UNIVERSIDAD ESTATAL
PENÍNSULA DE SANTA ELENA

"Simulación numérica Programa de Maestría en Petróleo Cohorte 3"

Asignatura: Simulación numérica de yacimientos
Nombre del Profesor: Ing. José Villegas
Autores: Alex Zambrano Acosta / Alisson Giler / Kevin Soto/Fabian Rivera

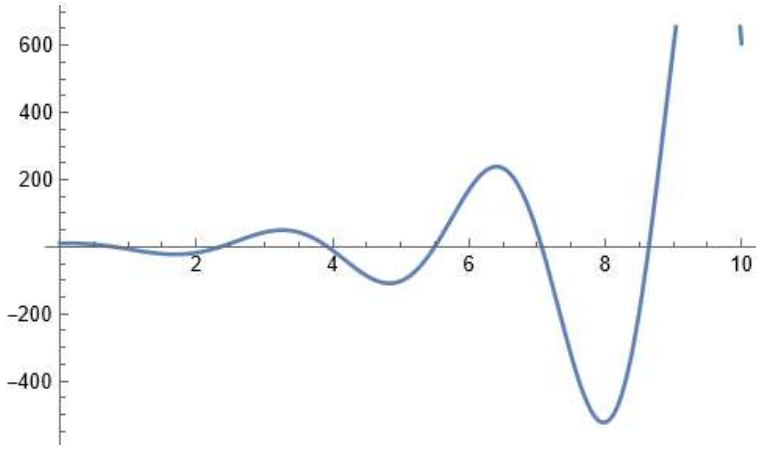
1. Resolver la siguiente ecuación

Utilizando software de programación resolver los siguientes problemas:
1. Dada la siguiente función $f(x) = 10e^{\frac{x}{2}}Cos(2x)$ en el intervalo [0,10]. Graficar la función y encontrar las raices.

Se utiliza el programa Wolfram mathematic

```
Plot[10^(x/2) Cos[2x],{x,0.0,10.0}]
f[x_]=10^(x/2) Cos[2x];
a=8.0;
b=10.0;
c=N[(b+a)/2];
n=10;
For[i=1,i<=n, i++,
c=N[(b+a)/2];
If[f[c]<0,
a=c,
b=c];
Print["iter= ",i,"  a= ", a, "  b= ",b, "  c=",c]
];
```

Grafica



Iteraciones

```
iter= 1   a= 8.     b= 9.     c=9.
iter= 2   a= 8.5    b= 9.     c=8.5
iter= 3   a= 8.5    b= 8.75   c=8.75
iter= 4   a= 8.625  b= 8.75   c=8.625
iter= 5   a= 8.625  b= 8.6875 c=8.6875
iter= 6   a= 8.625  b= 8.65625 c=8.65625
iter= 7   a= 8.625  b= 8.64063 c=8.64063
iter= 8   a= 8.63281 b= 8.64063 c=8.63281
iter= 9   a= 8.63672 b= 8.64063 c=8.63672
iter= 10  a= 8.63867 b= 8.64063 c=8.63867
```

Analizando la gráfica deducimos que en el rango [0,10] tenemos varias raíces, para resolver el ejercicio se utilizó el método de bisección, para lo cual se utilizó un rango de [8,10] debido a que el método considera que la función tiene una sola raíz.

2. Resolver el siguiente sistema de ecuaciones no lineales.

$$\begin{cases} f_1(x_1, x_2, x_3) = 3x_1 - Cos(x_2 x_3) - \frac{1}{2} \\ f_2(x_1, x_2, x_3) = x_1^2 - 81(x_2 + 0.1)^2 + Sin(x_3) + 1.06 \\ f_3(x_1, x_2, x_3) = e^{-x_1 x_2} + 20x_3 + \frac{10\pi - 3}{3} \end{cases}$$

Método de Newton-Raphson usando Phyton

- Para el primer paso importamos las librerías y definimos las variables

[1]:
```
import numpy as np
from sympy import symbols, cos, sin, exp, Matrix, lambdify

# Definir las variables
x1, x2, x3 = symbols('x1 x2 x3')

# Definir las funciones del sistema
f1 = 3*x1 - cos(x2 * x3) - 1/2
f2 = x1**2 - 81*(x2 + 0.1)**2 + sin(x3) + 1.06
f3 = exp(-x1 * x2) + 20 * x3 + (10 * 3.14 - 3) / 3
```

- Luego hacemos el cálculo de la matriz jacobiana

```
# Definir el sistema de ecuaciones como una matriz
system = Matrix([f1, f2, f3])

# Calcular la matriz Jacobiana
jacobian = system.jacobian([x1, x2, x3])

# Convertir las funciones y la jacobiana a funciones numéricas
F = lambdify((x1, x2, x3), system, 'numpy')
J = lambdify((x1, x2, x3), jacobian, 'numpy')
```

- Luego convertimos la jacobiana en funciones numéricas, e iniciamos la solución con los valores iniciales que a continuación se observan.

```
# Convertir las funciones y la jacobiana a funciones numéricas
F = lambdify((x1, x2, x3), system, 'numpy')
J = lambdify((x1, x2, x3), jacobian, 'numpy')

# Inicializar la solución
x = np.array([0.1, 0.1, -0.1], dtype=np.float64)
```

- Mediante el método de newton-Raphson evaluamos la función en el punto mencionado anteriormente y le damos una tolerancia de 1^{-10}.

```
# Método de Newton-Raphson
def newton_raphson(F, J, x0, tol=1e-10, maxiter=100):
    x = x0
    for _ in range(maxiter):
        # Evaluar la Jacobiana y la función en el punto actual
        J_eval = np.array(J(*x), dtype=np.float64)
        F_eval = np.array(F(*x), dtype=np.float64).flatten()
```

- Finalmente calculamos la inversa de la jacobina e imprimimos los resultados.

```
        # Calcular la inversa de la Jacobiana
        J_inv = np.linalg.inv(J_eval)

        # Calcular el cambio
        delta = np.dot(J_inv, F_eval)

        # Actualizar la solución
        x = x - delta

        # Verificar la convergencia
        if np.linalg.norm(delta, ord=2) < tol:
            break
    return x

# Encontrar la solución
solution = newton_raphson(F, J, x)
print(solution)
```

```
[ 5.00000000e-01  1.42091776e-05 -5.23332978e-01]
```

```
k     x1         x2         x3         ||x(k)-x(k-1)||
0     0.1000     0.1000     -0.1000
1     0.499870   0.019467   -0.521520   0.586567
2     0.500014   0.001589   -0.523557   0.017994
3     0.500000   0.000012   -0.523598   0.001577
4     0.500000   0.000000   -0.523599   0.000012
5     0.500000   0.000000   -0.523599   0.000000
6     0.500000   0.000000   -0.523599   0.000000
Procedimiento exitoso
```

3. Resolver el siguiente sistema de ecuaciones

$$\begin{cases} 6x_1 + 3x_2 - x_3 = 40 \\ -2x_1 + 4x_2 + 2x_3 = 20 \\ x_1 - 3x_2 + 8x_3 = 2 \end{cases}$$

```
niters=20;

Xn={0.0,0.0,0.0};
B={40/6,5,1/4};
M={{0.0,-1/3,1/6},{1/2,0.0,-1/2},{-1/8,3/8,0.0}};

For[i=1, i<= niters, i++,
Xnp1 = B + M.Xn;

 Print["iter= ",i," Xn= ", Xn, " Xnp1= ",Xnp1];

 Xn=Xnp1;

]
```

```
iter= 1 Xn= {0., 0., 0.} Xnp1= {6.66667, 5., 0.25}
iter= 2 Xn= {6.66667, 5., 0.25} Xnp1= {5.04167, 8.20833, 1.29167}
iter= 3 Xn= {5.04167, 8.20833, 1.29167} Xnp1= {4.14583, 6.875, 2.69792}
iter= 4 Xn= {4.14583, 6.875, 2.69792} Xnp1= {4.82465, 5.72396, 2.3099}
iter= 5 Xn= {4.82465, 5.72396, 2.3099} Xnp1= {5.14366, 6.25738, 1.7934}
iter= 6 Xn= {5.14366, 6.25738, 1.7934} Xnp1= {4.87977, 6.67513, 1.95356}
iter= 7 Xn= {4.87977, 6.67513, 1.95356} Xnp1= {4.76722, 6.46311, 2.1432}
iter= 8 Xn= {4.76722, 6.46311, 2.1432} Xnp1= {4.8695, 6.31201, 2.07776}
iter= 9 Xn= {4.8695, 6.31201, 2.07776} Xnp1= {4.90896, 6.39587, 2.00832}
iter= 10 Xn= {4.90896, 6.39587, 2.00832} Xnp1= {4.86943, 6.45032, 2.03483}
iter= 11 Xn= {4.86943, 6.45032, 2.03483} Xnp1= {4.8557, 6.4173, 2.06019}
iter= 12 Xn= {4.8557, 6.4173, 2.06019} Xnp1= {4.87093, 6.39775, 2.04953}
iter= 13 Xn= {4.87093, 6.39775, 2.04953} Xnp1= {4.87567, 6.4107, 2.04029}
iter= 14 Xn= {4.87567, 6.4107, 2.04029} Xnp1= {4.86981, 6.41769, 2.04456}
iter= 15 Xn= {4.86981, 6.41769, 2.04456} Xnp1= {4.8682, 6.41263, 2.04791}
iter= 16 Xn= {4.8682, 6.41263, 2.04791} Xnp1= {4.87044, 6.41014, 2.04621}
iter= 17 Xn= {4.87044, 6.41014, 2.04621} Xnp1= {4.87099, 6.41211, 2.045}
iter= 18 Xn= {4.87099, 6.41211, 2.045} Xnp1= {4.87013, 6.41299, 2.04567}
iter= 19 Xn= {4.87013, 6.41299, 2.04567} Xnp1= {4.86995, 6.41223, 2.04611}
iter= 20 Xn= {4.86995, 6.41223, 2.04611} Xnp1= {4.87027, 6.41192, 2.04584}
```

Se analiza las iteraciones, nos damos cuenta que los valores de Xn y Xnp1 tienen valores similares es porque convergen.