0UNIVERSIDAD DE ANTIOQUIA. FACULTAD DE INGENIERÍA
TAREA 1 PARTE A. MÉTODOS NUMÉRICOS. 2019-2

NOMBRE: Daniela Sanchez cc: 1023723010

2. Problemas

En cada uno de los casos, a no ser que se diga lo contrario; realizar una implementación computacional del método pedido y presentar gráficamente el problema.

2.1. Métodos cerrados (20 %)

1. (20 %) Usando los métodos de la bisección y la regla falsa usando una $TOL = 10^{-5}$, determine:
   (a) (50 %) Las raíces máxima y mínima de $f(x) = -25182x - 90x^2 + 44x^3 - 8x^4 + 0.7x^5$

El código del método de la bisección está dado por lo siguiente:

```
def biseccion(f,a,b,tol):
  #definir el errro para el arranque.
  error = 99999
  #agregar el contador
  i = 0
  while error > tol:
    #Calcular el punto medio
    p = (a+b)/2
    #Evaluar los signos
    if (f(b)*f(a) < 0):
      b = p
    else:
      a = p
    #actualizar el error
    error = (b-a)/2
    #actualizar las interacciones
    i += 1
  return p,i
```

Para este ejercicio la función “f” es la derivada de la función original, esto se realiza para encontrar los puntos críticos de la función,que son las raíces función derivada.

```
f = 3.5*x**4 - 32*x**3 + 132*x**2 - 180*x - 25182
```

Graficamos la función derivada para encontrar los intervalos donde están las raíces

```
#Graficar la derivada para saber el intervalo de la función
x = np.linspace(-20,20,100)
```

```
f = (lambda x : 3.5*x**4 - 32*x**3 + 132*x**2 - 180*x - 25182)
y = f(x)
plt.plot(x,y,color = 'blue')
plt.title('Grafico de la primera derivada')
plt.xlabel('Eje x')
plt.ylabel('Eje y')

plt.grid()
plt.show()
```

Figura 1: Gráfica de la derivada de la función

Las raíces se encuentran entre [-7.5, -5] y entre [7.5, 12]

- **Para el rango [-7.5, -5]**

```
a = -7.5
b = -5
tol = 1e-5
```

```
p,i = biseccion(f,a,b,tol)
```

El valor de la raíz es: -6.875019073486328 y el número de interacciones es: 17

Evaluamos si es un máximo o un mínimo con la ayuda de la segunda derivada. Se utilizó la siguiente función para lograr esto.

```
def maxomin(f,x):
  y = f(x)
  if (y > 0):
    resp = "minimo"
  else:
    resp = "maximo"
  return resp

segundaderi = (lambda x :14.0*x**3 - 96*x**2 + 264*x - 180)
resp = maxomin(segundaderi,p)
```

Se evalúa el resultado de la raíz en la función original para obtener el valor máximo o mínimo en Y.

```
funcion = (lambda x: -25182*x-90*x**2+44*x**3-8*x**4+0.7*x**5)
valor = funcion (p)
```

El valor máximo es: 125950.85380090223

- **Para el rango [7.5, 12]**

```
a = 10
b = 12
tol = 1e-5
p,i = biseccion(f,a,b,tol)
```

El valor de la raíz es: 10.999984741210938 y el número de interacciones es: 17

```
resp = maxomin(segundaderi,p)
```

```
funcion = (lambda x: -25182*x-90*x**2+44*x**3-8*x**4+0.7*x**5)
valor = funcion (p)
```

El valor mínimo es: -233720.26126442984.

El código del método de la regla falsa está dado por lo siguiente:

```
def ModFalsePos(xl,xu, f, tol):
    i = 0
    xr = (xl+xu)/2
    fl = f(xl)
    fu = f(xu)
    error = 99999
    while (error > tol):

        xold = xr
        xr = xu - fu * (xl - xu) / (fl - fu)
        test = f(xl)*f(xr)
        if (test<0):
            xu = xr
            fu = f(xu)
        else:
            xl = xr
            fl = f(xl)
        error = abs((xr - xold)/xr)
        i=i+1
    return xr, i
```

- **Para el rango [-7.5, -5]**

```
a = -7.5
b = -5
tol = 1e-5
p,i = biseccion(f,a,b,tol)
```

El valor de la raíz es: -6.828435309465146 y el número de interacciones es: 6.

El valor máximo es: 125962.75682080137

- **Para el rango [7.5, 12]**

a = 10
b = 12
tol = 1e-5
p, i = ModFalsePos(a,b,f,tol)

El valor de la raíz es: 11.249998657392343 y el número de interacciones es: 6
El valor mínimo es: -234041.94580074813

La figura 2 ilustra los valores aproximados hallados con los métodos.

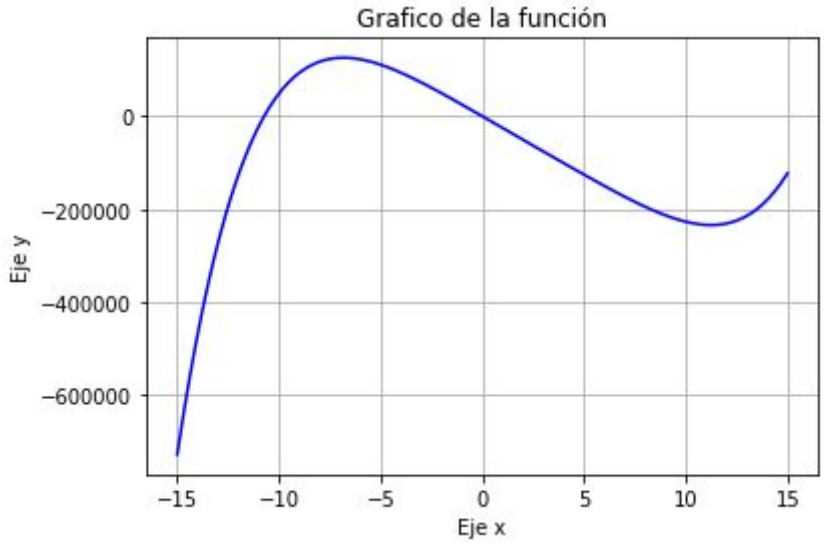


Figura 2: Gráfica de la función -25182*x-90*x**2+44*x**3-8*x**4+0.7*x**5

(b) (50 %) La raíz real de $x^2|\cos(\sqrt{x})| = 5$
Analice los resulados

Para encontrar los intervalos de las raíces se utilizó la figura 3, donde se evidencia que existe más de una raíz real. Para efectos prácticos se obtendrá 4 raíces, en los rangos [-4.8, -3], [-3,0], [0, 3] y [3, 4.8]

Figura 3: función x**2*abs((cos(x)))-5

Con el método de la bisección se encontró las siguientes raíces.

- **[-4.8, -3]**

b = -3
tol = 1e-5
p, i = biseccion(f,a,b,tol)

El valor de la raíz es: -4.575013732910156 y el número de interacciones es: 17

- **[-3,0]**

a = -3
b = 0
tol = 1e-5
p, i = biseccion(f,a,b,tol)

El valor de la raíz es: -2.625011444091797 y el número de interacciones es: 18

- **[0, 3]**

a = 0
b = 3
tol = 1e-5

p, i = biseccion(f,a,b,tol)

El valor de la raíz es: 1.4999885559082031 y el número de interacciones es: 18

- **[3, 4.8]**

a = 3
b = 4.8
tol = 1e-5
p, i = biseccion(f,a,b,tol)

El valor de la raíz es: 3.8999862670898438 y el número de interacciones es: 17
Con el método de la regla falsa se encontró las siguientes raíces.

- El valor de la raíz es: -4.458076080698392 y el número de interacciones es: 6
- El valor de la raíz es: -2.4990769883603057 y el número de interacciones es: 5
- El valor de la raíz es: 2.4990769883603057 y el número de interacciones es: 5
- El valor de la raíz es: 4.458076080698392 y el número de interacciones es: 6

Análisis de los resultados

El error decrece mucho más rápidamente en el método de la falsa posición que en el de la bisección, debido a un esquema más eficiente en el método de la falsa posición para la localización de raíces. Por lo tanto, el intervalo, como se definió por $\Delta x/2 = |xu – xl|/2$ para la primera iteración, proporciona una medida del error en este método. Éste no es el caso con el método de la falsa posición, ya que uno de los valores iniciales puede permanecer fijo durante los cálculos, mientras que el otro converge hacia la raíz.

2. (30 %) Como se muestra en la figura 1, la velocidad del agua, v (m/s) descargada por un tanque de forma cilindrica a través de un tubo largo está dada por:

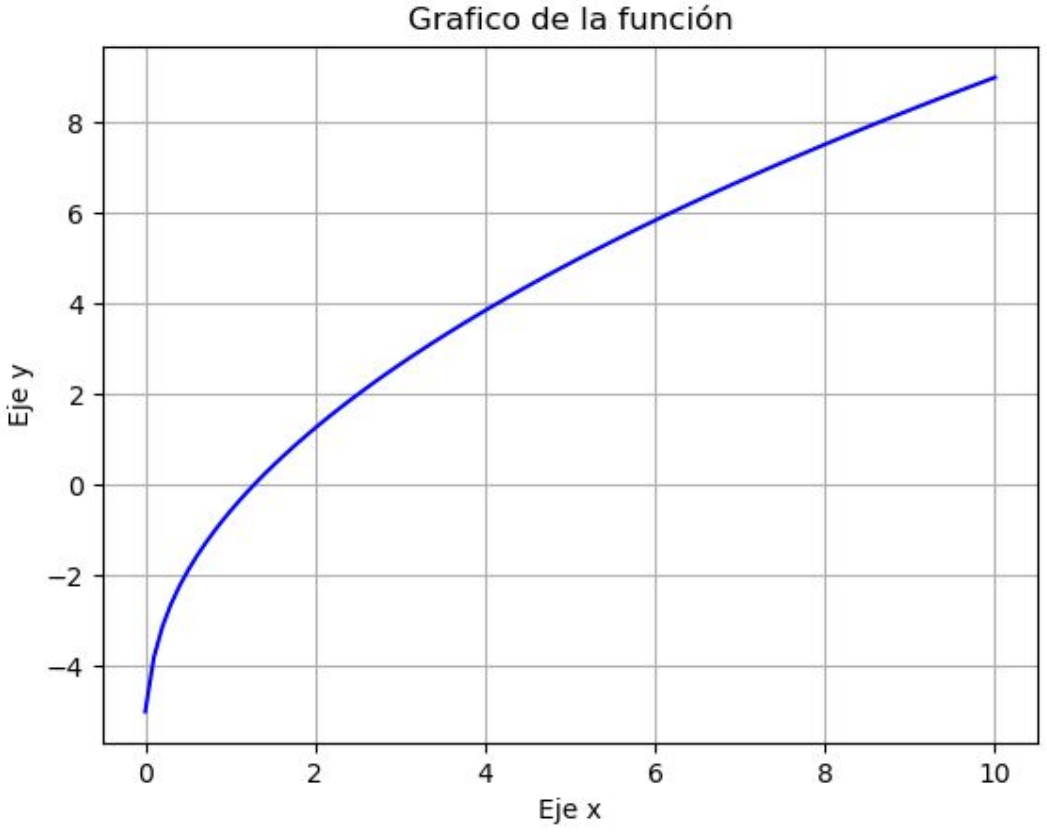
$$v = \sqrt{2gH}\tanh\left(\sqrt{\frac{2gH}{2L}}t\right)$$

donde $g = 9.81$ m/s^2, H es la cabeza inicial (m), L es la longitud del tubo (m) y t es el tiempo transcurrido (s). Escriba un código en **Matlab/Octave o Pyhton** que:

(a) (50 %) Utilice el método de la bisección con las semillas $a = 0$ y $b = 4m$ para determinar la cabeza inicial requerida para alcanzar una velocidad de salida de $v = 5$m/s en $t = 2.5$s para un tubo de 4m de largo. Use $tol = 0.0000001$

(b) (50 %) Repita el procedimiento anterior usando el método de la regla falsa. Analice los resultados

Para este ejercicio se obtuvo una gráfica que se observa en la figura 4.



.Figura 4: Función sqrt(19.6*x)*tanh(sqrt(2.45*x)*2.5)-5

- Con el método de la bisección se encontró

El valor de la raíz es: 0.9999998807907104 y el número de interacciones es: 25

- Con el método de la regla falsa se encontró

El valor de la raíz es: 1.276248363724627 y el número de interacciones es: 24

3. (20 %) La figura 2a muestra una viga uniforme sujeta a una carga distribuida que incrementa linealmente con la distancia. La ecuación de la curva elástica resultante en la figura 2b es:

$$y = \frac{\omega_0}{120EIL}(-x^5 + 2L^2x^3 - L^4x)$$

donde $L = 600\text{cm}$, $E = 50000\text{kN/cm}^2$, $I = 30000\text{cm}^4$ y $\omega_0 = 2.5$ kN/cm y $TOL = 10^{-6}$

(a) (75 %) Use el método de la bisección para determinar el punto de máxima deflexión.

(b) (15 %) Determine el valor de la deflexión máxima.

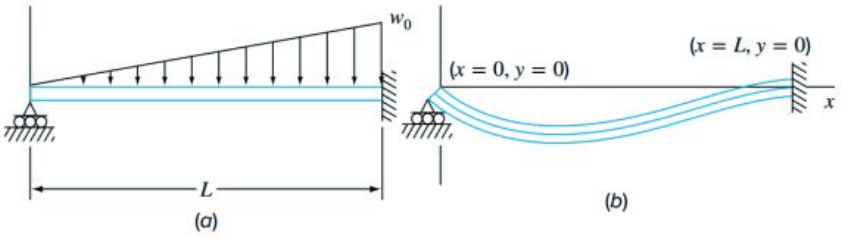


Figura 2: Viga deflactada por carga distribuida

Se encontró los puntos críticos con ayuda de la primera derivada. La gráfica de esta función se observa en la figura 5.

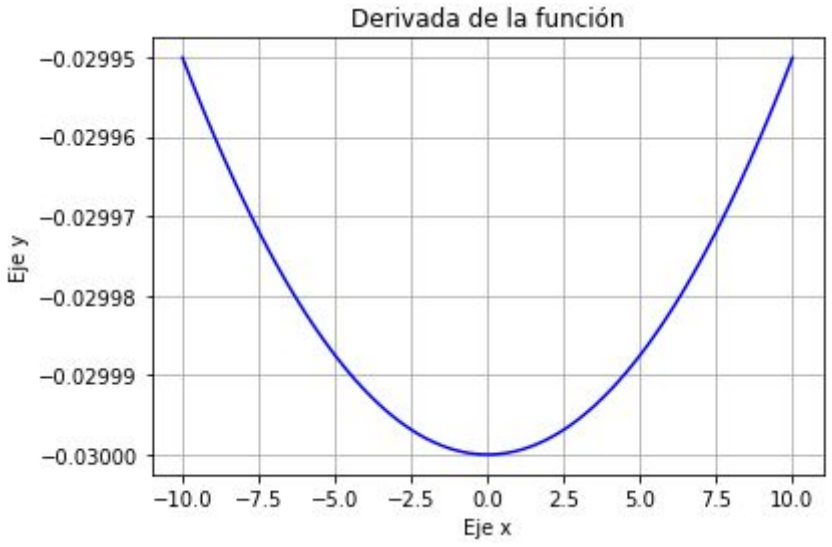


Figura 5: Primera derivada.

El rango donde se encuentra la raíz es: **[-10,10]**

El valor de la raíz es: 9.999998807907104 y el número de interacciones es: 24

La máxima deflexión es el punto: (-0.29983341337082287, 0)

La gráfica de la función se observa en la figura 6

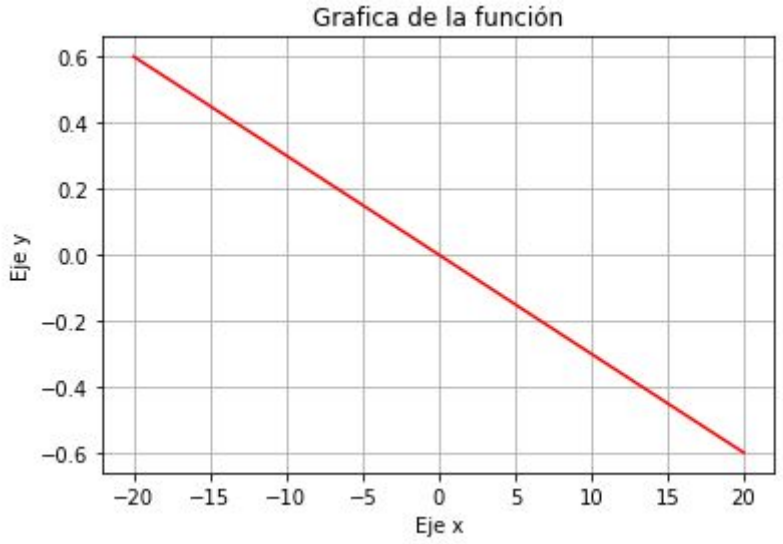


Figura 6: Gráfica de la función original

4. (30 %) La resistividad ρ del silicio dopado está basada en la carga q sobre el electrón, la densidad n del electrón y la movilidad del electrón μ. La densidad el electrón es dada en términos de la densidad dopada N y la densidad intrínseca del portador n_i. La movilidad del electrón es descrita por la temperatura T, la temperatura de referencia T_0 y la movilidad de referencia μ_0. Las ecuaciones requeridas para calcular la resistividad son:

$$\rho = \frac{1}{qn\mu}$$

donde,

$$n = \frac{1}{2}\left(N + \sqrt{N^2 + 4n_i^2}\right) \quad \text{y} \quad \mu = \mu_0 \left(\frac{T}{T_0}\right)^{-2.42}$$

Determine N dado $T_0 = 300K$, $T = 1000K$, $\mu_0 = 1360cm^2(Vs)^{-1}$, $q = 1.7 \times 10^{-19}C$, $n_i = 6.21 \times 10^9 cm^{-3}$ y un deseado $\rho = 6.5 \times 10^6 Vscm/C$. Suponga valores iniciales de $N = 0$ y $N = 2.5 \times 10^{10}$ con $TOL = 10^{-6}$

(a) (50 %) Utilice el método de la bisección

(b) (50 %) Utilice el método de la regla falsa y compare resultados

La gráfica de la función se observa en la figura 7

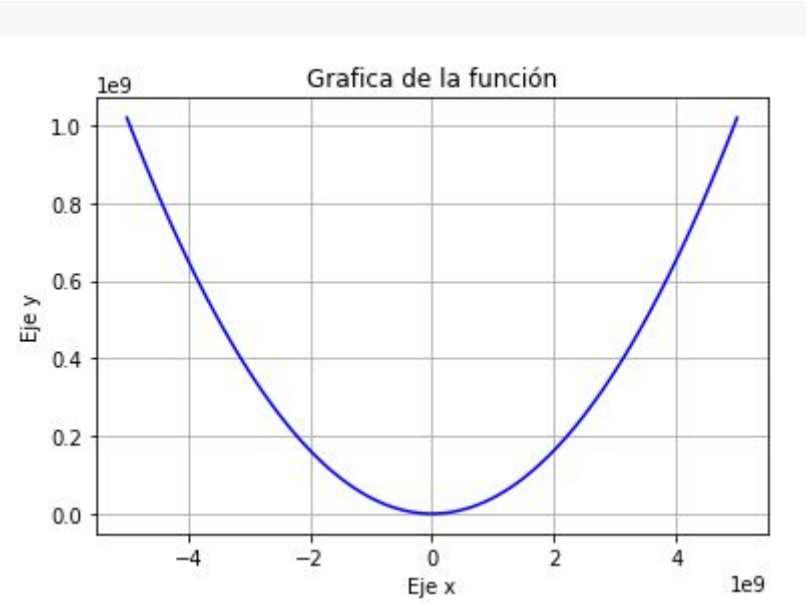


Figura 7: Gráfica de la función

Los intervalos son:

a = -2.5e9

b = 2.5e9

tol = 1e-6

p,i = biseccion(fu,a,b,tol)

El valor de la raíz es: 76293.94531138978 y el número de interacciones es: 52

a = 0

b = 2.5e9

tol = 1e-6

p,i = ModFalsePos(a,b,fu,tol)

El valor de la raíz es: 108750.81051063538 y el número de interacciones es: 58760

Los resultados tiende a ser más cercano al cero el método de la regla falsa. Aunque se encontró en más interacciones ya que el rango fue mayor.

2.2. Métodos abiertos (20 %)

1. (25 %) Para los ejercicios, escriba un código en **Matlab/Octave o Pyhton** que grafique en el intervalo pedido y que aplique la técnica SOR automáticamente. En todos los casos use $tol = 10^{-5}$

   (a) (33.3 %) Escriba la ecuación $x^3 + 3x - 6 = 0$ de 4 formas distintas y aplique la técnica SOR para determinar la raíz existente en el intervalo $[1, 2]$

1) $3x = 6 - x^3$

$$x = \frac{6 - x^3}{3}$$

2) $x(x^2 + 3) = 6$

$$x = \frac{6}{x^2 + 3}$$

3) $x(x^2 + 3) = 6$

$$x^2 + 3 = \frac{6}{x}$$

$$x = \sqrt{\frac{6}{x} - 3}$$

4) $x^3 = 6 - 3x$

$$x = \sqrt[3]{6 - 3x}$$

Con ayuda del siguiente código se aplica la técnica SOR:

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt


def m(g, a, b):
  m= abs(g(b)-g(a))/(b-a)
  return m




def Fixpt(g,x0,a,b,tol):
  #definir el errro para el arranque.
```

```
error = 99999
#agregar el contador
i = 0
#raiz inicial
xr = x0
while error > tol:
  xrold = xr
  xr = g(xrold)
  i = i + 1
  if(xr != 0):
    error = abs((xr-xrold)/xr)
  else:
    break

return xr, i
```

Raíz que genera el resultado es el siguiente:

El valor de la raíz es: 1.2879155176396861 y el número de interacciones es: 97

La gráfica de la función se muestra a continuación.

Figura 8: Gráfica de la función

(b) (33.3 %) Escriba la ecuación $10e^{-x}\sin(x) - 1 = 0$ de 3 formas distintas y encuentre la solución en el intervalo $[0, 1]$

1. $10e^{-x}\sin(x)=1$

$$\sin(x)=\frac{1}{10e^{-x}}$$

2. $e^{-x}=\frac{1}{10\sin(x)}$

$$-x=\ln\left(\frac{1}{10\sin(x)}\right)$$

$$x=-\ln\left(\frac{1}{10\sin(x)}\right)$$

3. $10e^{-x}\left(\frac{e^{ix}-e^{-ix}}{2i}\right)=1$

$$10e^{x(1-i)}-10e^{-x(1+i)}=2i$$

$$e^{x(1-i)}=\frac{2i+10e^{-x(1+i)}}{10}$$

$$x(1-i)=\ln\left(\frac{2i+10e^{-x(1+i)}}{10}\right)$$

$$x=\frac{\ln\left(\frac{2i+10e^{-x(1+i)}}{10}\right)}{1-i}$$

```
g = lambda x : (np.arcsin(1/(10*np.exp(-x))))
x0 = 0.5
a = 0
b = 1
tol = 1e-5
m = m(g,a,b)
G = (lambda x, M = m: (float(M*x)-1.0*(np.arcsin(1/(10*np.exp(-x)))))/
(float(M)-1))
xr, i = Fixpt(G,x0,a,b,tol)
```

El valor de la raíz es: 0.11209676353600473 y el número de interacciones es: 6

La gráfica de la función está dada en la figura 9

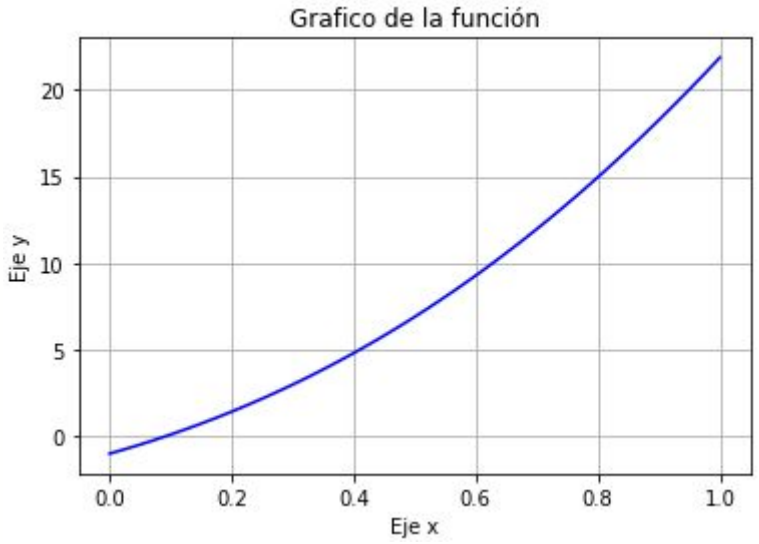


Figura 9: Gráfica de la función

(c) (33.3 %) Escriba la ecuación $\tan(2-x)-x=0$ de 3 formas distintas y encuentre la raíz en el intervalo $[1.1, 5]$

1. $x=\tan(2-x)$

2. $2-x=\arctan(x)$

   $x=2-\arctan(x)$

3. $\frac{e^{(2-x)i}-e^{-(2-x)i}}{e^{(2-x)i}+e^{-(2-x)i}}\cdot\frac{1}{i}=x$

   $e^{(2-x)i}-e^{-(2-x)i}=xi\left(e^{(2-x)i}+e^{-(2-x)i}\right)$

   $e^{(2-x)i}=xi\left(e^{(2-x)i}+e^{-(2-x)i}\right)+e^{-(2-x)i}$

$$2-x=\frac{\ln\left(xi\left(e^{(2-x)i}+e^{-(2-x)i}\right)+e^{-(2-x)i}\right)}{i}$$

$$x=2-\frac{\ln\left(xi\left(e^{(2-x)i}+e^{-(2-x)i}\right)+e^{-(2-x)i}\right)}{i}$$

g = lambda x : (2-np.arctan(x))

x0 = 2

a = 1.1

b = 5

tol = 1e-5

m = m(g,a,b)

```
G = (lambda x, M = m: (float(M*x)-1.0*(2-np.arctan(x))/(float(M)-1)))
xr, i = Fixpt(G,x0,a,b,tol)
```

El valor de la raíz es: 1.4097956151409228 y el número de interacciones es: 9

La gráfica está está dada en la figura 10

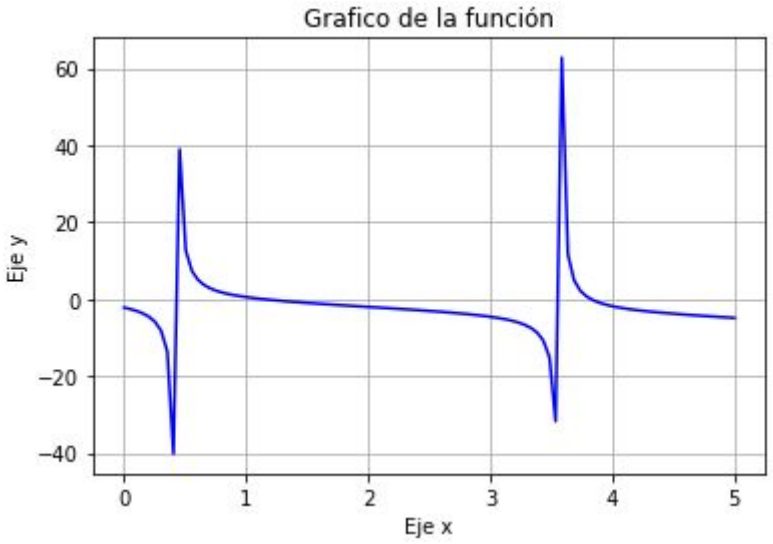


Figura 10: Gráfica de la función

2. (25 %) Una catenaria es un cable que está colgado entre dos puntos que no están en la misma línea vertical. Como se observa en la figura 3a, este no está sujeto a otra carga que su mismo peso.

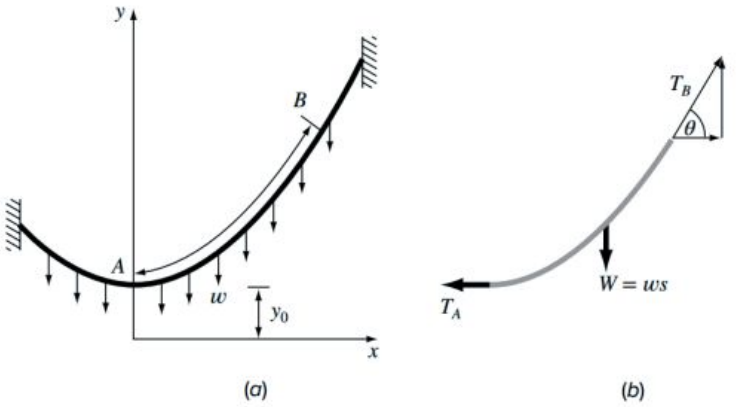


Figura 3: Esquema de catenaria y diagrama de cuerpo libre

Además, el peso actúa como una carga uniforme y distribuida por unidad de longitud a lo largo del cable ω (N/m). Un diagrama de cuerpo libre para el cable se ve en la figura 3b, donde T_A y T_B son las fuerzas de tensión al final. Basado en un balance de fuerzas horizontal y vertical, se puede derivar el siguiente modelo diferencial de la situación:

Además, el peso actúa como una carga uniforme y distribuida por unidad de longitud a lo largo del cable ω (N/m). Un diagrama de cuerpo libre para el cable se ve en la figura 3b, donde T_A y T_B son las fuerzas de tensión al final. Basado en un balance de fuerzas horizontal y vertical, se puede derivar el siguiente modelo diferencial de la situación:

$$\frac{d^2y}{dx^2} = \frac{\omega}{T_A}\sqrt{1+\frac{dy}{dx}}$$

Solucionando esta ecuación se obtiene que la altura del cable en función de la distancia es:

$$y = \frac{T_A}{\omega}\cosh\left(\frac{\omega}{T_A}x\right) + y_0 - \frac{T_A}{\omega}$$

(a) (50 %) Utilice el método de Newton-Raphson para calcular el valor del parámetro T_A cuando $\omega = 10$ y $y_0 = 5$. Suponga que el cable tiene una altura de 15m y una distancia de 50m con $TOL = 10^{-6}$

(b) (50 %) Repita el procedimiento anterior con el método de la secante y analice los resultados.

Reemplazando la función se obtiene la siguiente expresión:

$$15=\frac{T_A}{10}\cosh\left(\frac{150}{T_A}\right)+5-\frac{T_A}{10}$$

$$\frac{T_A}{10}\cosh\left(\frac{150}{T_A}\right)-\frac{T_A}{10}-10=0$$

def NewtonRaphson(f,fde,x0,tol):

```
  #definir el errro para el arranque.
  error = 99999
  #agregar el contador
  i = 0
  #raiz inicial
  xr = x0
  while error > tol:
    xrold = xr
    xr = xrold-(f(xrold)/fde(xrold))
    i = i + 1
    if(xr != 0):
      error = abs((xr-xrold)/xr)
    else:
      break
  return xr, i

x0 = 1.1

tol = 1e-6

xr, i = NewtonRaphson(f[0],f[1],x0,tol)
```

El valor de la raíz es: 1266.324360399856 y el número de interacciones es: 506

(b) (50 %) Repita el procedimiento anterior con el método de la secante y analice los resultados.

Para el método de la secante se utilizó el siguiente código

```
def Secante(f,x0, x1, tol):
  error = 99999
  raiz = []
  raiz.insert(0,0)
  i = 0

  while error > tol:
    x2 = x1- (f(x1)*(x1-x0))/((f(x1)-f(x0)))
    raiz.append(x2)
    i = i + 1
```

```
    x0 = x1
    x1 = x2
    error = abs ((raiz[i]-raiz[i-1])/(raiz[i]))

  return x2, i
```

El valor de la raíz es: 1266.3243603966591 y el número de interacciones es: 8

La gráfica de la función está dada en la figura 11

Figura 11: Gráfica de la función

3. (25 %) La ecuación de Manning se puede escribir para un canal rectangular de la siguiente forma:

$$Q = \frac{\sqrt{S}(BH)^{5/3}}{n(B+2H)^{2/3}}$$

donde Q es el caudal en m^3/s, S es la pendiente en m/m, H es la profundidad del canal en m y n es el coeficiente de rugosidad de Manning. Plos valores de $Q = 5$, $S = 0.0002$, $B = 20$ y $n = 0.03$:

(a) (50 %) Desarrolle un esquema de iteración de punto fijo para encontrar H hasta que $tol < 0.05\,\%$

Para la técnica de punto fijo despejamos la ecuación resultante en términos de H.

$$\left(\frac{\left(\left(\frac{0.15(20+2H)^{2/3}}{\sqrt{0.0002}}\right)^{3/5}\right)}{20}\right) = H$$

g = (lambda x :((((0.15)*(20+2*x)**(2/3))/((0.0002)**(1/2)))**(3/5))/20)

x0 = 1.1

a = 0

El valor de la raíz es: 0.7023003678181243 y el número de interacciones es: 3

La gráfica de la función está dada en la figura 12

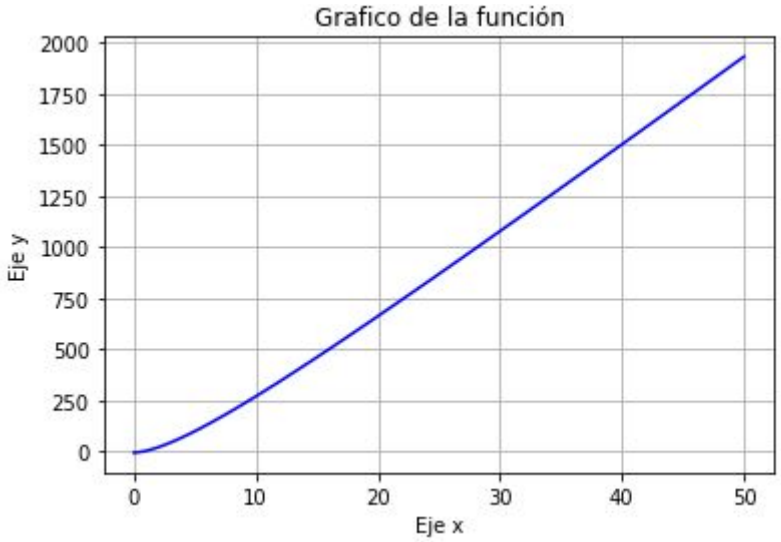


Figura 12: Gráfico de la función

(b) (50 %) Repita el procedimiento anterior con el método de la secante y el de Newton. Analice los resultados.

Newton Raphson

f = (lambda x : (((0.0002)**(1/2)*(20*x)**(5/3))/((0.03)*(20+2*x)**(2/3)))-5,
lambda x: ((x/(x+10))*(437613*x+729355))/(x+10))

x0 = 0.1

tol = 0.0005

xr, i = NewtonRaphson(f[0],f[1],x0,tol)

El valor de la raíz es: 0.5714056013052352 y el número de interacciones es: 500

Secante

x0 = 0

x1 = 1

tol = 0.0005

xr, i = Secante(f,x0,x1,tol)

El valor de la raíz es: 0.7022932531926368 y el número de interacciones es: 5

En este punto en particular se converge más rápido a la solución con con el método de punto fijo y con una solución aproximada la real.

4. (25 %) Describir el movimiento de fluidos a través de tuberías es una tarea muy relevante en diferentes área de ingeniería. Aplicaciones típicas de este problema incluyen sistemas de distribución de agua o sistemas de refrigeración. En la naturaleza, procesos como el flujo de nutrientes en las plantas o el flujo de sangre al interior de las venas se pueden modelar usando esta aproximación. La resistencia a fluir en tales conductos se puede parametrizar mediante un número adimensional llamado *factor de fricción*. Para flujo turbulento, la ecuación de Colebrook sugiere que:

$$0 = \frac{1}{\sqrt{f}} + 2.0 \log\left(\frac{\epsilon}{3.7D} + \frac{2.51}{Re\sqrt{f}}\right)$$

donde ϵ es la rugosidad (m), D el diámetro de la tubería (m) y Re es el número de Reynolds que se calcula como:

$$Re = \frac{\rho V D}{\mu}$$

donde ρ es la densidad del fluído (kg/m^3), V es la velocidad (m/s) y μ es la viscosidad dinámica (N $\cdot$ s/m^2). Se dice que el flujo es Turbulento si $Re > 4000$.
Usando los valores de $\rho = 1.23kg/m^3$, $\mu = 1.79 \times 10^{-5} N \cdot s/m^2, D = 0.005m, V = 40m/s$, $\epsilon = 0.0015mm$ y $TOL = 10^{-5}$, determine el factor de fricción f mediante:

(a) (20 %) El método de Newton - Raphson

La gráfica de la función se observa en la figura 13:

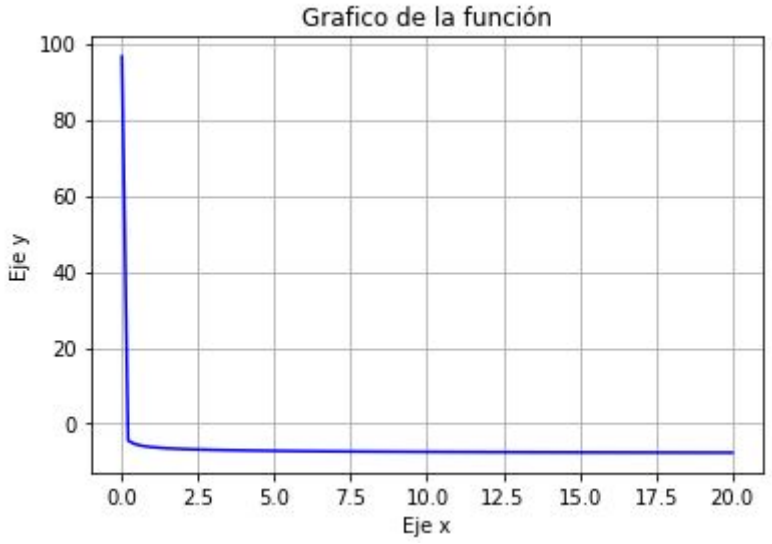


Figura 13: Gráfica de la función

Para este método se utilizaron los siguientes parámetros:

```
f = (lambda x :
(1/(x)**(1/2))+2*np.log10((0.0000015)/(3.7*0.005)+((2.51)/((13743.01676)*(x)**(1/2)))
),
    lambda x: (-1.12627 - 1.47826*np.sqrt(x))/(2.25254*x**(3/2) + x**2))
x0 = 0.01
tol = 1e-6
xr, i = NewtonRaphson(f[0],f[1],x0,tol)
```

Se obtuvo el siguiente resultado:

El valor de la raíz es: 0.02896781017132676 y el número de interacciones es: 6

(b) (20 %) El método de la secante

```
x0 = 0.0001
x1 = 0.001
tol = 1e-5
f = (lambda x :
(1/(x)**(1/2))+2*np.log10((0.0000015)/(3.7*0.005)+((2.51)/((13743.01676)*(x)**(1/2)))
))
xr, i = Secante(f,x0,x1,tol)
```

El valor de la raíz es: 0.0289678101713195 y el número de interacciones es: 12

(c) (30 %) Un esquema de iteración de punto fijo.

```
g = (lambda x :(1/(-2*np.log10((0.0000015)/(3.7*0.005)+((2.51)/
((13743.01676)*(x)**(1/2))))))**2)
x0 = 0.0001
a = 0.0001
b = 1
tol = 1e-5
xr, i = Fixpt(g,x0,a,b,tol)
```

El valor de la raíz es: 0.028967783315271292 y el número de interacciones es: 8

(d) (30 %) Compare y analice los resultados obtenidos con el factor f calculado mediante la aproximación explícita de Swamee - Jain dada por:

$$f = \frac{1.325}{\left[\ln\left(\frac{\epsilon}{3.7D} + \frac{5.74}{Re^{0.9}}\right)\right]^2}$$

$$f = \frac{1.325}{\left(\ln\left(\frac{0.0000015}{3.7*0.005}\right) + \frac{5.74}{13743.01676^{0.9}}\right)^2}$$

$$f = 0.02903099711$$

2.3. Eliminación Gaussiana (20 %)

1. (10 %) Cinco reactores conectados mediante tubos se muestran en la figra 4

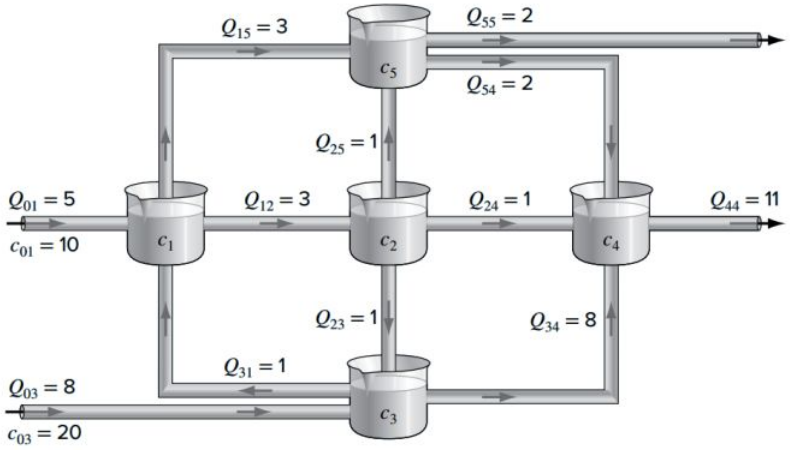


Figura 4: Sistema de reactores interconectados

El flujo másico a través de cada tubo se calcula como el producto del caudal (Q) y la concentración (c) de constituyente. En estado estacionario, el flujo másico de entrada y salida de cada reactor debe ser igual. Por ejemplo, para el primer reactor se tiene que el balance de masa se puede escribir como:

$$Q_{01}c_{01} + Q_{31}c_3 = Q_{15}c_1 + Q_{12}c_1$$

Escriba las ecuaciones para cada reactor y use las funciones de **Matlab/Octave o Pyhton** para resolver el sistema.

$$(5)(10)+1(c_3)=3c_1+3c_1$$

$$6c_1-c_3=50\,(1)$$

$$3c_1=1c_2+1c_2+c_2$$

$$c_1-c_2=0\,(2)$$

$$1c_2+(8)(20)=1c_3+8c_3$$

$$c_2-9c_3=-160\,(3)$$

$$1c_2+2c_5+8c_3=11c_4$$

$$1c_2+2c_5+8c_3-11c_4=0\,(4)$$

$$3c_1+1c_2=2c_5+2c_5$$

$$3c_1+c_2-4c_4=0\,(5)$$

Se implementó el siguiente código para solucionar el sistema de ecuaciones lineales 5x5, con la ayuda de linalg.solve una función de numpy.

```
import numpy as np
a = np.array([[6,0,-1,0,0],[1,-1,0,0,0],[3,1,0,0,-4],[0,1,8,-11,2],[0,1,-
9,0,0]])
b = np.array([50,0,0,0,-160])
x = np.linalg.solve(a,b)
```

El resultado es:

$$c_1=11.50943396, c_2=11.50943396, c_3=19.05660377, c_4=16.99828473, c_5=11.50943396$$

2. (90 %) Siguiendo el mismo principio que los reactores anteriores, plantee las ecuaciones de balance para los reactores mostrados en la figura 5 y solucionelas usando eliminación Gaussiana.

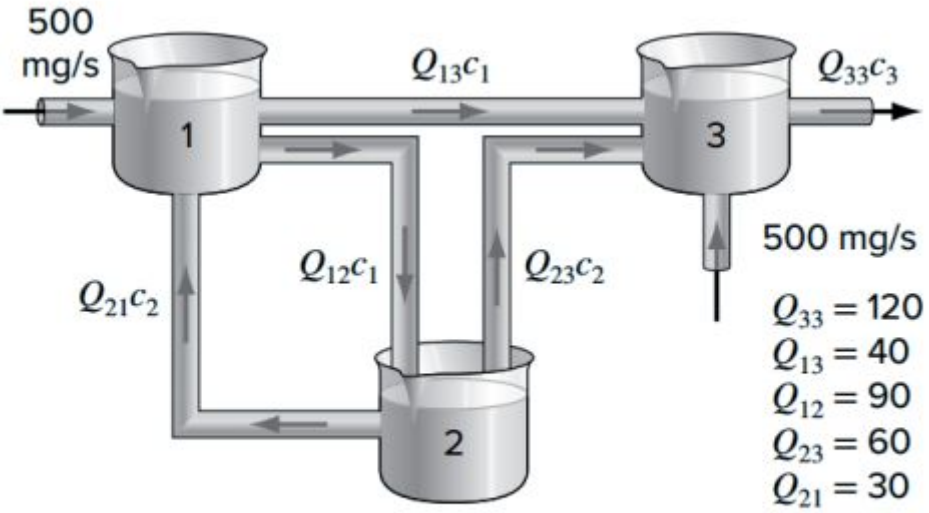


Figura 5: Tres reactores conectados por tuberias.

$$500+30c_2=40c_1+90c_1$$

$$130c_1-30c_2=500(1)$$

$$90c_1=30c_2+60c_2$$

$$c_1=c_2(2)$$

$$40c_1+60c_2+500=120c_3$$

$$40c_1+60c_2-120c_3=-500(3)$$

```
def eliminacionGausian(matrix,vector,m,n):
x =np.zeros((m))
for k in range(0, m):
for r in range(k+1,m):
factor=(matrix[r,k]/matrix[k,k])
vector[r]=vector[r]-(factor*vector[k])
for c in range(0,n):
matrix[r,c]=matrix[r,c]-(factor*matrix[k,c])
#sustitución hacia atrás
x[m-1]=vector[m-1]/matrix[m-1,m-1]
for r in range(m-2,-1,-1):
suma = 0
for c in range(0,n):
```

```
suma=suma+matrix[r,c]*x[c]
x[r]=(vector[r]-suma)/matrix[r,r]
return x

matrix = np.array([[130,-30,0],[1,-1,0],[40,60,-120]],dtype=np.float64)
vector = np.array([500,0,-500],dtype=np.float64)
m = 3
n =3
```

La solución que se encuentra para este ejercicio es la siguiente:

```
x = EliminacionGausiana(matrix,vector)
```

la solución es: c1 = 3.8461538461538463,c2 = -5.773159728050814e-16,c3 = 8.333333333333334

2.4. Descomposición matricial (20 %)

1. (50 %) Tome los reactores de la figura 5 y solucionelo usando factorización LU.

Se aplicó el siguiente código para la factorización LU.

```
import numpy as np
def factorizacionLU(matrix,vector,m):
  x =np.zeros(m)
  u = np.zeros([m,m])
  l = np.zeros([m,m])
  u = matrix
  for k in range(0, m):
    for r in range(0,m):
      if(k==r):
        l[k,r] = 1
      if (k<r):
        factor = (matrix[r,k]/matrix[k,k])
        l[r,k] = factor
        for c in range(0,m):
          matrix[r,c] = matrix[r,c]-(factor*matrix[k,c])
          u[r,c] = matrix[r,c]
```

```
  z = np.linalg.inv(l)@ vector
  #sustitución hacia atrás
  x[m-1] = z[m-1]/u[m-1,m-1]
  for r in range(m-2,-1,-1):
    suma = 0
    for c in range(0,m):
      suma=suma+u[r,c]*x[c]
      x[r]=(vector[r]-suma)/u[r,r]

  return l,u,x

matrix = np.array([[130,-30,0],[1,-1,0],[40,60,-120]],dtype=np.float64)
vector = np.array([500,0,-500],dtype=np.float64)
m = 3
l,u,x = factorizacionLU(matrix,vector,m)
print('la solución es: c1 = {},c2 = {},c3 = {}'.format(x[0],x[1],x[2]))
```

La solución es: c1 = 3.8461538461538463,c2 = -0.0,c3 = 8.333333333333334

Esta solución es igual a la del método de solución Gausssina.

2. (50 %) La parte baja del río colorado consiste en una serie de cuatro almacenamientos conectados como se ilustra en la figura 6.

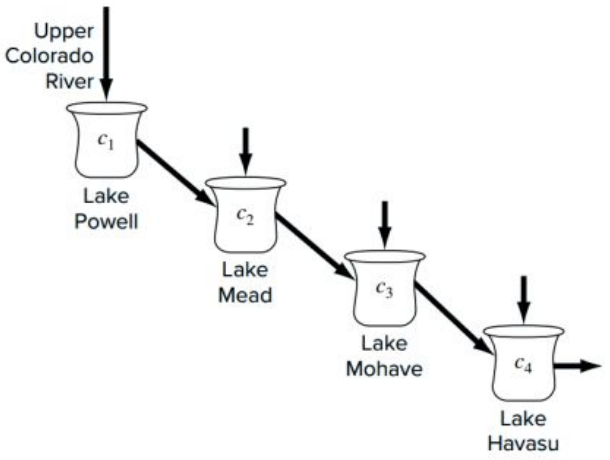


Figura 6: Parte baja del río colorado interconectada con varios lagos

Las ecuaciones de balance para cada reservorio, para unas condiciones específicas están dadas por el siguiente sistema tridiagonal:

$$\begin{bmatrix} 13.422 & 0 & 0 & 0 \\ -13.422 & 12.252 & 0 & 0 \\ 0 & -12.252 & 12.377 & 0 \\ 0 & 0 & -12.377 & 11.797 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 750.5 \\ 300 \\ 102 \\ 30 \end{bmatrix}$$

donde la parte derecha representa la carga de cloro descargada en cada lago y c_i la concentración en cada lago. Con esto en mente:

(a) (25 %) Solucione el sistema tridiagonal mediante un código implementado en **Matlab/Octave o Pyhton** para encontrar la concentración en cada lago.

Se soluciono el sistema, con la matriz tridiagonal utilizando el siguiente codigo:

```
def TDMA(a,b,c,d):
    a = a.astype('double')
    b = b.astype('double')
    c = c.astype('double')
    d = d.astype('double')
    n=np.shape(a)[0]
    x=np.zeros(n)
    c[0]=c[0]/b[0]
    for i in np.arange(1,n-1,1):
      c[i]=c[i]/(b[i]-a[i]*c[i-1])
    d[0]=d[0]/b[0]
    for i in np.arange(1,n,1):
      d[i]=(d[i]-a[i]*d[i-1])/(b[i]-a[i]*c[i-1])
    x[n-1]=d[n-1]
    for i in np.arange(n-2,-1,-1):
      x[i]=d[i]-c[i]*x[i+1]
    return x
```

Se ejecutó esta función con los parámetros de la matriz del ejercicio de la siguiente forma:

```
a = np.array([0,-13.422,-12.252,-12.377])
b = np.array([13.422,12.252,12.377,11.797])
c = np.array([0,0,0,0])
d = np.array([750.5, 300, 102, 30])
```

x = TDMA(a,b,c,d)

Obteniendo los siguientes resultados:

la solución es: c1 = 55.91566085531217,c2 = 85.74110349330721,c3 = 93.11626403813524,c4 = 100.23734847842671

(b) (25 %) ¿Cuánto se debe reducir la carga de cloro en el Lago Powell de modo quue la concentración en el Lago Havasu sea de 75?

$$13.422c_1 = 750.5$$

$$-13.422c_1 + 12.252c_2 = 300$$

$$-12.252c_2 + 12.377c_3 = 102$$

$$-12.377c_3 + 884.775 = 30$$

$$c_3 = \frac{30 - 884.775}{-12.377}$$

$$c_3 = 69.06156580754626$$

$$c_2 = \frac{102 - 12.377c_3}{-12.252}$$

$$c_2 = 61.44098922624878$$

$$c_1 = \frac{300 - 12.252c_2}{-13.422}$$

$$c_1 = 33.733795261510956$$

$$55.91566085531217 - 33.733795261510956 = 22.181865593801213$$

Se debe disminuir 22.181865593801213 de lago Powell para que llegue 75 al lago Havasu 75

2.5. Métodos iterativos (20 %)

1. (100 %) Solucione los problemas de las figuras 4 y 5 usando los métodos de Jacobi y Gauss-Seidel. Compare y analice los resultados.

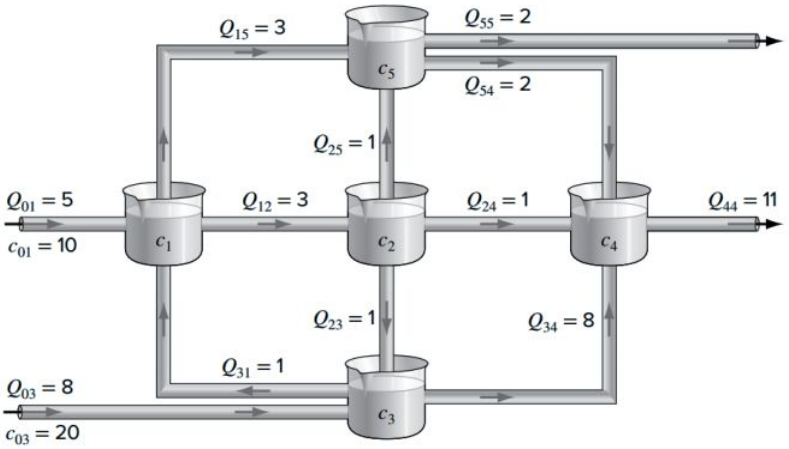


Figura 4: Sistema de reactores interconectados

El flujo másico a través de cada tubo se calcula como el producto del caudal (Q) y la concentración (c) de constituyente. En estado estacionario, el flujo másico de entrada y salida de cada reactor debe ser igual. Por ejemplo, para el primer reactor se tiene que el balance de masa se puede escribir como:

$$Q_{01}c_{01} + Q_{31}c_3 = Q_{15}c_1 + Q_{12}c_1$$

Escriba las ecuaciones para cada reactor y use las funciones de **Matlab/Octave o Pyhton** para resolver el sistema.

$$(5)(10)+1(c_3)=3c_1+3c_1$$

$$6c_1-c_3=50\,(1)$$

$$3c_1=1c_2+1c_2+c_2$$

$$c_1-c_2=0\,(2)$$

$$1c_2+(8)(20)=1c_3+8c_3$$

$$c_2-9c_3=-160\,(3)$$

$$1c_2+2c_5+8c_3=11c_4$$

$$1c_2+2c_5+8c_3-11c_4=0\,(4)$$

$$3c_1+1c_2=2c_5+2c_5$$

$$3c_1+c_2-4c_4=0\,(5)$$

Para el método de Gauss-Seidel se utilizó el siguiente código: El resultado siguiente se obtuvo con 20 interacciones

```
  for i in range (0,n):
    dummy = a[i][i]
    for j in range (0,n):
      a[i][j] = a[i][j] /(dummy+1e-6)
    b[i]= b[i]/(dummy+1e-6)

  for i in range (0, n):
    suma = b[i]
    for j in range (0, n):
      if (i != j):
        suma = suma - (a[i][j] *x[j])
      x[i] = suma
  inm=1
  centinela = 0
  while(centinela == 1 or (inm <= imax)):
    centinela = 1
    for i in range (0,n):
      old = x[i]
      suma = b[i]
      for j in range (0,n):
        if(i != j):
          suma = suma-a[i][j]*x[j]
      x[i] = lamda*suma +(1-lamda)*old

      if(centinela == 1 and x[i] != 0):

        ea = abs(x[i] - old)/abs(x[i])*100

        if (ea < es):
          centinela = 0

    inm+=1

  return x
```

Con este método en particular no converge la solución.

la solución es: c1 = 8.0,c2 = 8.0,c3 = -4.277600748103214e+239,c4 = 0.0,c5 = -3.8498406733025174e+245

Para el método de Jacobi se implementó el siguiente codigo:

```
 def jacobi(A,b,x0,tol,N):
    A = A.astype('double')
    b = b.astype('double')
    x0 = x0.astype('double')

    n=np.shape(A)[0]
    x = np.zeros(n)
    it = 0
    #iteracoes
    while (it < N):
        it = it+1
        #iteracao de Jacobi
        for i in np.arange(n):
            x[i] = b[i]
            for j in np.concatenate((np.arange(0,i),np.arange(i+1,n))):
                x[i] -= A[i,j]*x0[j]
            x[i] /= A[i,i]
        #tolerancia
        if (np.linalg.norm(x-x0,np.inf) < tol):
            return x
        #prepara nova iteracao
        x0 = np.copy(x)

    return x
```

No converge con el método de Jacobi. El resultado siguiente se obtuvo con 20 interacciones

la solución es: c1 = -3.1972573039030914e+122,c2 = -1.5045932253683726e+122,c3 = -3.2499186133895834e+136,c4 = -1.4772372833374157e+129,c5 = -1.7265192318608258e+130

2. (90 %) Siguiendo el mismo principio que los reactores anteriores, plantee las ecuaciones de balance para los reactores mostrados en la figura 5 y solucionelas usando eliminación Gaussiana.

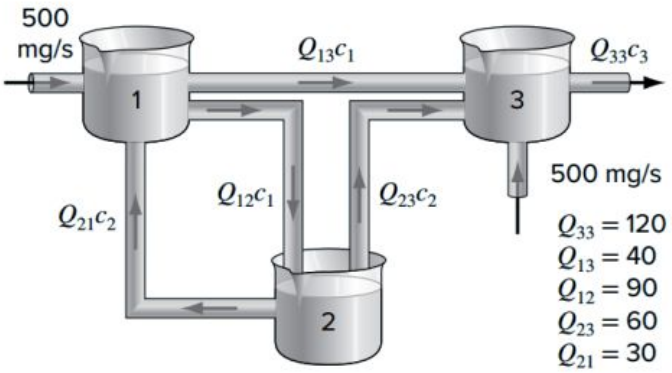


Figura 5: Tres reactores conectados por tuberias.

Gauss-seidel

la solución es: c1 = 5.000001450001911,c2 = 5.000006450008356,c3 = 8.333337111115949

Jacobi

la solución es: c1 = 4.9999907195053215,c2 = 4.9999597845230594,c3 = 8.333299820435883