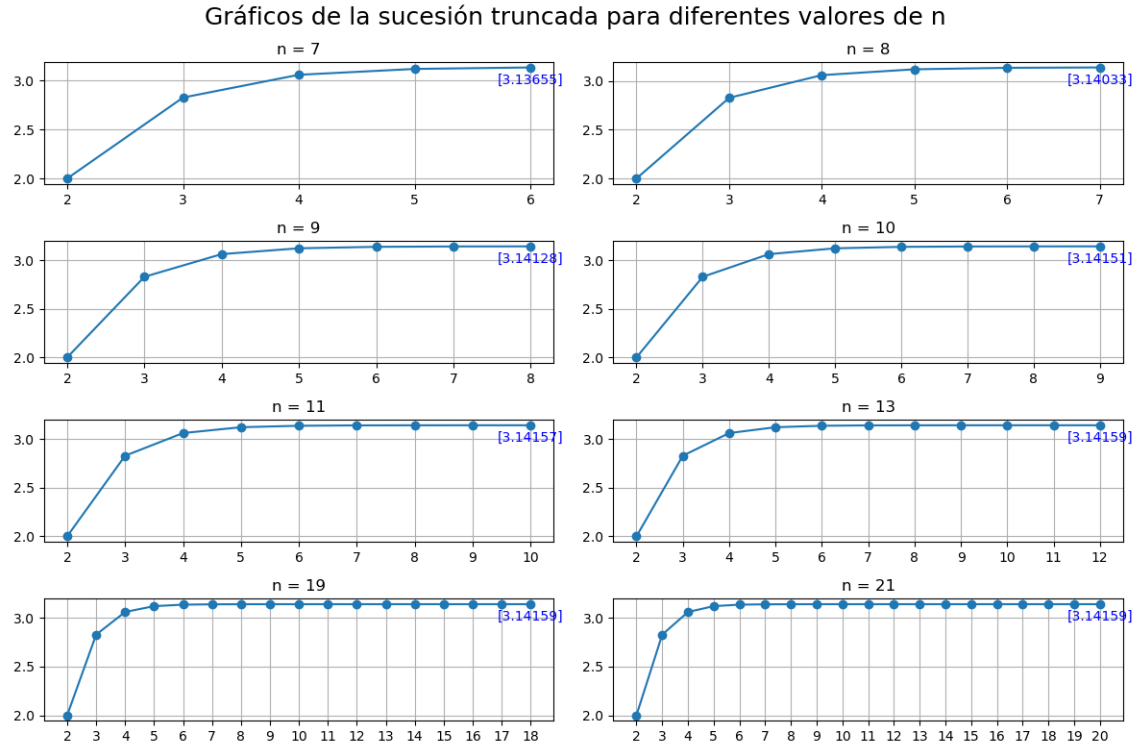
$$x_2 = 2$$
$$x_{n+1} = 2^{n-\frac{1}{2}}\sqrt{1-\sqrt{1-4^{1-n}x_n^2}}$$

- Escribe un programa para encontrar los primeros n términos de la sucesión.
- Aplica tu programa para encontrar los términos de la sucesión cuando n = 6; 7; 8; 9; 10; 12; 18; 20. ¿A qué valor tiende la sucesión? (Esta sucesión tiende a un valor distinto de cero).
- Encuentra los términos 50 y 100. ¿Qué sucede con la sucesión? Gráfica los términos de la sucesión desde 2 hasta 100. Explica detalladamente el por qué del comportamiento que tiene la computadora al calcular estos términos.

Creamos una función para calcular los valores de la sucesión, y después realizamos un multigráficos para visualizar la convergencia de cada valor. Para cada uno de las secciones del siguiente código de este problema se utilizarán las paqueterías **numpy** y **matplotlib.pyplot**.

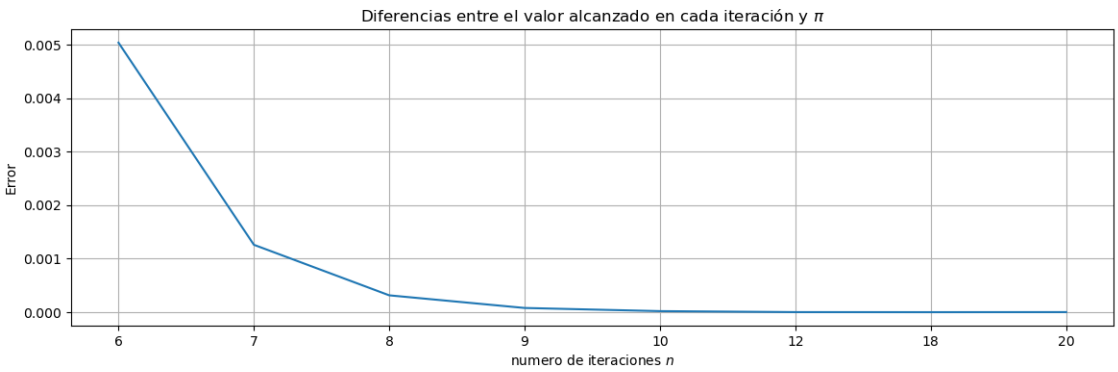
```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Escribe un porgrama para encontrar los primeros n terminos de la sucesion

def sucesion(n):
    """
    Genera una sucesion de numeros basada en la formula dada.

    Args:
    n (int): Numero de terminos de la sucesion a generar.

    Returns:
    list: Una lista de `n` terminos de la sucesion calculados segun la formula.

    Formula:
    El termino general de la sucesion esta definido por:
    x_2 = 2,
    x_k+1 = 2**(k - 0.5) * (1 - (1 - 4**(1 - k) * (x_(k)**2))**0.5) ** 0.5

    Donde:
    - k (int): Indice del termino actual.
    - x_k (float): Valor del termino actual.
    - x_(k-1) (float): Valor del termino anterior.

    Ejemplo:
    >>> sucesion(5)
    [2, 2.8284271247461903, 3.0614674589207187, 3.121445152258053]
    """
    data = [2] # x_2
    for elemento in range(2,n):
        data.append(2**(elemento - 0.5)*((1 - (1 - (4**(1-elemento))*(data[-1]**2))**0.5)**0.5))
    return data
# Aplica tu programa para encoentrar los terminos de la sucesion cuando n=6,7,8,9,10,12,18,20,
#     a  que valor tiende la sucesion? (esta sucesion tiende a un valor distinto de cero).

valores = [6,7,8,9,10,12,18,20]
result = sucesion(max(valores))

num_plots = len(valores)
cols = 2
rows = -(-num_plots // cols)

plt.figure(figsize=(12, 8))

for i, p in enumerate(valores, 1):
    data = result[:p - 1]
    x_valores = range(2, p + 1)

    plt.subplot(rows, cols, i)
    plt.plot(x_valores, data, marker='o')

    last_value = result[p - 2]
    plt.text(p, last_value - 0.2, f'[{last_value:.5f}]', ha='center', va='bottom', color = "blue")

    plt.title(f'n = {p + 1}')
    plt.grid()
    plt.xticks(x_valores)

```

```
60
61 plt.suptitle("Graficos de la sucesion truncada para diferentes valores de n", fontsize = 18)
62 plt.tight_layout()
63 plt.show()
```



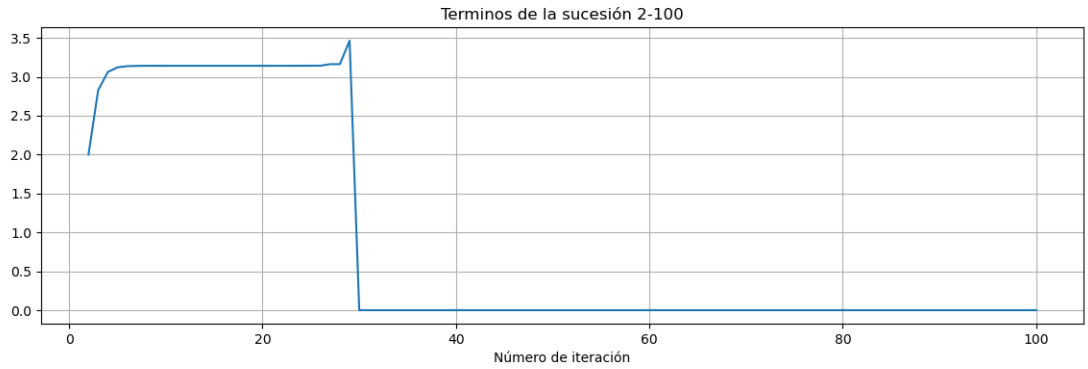
De esta gráfica y los valores que se muestran en la última iteración, sospechamos que se llega a un valor cercano a π, comparamos entonces los últimos valores de la sucesión para cada número de iteraciones con el valor π de la librería **numpy**.

```
1 comparacion = [abs(result[termino-2] - np.pi) for termino in valores]
2 x = [i for i in range(1, len(comparacion) + 1)]
3
4 plt.figure(figsize=(14, 4))
5
6 plt.plot(x, comparacion)
7 plt.xticks(x,[6,7,8,9,10,12,18,20])
8 plt.title("Diferencias entre el valor alcanzado en cada iteracion y $\\pi$")
9 plt.ylabel("Error"); plt.xlabel("numero de iteraciones $n$")
10 plt.grid()
11 plt.show()
```



Notamos que efectivamente el valor tiende a π con un error cada vez menor. Para el siguiente inciso realizamos la gráfica para los valores de la sucesión $[2, 100]$ con el siguiente código:

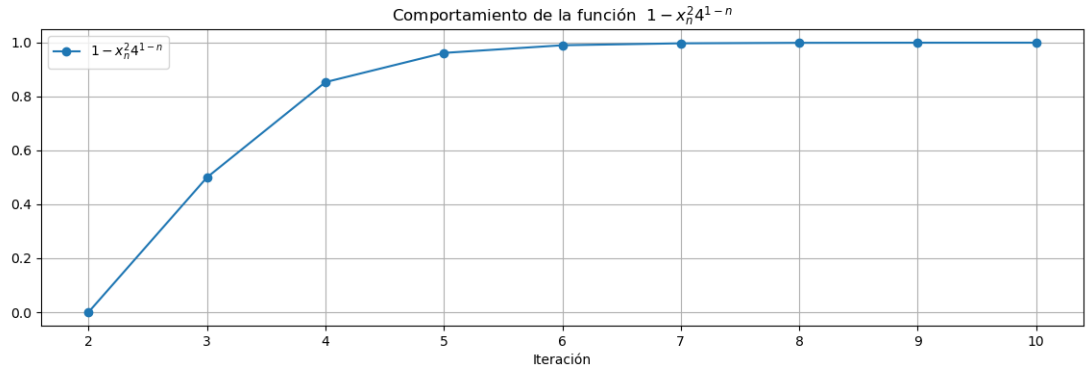
```python
# Encuentra los terminos 50 y 100,  que  sucede con la sucesion? Grafica Los terminos de la sucesion desde
#    2 hasta 100. Explica detalladamente el porque del comportamiento que tiene la computadora al calcular
#    terminos.

result = sucesion(100)

print(f"El termino numero 50 es: {result[49]}")
print(f"El termino numero 100 es: {result[-1]}")

plt.figure(figsize=(14, 4))

x_valores = range(2, 101)
plt.plot(x_valores, result)
plt.title(f"Terminos de la sucesion 2-{len(result)+1}")
plt.xlabel("Numero de iteracion")
plt.grid()
plt.show()
```



Que regresa para ambos valores 50 y 100 de la sucesión 0,0. Notamos que a partir de de la sucesión 25 el valor sube por arriba de π para después desplomarse a cero. Este comportamiento se debe en esencia a como se calcula el valor, notamos que hasta la derecha de la expresión tenemos la forma $1 - x_n^2 4^{1-n}$, tenemos un valor 4^{1-n} que para n cada vez mayor se va haciendo más pequeño y hace que esta parte de la función tienda a 1, lo cual hace que toda la función se haga cero.

Para hacer esto más claro, veremos como se comporta esta parte modificando el código para la evaluación de la sucesión y así obteniendo su evolución

```python
def sucesion_mod(n):
    data = [2]  # x_2
    datos_investigar = []

    for elemento in range(2, n):
        expresion = 2**(elemento - 0.5) * ((1 - (1 - (4**(1 - elemento)) * (data[-1]**2))**0.5)**0.5)
        zero = 1 - (4**(1 - elemento)) * (data[-1]**2)

        data.append(expresion)
        datos_investigar.append(zero)

    return datos_investigar

n = 10
datos = sucesion_mod(n + 1)

plt.figure(figsize=(14, 4))
x = range(2, n + 1)
plt.plot(x, datos, marker="o", label="$1 - x_{n}^{2}4^{1-n}$")
plt.title("Comportamiento de la funcion  $1 - x_{n}^{2}4^{1-n}$")
plt.xlabel("Iteracion")
plt.xticks(x)
plt.grid()
plt.legend()
plt.show()
```



Vemos que en apenas 10 iteraciones se converge bastante bien a 1. Si esto todavía no nos convence de que este valor debe llegar a 1, podemos evaluar el error que existe con el propio valor, para ello realizar el siguiente sencillo código:

```
n = 30
datos = sucesion_mod(n + 1)
errores = [abs(1 - dat) for dat in datos]
for iteracion, error in enumerate(errores):
    print(f"Iteracion {iteracion} : {error}")
```

```
    Iteración 0 : 1.0
    Iteración 1 : 0.5000000000000001
    Iteración 2 : 0.14644660940672627
    Iteración 3 : 0.03806023374435663
    Iteración 4 : 0.009607359798384785
    Iteración 5 : 0.0024076366639015356
    Iteración 6 : 0.0006022718974137975
    Iteración 7 : 0.00015059065189793053
    Iteración 8 : 3.7649080427692994e-05
    Iteración 9 : 9.412358699445456e-06
    Iteración 10 : 2.35309521190441e-06
    Iteración 11 : 5.882741490603749e-07
    Iteración 12 : 1.4706855888668713e-07
    Iteración 13 : 3.676714110945056e-08
    Iteración 14 : 9.191785443896094e-09
    Iteración 15 : 2.2979463887295992e-09
    Iteración 16 : 5.744866804491267e-10
    Iteración 17 : 1.4362167011228166e-10
    Iteración 18 : 3.590550079479726e-11
    Iteración 19 : 8.976375198699316e-12
    Iteración 20 : 2.244093799674829e-12
    Iteración 21 : 5.611067166455541e-13
    Iteración 22 : 1.4022116801015727e-13
    Iteración 23 : 3.5083047578154947e-14
    Iteración 24 : 8.770761894538737e-15
    Iteración 25 : 2.220446049250313e-15
    Iteración 26 : 5.551115123125783e-16
    Iteración 27 : 1.1102230246251565e-16
    Iteración 28 : 0.0
```

¡Bingo! El valor de la iteración número 27 coincide perfectamente con nuestro epsilon de la computadora (valor encontrado en el siguiente problema). Efectivamente notamos que durante esta iteración en la sucesión original el valor se desploma a cero. Por otra parte, el pequeño pico que podemos observar en la gráfica "Términos de la sucesión 2 - 100"puede estar relacionado con la inestabilidad numérica que proporciona esta variable (mejor dicho, parte de la función) al ser un valor que disminuye con tanta velocidad.

Con respecto a la utilidad de la función para el cálculo de π podemos decir que debido a que la función converge demasiado rápido se requiere de pasos que decrecen todavía mucho menores, por lo que analíticamente se presenta un problema en los cálculos ya mostrado, sin embargo como se muestra en la gráfica "Diferencias entre el valor alcanzado en cada iteración y π", para propósitos numéricos y de cálculo la sucesión truncada al término 24 es una aproximación bastante buena.

3. Al número representable inmediatamente después de la unidad se le conoce como el epsilon de la computadora $\epsilon_{machine}$, el cual nos permite calcular el número real representable inmediatamente posterior; esto se obtiene al multiplicar el epsilon por el real y sumarlo a ese real. Utilizando el siguiente algoritmo, crea un código que calcule el $\epsilon_{machine}$ de tu computadora, sólo hace falta indicar el tipo de variable que es .epsilon"(float o doble). El último valor impreso corresponde al epsilon de la computadora. **(2 pts)**

Cálculo del epsilón

```
epsilon = 1;
while(1+epsilon>1)
{
   printf("%f\n",epsilon);
   epsilon = epsilon/2;
}
```

Cabe señalar que el denominador para calcular el epsilon es dos porque la computadora usa numeración binaria.

Escribimos el código en Python y pedimos que imprima la iteración de cada paso junto con su valor:

```
epsilon = 1
iterador = 0
while 1 + epsilon > 1:
    print(f"Epsilon {iterador} ---> {epsilon}")
    epsilon /= 2
    iterador += 1
print(f"El epsilon de esta m quina es de: {epsilon}")
```

```
    Epsilon 0 ---> 1
    Epsilon 1 ---> 0.5
    Epsilon 2 ---> 0.25
    Epsilon 3 ---> 0.125
    Epsilon 4 ---> 0.0625
    Epsilon 5 ---> 0.03125
    Epsilon 6 ---> 0.015625
    Epsilon 7 ---> 0.0078125
    Epsilon 8 ---> 0.00390625
    Epsilon 9 ---> 0.001953125
    Epsilon 10 ---> 0.0009765625
    Epsilon 11 ---> 0.00048828125
    Epsilon 12 ---> 0.000244140625
    Epsilon 13 ---> 0.0001220703125
    Epsilon 14 ---> 6.103515625e-05
    Epsilon 15 ---> 3.0517578125e-05
    Epsilon 16 ---> 1.52587890625e-05
    Epsilon 17 ---> 7.62939453125e-06
    Epsilon 18 ---> 3.814697265625e-06
    Epsilon 19 ---> 1.9073486328125e-06
    Epsilon 20 ---> 9.5367431640625e-07
    Epsilon 21 ---> 4.76837158203125e-07
    Epsilon 22 ---> 2.384185791015625e-07
    Epsilon 23 ---> 1.1920928955078125e-07
    Epsilon 24 ---> 5.960464477539063e-08
    Epsilon 25 ---> 2.9802322387695312e-08
    Epsilon 26 ---> 1.4901161193847656e-08
    Epsilon 27 ---> 7.450580596923828e-09
    Epsilon 28 ---> 3.725290298461914e-09
    Epsilon 29 ---> 1.862645149230957e-09
    Epsilon 30 ---> 9.313225746154785e-10
    Epsilon 31 ---> 4.656612873077393e-10
    Epsilon 32 ---> 2.3283064365386963e-10
    Epsilon 33 ---> 1.1641532182693481e-10
    Epsilon 34 ---> 5.820766091346741e-11
    Epsilon 35 ---> 2.9103830456733704e-11
```

```
    Epsilon 36 ---> 1.4551915228366852e-11
    Epsilon 37 ---> 7.275957614183426e-12
    Epsilon 38 ---> 3.637978807091713e-12
    Epsilon 39 ---> 1.8189894035458565e-12
    Epsilon 40 ---> 9.094947017729282e-13
    Epsilon 41 ---> 4.547473508864641e-13
    Epsilon 42 ---> 2.2737367544323206e-13
    Epsilon 43 ---> 1.1368683772161603e-13
    Epsilon 44 ---> 5.684341886080802e-14
    Epsilon 45 ---> 2.842170943040401e-14
    Epsilon 46 ---> 1.4210854715202004e-14
    Epsilon 47 ---> 7.105427357601002e-15
    Epsilon 48 ---> 3.552713678800501e-15
    Epsilon 49 ---> 1.7763568394002505e-15
    Epsilon 50 ---> 8.881784197001252e-16
    Epsilon 51 ---> 4.440892098500626e-16
    Epsilon 52 ---> 2.220446049250313e-16
    El epsilon de esta máquina es de: 1.1102230246251565e-16
```

Verificamos que se cumpla la condición del epsilon

```
# Verificamos que se cumpla la condicion del epsilon
1 + epsilon == 1
```

```
    True
```

Por lo tanto, vemos que efectivamente la computadora no puede evaluar este epsilon con respecto a otro valor añadido.