CENTRO DE ENSEÑANZA TÉCNICA Y SUPERIOR

Escuela de Ingeniería

Métodos Numéricos

Actividad:
Proyecto Segundo Parcial

Presenta:
Elian Javier Cruz Esquivel

Matrícula: T032218

Tijuana, B.C., a jueves 29 de octubre de 2020

CETYS UNIVERSIDAD

CENTRO DE ENSEÑANZA TÉCNICA Y SUPERIOR
ESCUELA DE INGENIERÍA

Métodos Numéricos
Elian Javier Cruz Esquivel T032218

Proyecto Segundo Parcial

Introducción

Dentro de la solución de sístemas de ecuaciones lineales existe una variedad de métodos conocidos como iterativos debido a que su funcionamiento para determinar las soluciones al sístema depende de la iteración, de acuerdo a Burden y Faires (2002), a diferencia de los métodos directos los métodos iterativos permiten la obtención de aproximaciones a las soluciones de sístemas lineales con una gran precisión, incluso si estos poseen una gran cantidad de ceros.

La aplicación de estos sistemas no suele estar en sístemas pequeños, puesto que es necesario mas tiempo para encontrar una aproximación satisfactoria. Según Burden y Faires (2002), los métodos iterativos comienzan con una aproximación inicial a la solución del sístema, a partir de la cual mediante la modificación del sístema matricial $Ax = b$ es convertido iterativamente en otro, haciendo que su solución converga a la del sistema original.

Entre los métodos iterativos mas reconocidos se encuentran los denominados como Gauss-Seidel y de Jacobi, ambos utilizan la factorización de la matriz A en sus componentes L, U y D, siendo respectivamente la matriz triangular inferior, superior y diagonal. A partir de ellas, y mediante el respectivo procedimiento de cada uno de los métodos, se obtiene una aproximación a partir de la aproximación inicial, una segunda iteración a partir de la cual se itera de la misma forma. Sin embargo, existe la limitante que estos métodos son solo aplicables a matrices cuya diagonal sea estrictamente dominante.

De igual forma, existen métodos mediante los cuales la principal ventaja es que no se requiere de demasiadas iteraciones, como en el caso de la factorización LU por el método de Doolittle, a partir de la cual, de acuerdo a Chapra (2015), se generan las matrices L y U mediante un paseo por las filas y columnas de la matriz inicial A, al descomponer a esta última en las dos primeras, es posible generar dos sístemas lineales más sencillos de resolver de forma individual. Al resolver un sístema y utilizar su resultado como parte de la solución del segundo se obtiene la solución del sístema original.

Por último, tenemos los métodos mediante los cuales es posible generar funciones que aproximen los valores de una colección de puntos con el menor error posible, de acuerdo a Gergonne (1974) se utiliza el método de mínimos cuadrados para obtener el menor error posible entre la función y la colección de puntos, dicho método permite que el cuadrado de los errores de aproximación sea el mínimo, para esto, se utilizan sístemas matriciales que permiten expresar las colecciones de puntos como vectores x e y a partir de los cuales se determina qué tipo de regresión es la mejor para la colección, sea ésta lineal, cuadrática, exponencial, logarítmica, potencial o incluso polinómica,

Problema 1

Dado el siguiente Sistema de Ecuaciones Lineales.

$$\begin{array}{rcrcrcrcl} 4x_1 & - & x_2 & + & x_3 & & & = & 7 \\ 4x_1 & - & 8x_2 & + & x_3 & + & 2x_4 & = & -23 \\ -2x_1 & + & x_2 & + & 5x_3 & - & x_4 & = & 16 \\ x_1 & - & 4x_2 & + & 3x_3 & + & 10x_4 & = & -15 \end{array}$$

Use los Métodos de Jacobi y Gauss-Seidel, y realice lo siguiente:

a) Calcule los primeros 15 términos de la sucesión generada a partir del punto $P_0 = (0, 2, 1, 1)$ utilizando los algoritmos antes mencionados

b) De un análisis comparativo de los dos algoritmos, según sus resultados.

Para comenzar,podemos ver que la matriz tiene una diagonal estrictamente dominate, por lo tanto ambos métodos son aplicables. Tenemos que el método de Jacobi nos permite descomponer la matriz en sus tres matrices distintas, siendo estas L, U y D. Dichas matrices nos permiten hacer que el sístema $Ax = b$ se convierta en lo siguiente:

$$Ax = b \Rightarrow (D + L + U)x = b$$

$$Dx = (-L - U)x + b \Rightarrow x = D^{-1}(-L - U)x + D^{-1}b$$

Por lo tanto:

$$x^{(k+1)} = D^{-1}(-L - U)x^k + D^{-1}b$$

Si $R = -L - U$:

$$x^{(k+1)} = D^{-1}Rx^k + D^{-1}b$$

Entonces, podemos ver que la $k + 1$-ésima iteración depende directamente de la inversa de la matriz diagonal D y de la aproximación anterior, es decir, la k-ésima iteración. Con esto en mente, podemos proceder a explicar el código utilizado para calcular la solución del problema planteado.

Tenemos que el sístema puede ser representado en forma matricial como:

$$A = \begin{bmatrix} 4 & -1 & 1 & 0 \\ 4 & -8 & 1 & 2 \\ -2 & 1 & 5 & -1 \\ 1 & -4 & 3 & 10 \end{bmatrix}, \quad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad b = \begin{bmatrix} 7 \\ -23 \\ 16 \\ -15 \end{bmatrix}$$

A partir de esto, procedemos a explicar el código para posteriormente mostrar la solución:

```python
def JacobiOne():
    print("Método de Jacobi")
    x_i = np.matrix([[0],[2],[1],[1]])
    A = np.matrix([[4,-1,1,0],[4,-8,1,2],[-2,1,5,-1],[1,-4,3,10]])
    b = np.matrix([[7],[-23],[16],[-15]])
    D = np.diag(np.diag(A))
    Dinv = np.linalg.inv(np.diag(np.diag(A)))
    R = A-D
    def printJacobi():
        print("R:")
        print(R)
        print("D:")
        print(D)
    printJacobi()
    real = [2,4,3,-1]
    error = 1
    for i in range(15):
        x = Dinv*(b-R*x_i)
        x_i = x
        error = math.sqrt((x_i.item(0)-real[0])**2 + (x_i.item(1)-real[1])**2 + (x_i.item(2)-real[2])**2+ (x_i.item(3)-real[3])**2)
        print("a: ",x_i.item(0)," b: ",x_i.item(1), " c: ",x_i.item(2), " d: ",x_i.item(3), " Error: ","{:3e}".format(error))
    print("Iteraciones: ", i)
    print("a:",x_i.item(0)," b:",x_i.item(1), " c:",x_i.item(2), " d:",x_i.item(3), "con un error absoluto de: ", error)
```

Figura 1: Código para el método de Jacobi

Tenemos a la función JacobiOne, dentro de la cual declaramos la matriz x_i, siendo esta la primera aproxiamción dada por el problema, así tambien, declaramos la matriz A y b con los datos del problema. Posteriormente calculamos la matriz diagonal D de A mediante la función $np.diag(np.diag(A))$. Decimos que D_inv es igual a la inversa de la diagonal de A y por último, hacemos que R sea igual a la matriz original A menos los elementos de la diagonal D, quedando en R los triangulos superiores e inferiores.

Posteriormente creamos la lista $real$ con los valores reales obtenidos mediante la utilización del software OctaveOnline. A partir de aqui, comenzamos con un ciclo for en el que hacemos 15 iteraciones; dentro de él tenemos que la x actual es igual a $D_inv(b - Rx_i)$, tal y como se mencionó al principio de la explicación del problema, dicha variable x es la nueva aproximación a la solución del sístea; despues hacemos que $x_i = x$ para que la siguiente iteración utilice los datos obtenidos en la actual, por último calculamos el error mediante la siguiente fórmula:

$$error = \sqrt{(x_{i_0} - x_{real0})^2 + (x_{i_1} - x_{real1})^2 + (x_{i_2} - x_{real2})^2 + (x_{i_3} - x_{real3})^2}$$

Y finalmente imprimimos los resultados, los cuales son los siguientes:

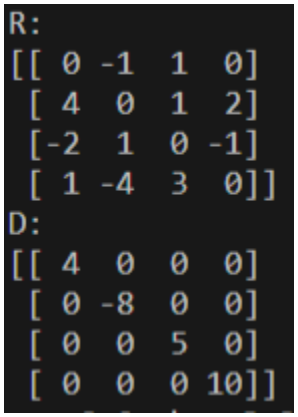

Figura 2: Matrices R y D

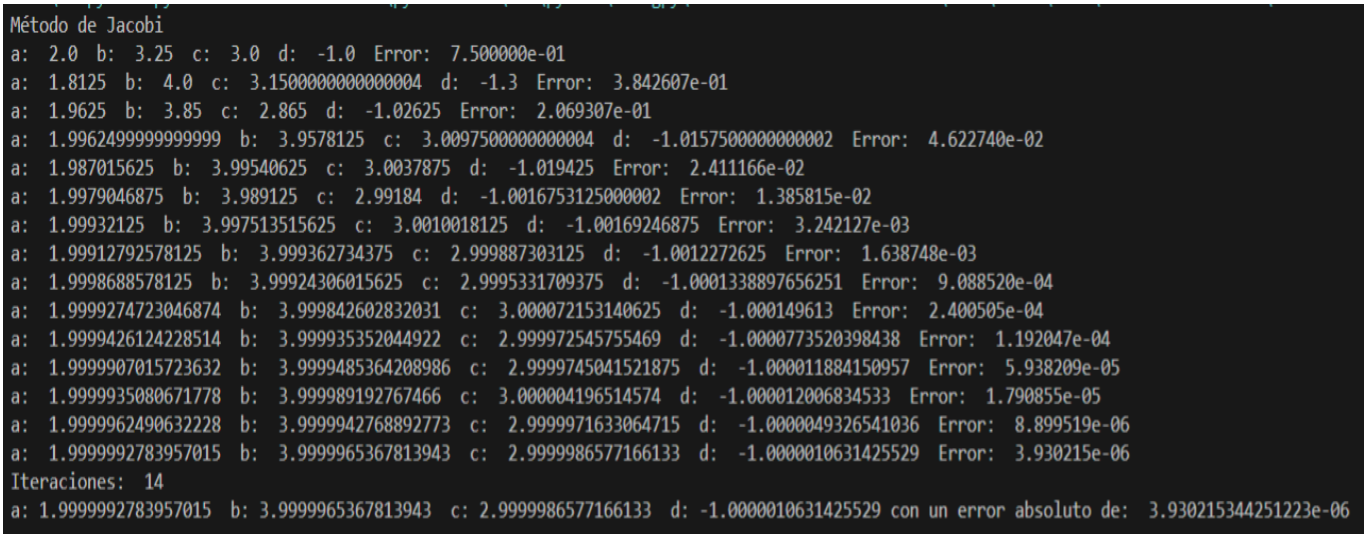

Figura 3: Resultados del método de Jacobi

Podemos ver que las soluciones son

$$x_0 = 1.99999 \approx 2, \;\; x_1 = 3.999999 \approx 4, \;\; x_2 = 2.99999 \approx 3, \;\; x_3 = -1.00000 \approx -1$$

Y que se logra un error de grado 10^{-6} con las 15 iteraciones.

Para utilizar el método de Gauss-Seidel tenemos que

$$(D+L)x = Ux + b \Rightarrow x = (D-L)^{-1}(b-Ux)$$

por lo tanto:

$$x^{(k+1)} = (D-L)^{-1}(b-Ux^k)$$

Es decir, tenemos que la nueva iterción es igual a la inversa de la matrix diagonal D menos la matriz triangular inferior L, a la vez, multiplicando dicha inversa por el vector b menor la matriz triangula superior U multiplicando a la iteración anterior. A partir de esto tenemos que el código para el método de Gauss-Seidel es el siguiente:

```
def GaussSeidelOne():
    x = np.matrix([[0],[2],[1],[1]])
    A = np.matrix([[4,-1,1,0],[4,-8,1,2],[-2,1,5,-1],[1,-4,3,10]])
    b = np.matrix([[7],[-23],[16],[-15]])
    D = np.diag(np.diag(A))
    L = np.tril(A)
    U = np.triu(A)-D
    DLinverse = np.linalg.inv(L)
    print("\nMétodo de Gauss-Seidel")
    def printGaussSeidel():
        print("D")
        print(D)
        print("L")
        print(L-D)
        print("U")
        print(U)
    printGaussSeidel()
    real = [2,4,3,-1]
    error = 1
    for i in range(15):
        x = DLinverse*(b-U*x)
        error = math.sqrt((x.item(0)-real[0])**2 + (x.item(1)-real[1])**2 + (x.item(2)-real[2])**2+ (x.item(3)-real[3])**2)
        print("a: ",x.item(0)," b: ",x.item(1), " c: ",x.item(2), " d: ",x.item(3), " Error: ","{:3e}".format(error))
    print("Iteraciones: ", i)
    print("a:",x.item(0)," b:",x.item(1), " c:",x.item(2), " d:",x.item(3), "con un error absoluto de: ", error)
```

Figura 4: Código del método de Gauss-Seidel

De la misma forma que para Jacobi, tenemos las matrices x, A y b, calculamos la matriz diagonal D y esta vez si separamos las matrices triangulares superior U e inferior L, calculamos la inversa de la matrix triangular inferior y la diagonal en D_inv.

Posteriormente dentro del ciclo for hacemos que la nueva iteración sea igual a $(D-L)^{-1}(b-Ux^k)$, es decir, la inversa calculada por el vector b menos la matrix triangular superior U multiplicando a la iteración anterior. Despues calculamos el error al igual que en el método de Jacobi y por último imprimimos los resultados, siendo estos:

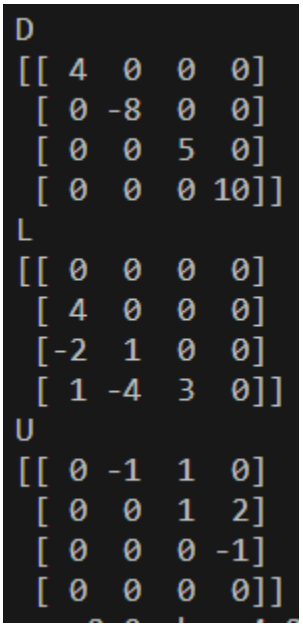

Figura 5: Matrices D, L y U

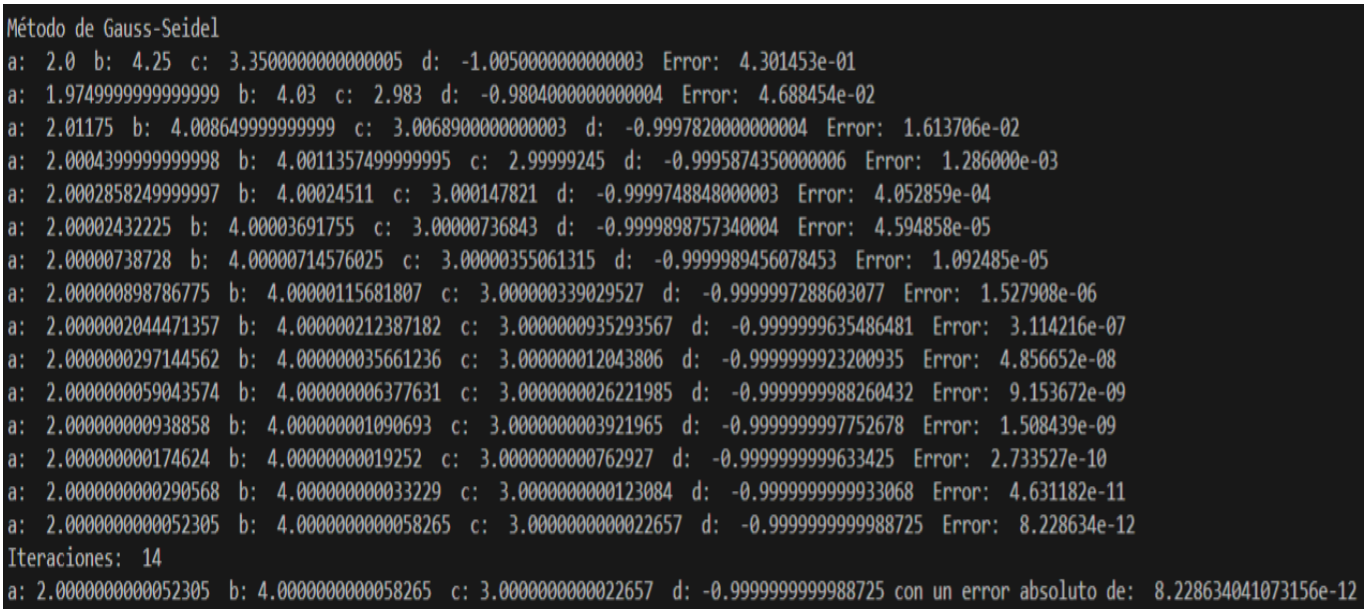

Figura 6: Resultados para el método de Gauss-Seidel

Podemos ver que las soluciones al problema presentan un error de grado 10^{-12}, es decir, es doblemente mas certero en la misma cantidad de iteraciones, lo que nos permite asumir que el método de Gauss−Seidel generalmente converge más rapido que el método de Jacobi.

Problema 2

Amanda, Bryce y Corey entran a un evento atlético, en el cual tienen que correr, nadar y andar en bicicleta a lo largo de una ruta trazada. Sus velocidades promedio se proporcionan en la Figura 1. Corey termina primero con un tiempo total de 1h 45 min. Amanda llega en segundo lugar con un tiempo de 2h 30 min. Bryce termine al último con un tiempo de 3h. Determine la distancia en millas para cada parte de la carrera. Utilizar la Descomposición LU para resolver el problema.

	Velocidad promedio (millas/h)		
	Carrera	Natación	Ciclismo
Amanda	10	4	20
Bryce	$7\frac{1}{2}$	6	15
Corey	15	3	40

Figura 1: Velocidades

Sabemos que el método de descomposición LU por Doolittle permite generar una descomposión de la matriz A del sistema $Ax = b$ tal que:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

De tal modo que podamos obtener los valores de L y U de tal forma que:

$$a_{11} = u_{11}, \quad a_{12} = u_{12}, \quad a_{13} = u_{13}$$
$$a_{21} = l_{21}u_{11}, \quad a_{22} = u_{12}l_{21} + u_{22}, \quad a_{23} = u_{13}l_{21} + u_{23}$$
$$a_{31} = l_{31}u_{11}, \quad a_{32} = u_{12}l_{31} + l_{32}u_{22}, \quad a_{33} = l_{31}u_{13} + l_{32}u_{23} + u_{33}$$

De esta forma podemos descomponer A de tal forma que

$$Ax = b \Rightarrow (LU)x = b$$

Si decimos que $y = Ux$ entonces tenemos dos sístemas:

$$Ly = b, \quad Ux = y$$

Cuya representación matricial está dada por

$$\begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Utilizando la eliminación Gaussiana tenemos que las soluciones al vector y están dadas por las siguientes ecuaciones:

$$\begin{aligned} y_1 &= b_1 \\ y_1 l_{21} + y_2 = b_2, &\quad y_2 = b_2 - y_1 l_{21} \\ y_1 l_{31} + y_2 l_{32} + y3 = b_3, &\quad y_3 = b_3 - y_1 l_{31} - y_2 l_{32} \end{aligned}$$

A partir de lo cual podemos econtrar las soluciones al sístema $Ux = y$ de tal forma que:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Por lo tanto, las soluciones están dadas por una sustitución hacia atras de tal forma que nos quede lo siguiente:

$$\begin{aligned} x_3 u_{33} = y_3, &\quad x_3 = \frac{y_3}{u_{33}} \\ x_2 u_{22} + x_3 u_{23} = y_2, &\quad x_2 = \frac{y_2 - x_3 u_{23}}{u_{22}} \\ x_1 u_{11} + x_2 u_{12} + x_3 u_{13} = y_1, &\quad , x_1 = \frac{y_1 - x_2 u_{12} - x_3 u_{13}}{u_{11}} \end{aligned}$$

Ahora que ya sabemos como obtener las soluciones al sístema de ecuaciones lineales, debemos de plantear nuestro sístema y establecerlo en forma matricial. Si decimos que la distancia recorrida en carrera en x_0, en natación es x_1 y ciclismo es x_2 y sabiendo que $v = \frac{d}{t}$, tenemos que encontrar un sístema $Ax = b$, para el cual x es igual a las distancias, por lo tanto, la matriz A debe de representar $\frac{1}{v}$, forma deducida de $t = (d)(\frac{1}{v}) = \frac{d}{v}$, entonces:

$$\begin{bmatrix} \frac{1}{10} & \frac{1}{4} & \frac{1}{20} \\ \frac{1}{7.5} & \frac{1}{6} & \frac{1}{15} \\ \frac{1}{15} & \frac{1}{3} & \frac{1}{40} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 3 \\ 1.75 \end{bmatrix}$$

Entonces, siguiendo el procedimiento descrito anteriormente tenemos que el código para la descomposición LU por Doolittle es el siguiente:

```
def TwoLU():
    n=3
    A = np.matrix([[1/10,1/4,1/20],[1/7.5,1/6,1/15],[1/15,1/3,1/40]])
    b = np.matrix([[2.5],[3],[1.75]])
    L = np.zeros((n,n))
    U = np.zeros((n,n))
    def LU():
        for j in range(n):
            for i in range(n):
                if i <= j:
                    U.itemset((i,j),A.item((i,j)))
                    for k in range(i):
                        U.itemset((i,j),U.item((i,j))-L.item((i,k))*U.item((k,j)))
                if (j<=i):
                    L.itemset((i,j),A.item((i,j)))
                    for k in range(j):
                        L.itemset((i,j),L.item((i,j))-L.item((i,k))*U.item((k,j)))
                    L.itemset((i,j),L.item((i,j))/U.item((j,j)))

    def Lyb():
        y = [0 for _ in range(n)]
        x = [0 for _ in range(n)]
        for i in range(n):
            res = b.item(i)
            for j in range(n):
                res -= y[j]*L.item((i,j))
            y[i] = res
        k = 0
        for i in range(n-1,-1,-1):
            res = y[i]
            for j in range(n-k,n):
                res -= x[j]*U.item((i,j))
                #print(x[j],U.item((i,j)))
            x[i] = res/U.item((i,i))
            k+=1
```

Figura 7: Código para descomposición LU por Doolittle

Para comenzar, tenemos que $n = 3$ debido a que nuestro sístema es de 3×3; declaramos las matrices A, b, L y U, siendo estas últimos dos matrices cero.

Entonces mandamos llamar la función $LU()$, dentro de ésta tenemos que se calculan los elementos de las matrices L y U, tenemos dos ciclos anidados con índices j e i de tal forma que podamos recorrer filas y columnas de manera correcta. Dentro del segundo for tenemos que existe la condición $i \leq j$, esto es debido que dentro de la matriz U, todos los coeficientes debajo de la diagonal principal son ceros, y dichos elementos tienen la caracteristica de tener su índice de columna j mayor o igual al índice de su fila, por lo tanto no los recorremos en la iteración.

Posteriormente se encuentra el cálculo de los coeficientes de L, cuyo procedimiento es igual al de U en el sentido que hacemos que el elemento a calcular tome el valor de A en esa posición y consecutivamente le restamos la multiplicación de fila por columna de su respectiva posición de la forma establecida al principio del problema. En etse caso, queremos que $i \leq j$ debido que el triángulo superior de la matriz está lleno de ceros.

Por último, tenemos la función $Lyb()$, dentro de ella calculamos a x e y mediante sustitución hacia atras y hacia adelante respectivamente, el funcionamiento de ambas sustituciones ya ha sido detallada anteriormente, el único cambio significativo es el hecho que se tuvo que agregar un apuntador extra denominado k el cual nos dice hasta qué fila tenemos que iterar en la sustitución hacia atras.

Por último imprimimos los resultados, los cuales son los siguientes:

```
L
[[ 1.          0.          0.        ]
 [ 1.33333333  1.          0.        ]
 [ 0.66666667 -1.          1.        ]]
U
[[ 0.1         0.25        0.05      ]
 [ 0.         -0.16666667  0.        ]
 [ 0.          0.         -0.00833333]]
y
[2.5, -0.33333333333333304, -0.24999999999999956]
x
[5.000000000000027, 1.9999999999999982, 29.999999999999954]
```

Figura 8: Resultados de la descomposición LU

Podemos ver que el método nos dice que las distancias son las siguientes:
La sección de carrera se realiza en una distancia de 5 millas, la sección de natación tiene una longitud de 2 millas y la sección de ciclismo tiene una longitud de 30 millas.

Problema 3

Un químico tiene una muestra de 100 gramos de material radiactivo. El registra la cantidad de material radiactivo cada semana y obtiene los siguientes datos por 7 semanas:

Semanas	0	1	2	3	4	5	6
Peso (gramos)	100	88.3	75.9	69.4	59.1	51.8	45.5

Figura 2: material radiactivo

a) Elabore un diagrama de dispersión para los datos

b) Determine la ecuación de un modelo exponencial $y = ae^{bx}$. Graficar

c) Determine el material radiactivo después de 50 semanas.

d) Determine el número de semanas que deben pasar para que quede 20 gramos de material.

Sabemos que las semanas representan los valores de x y que y está representada por el peso. Sabemos que un modelo exponencial está dado por la ecuación $y = ae^{bx}$, por lo tanto, si buscamos despejar x tenemos que:

$$Ln(y) = Ln(ae^{bx}) \Rightarrow Ln(y) = Ln(a) + bx$$

Entonces, expresandolo en forma matricial tenemos que:

$$\begin{bmatrix} ln(y_0) \\ ln(y_1) \\ \vdots \\ ln(y_n) \end{bmatrix} = \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} ln(a) \\ b \end{bmatrix}$$

Si lo expresamos con variables podemos decir que $\vec{y} = A\vec{v}$, entonces, multiplicando ambos lados por la transpuesta de A para aislar $\vec{v}$ tenemos que:

$$A^T\vec{y} = A^T A\vec{v}$$

Es decir:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_0 & x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} ln(y_0) \\ ln(y_1) \\ \vdots \\ ln(y_n) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_0 & x_1 & \dots & x_n \end{bmatrix} \begin{bmatrix} 1 & x_0 \\ 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} ln(a) \\ b \end{bmatrix}$$

Haciendo la multiplicación de las matrices tenemos:

$$\begin{bmatrix} \sum ln(y_i) \\ \sum ln(y_i)x \end{bmatrix} = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} ln(a) \\ b \end{bmatrix}$$

Por lo tanto, las soluciones al problema están dadas por $(A^TA)^{-1}(A^T\vec{y})$. El código de regresión exponencial está dado a continuación:

```
def generalizeExponential(n,x,y):
    yLn = np.log(y)
    res = regressionGeneralized(n,x, yLn, True, False)
    yAsArray = [y.item(i) for i in range(np.prod(y.shape))]
    s = res[4]
    def error():
        error = 0
        getEstimation = lambda v: math.e**(s.item(0)) * math.e**(v*s.item(1))
        for i in range(n):
            result = (getEstimation(x.item(i))-y.item(i))
            error+=result
        print("error:", math.sqrt(error**2/n))
    error()
    plot(res[0], yAsArray, res[2], res[3], res[2].min()-1, res[3].min()-1, res[2].max()+1, res[3].max()+1)
    getEstimation = lambda v: math.e**(s.item(0)) * math.e**(v*s.item(1))
    print(getEstimation(50))
    res = lambda v: math.log1p(v-1)
    getEstimationY = lambda v:(res(v) - res(math.e**s.item(0)))/s.item(1)
    print(getEstimationY(20))
```

Figura 9: Código para regresión exponencial

En primer lugar, tenemos a la función *generalizeExponential*(), la cual recibe como parámetros la longitud de la matriz (n), los vectores x e y. A partir de aqui tenemos que utilizar los valores de los logaritmos de y para utilizarlos en la matriz, a partir de ella llamamos a la función *regressionGeneralized*() la cual acepta como parámetros n, x y los logaritmos de y, una vez que la función es ejecutada se retorna una lista con los valores a graficar para x e y correspondientes a la nueva función exponencial, así tambien se ejecuta la función error la cual obtiene la suma de los errores cuadrados de las estimaciones hechas con la ecuación encontrada y el punto y dado por el problema. Posteriormente se manda llamar a la función graficadora (*plot*) y por último so obtienen las estimaciones para alguna x y para alguna y.

En la función del error, se obtiene a raíz de los errores al cuadrado divididos entre el numero de puntos que se han utilizado en la regresión.

En la siguiente página se encuentra la descripción de la función *regressionGeneralized*()

```
def regressionGeneralized(n, x, y, isExp, isPot):
    A = zeros((n,n))
    b = zeros((n,1))
    xAsArray = [x.item(i) for i in range(np.prod(x.shape))]
    yAsArray = [y.item(i) for i in range(np.prod(y.shape))]
    def Afunc(last, i):
        for exp in range(n**2):
            if (exp%n==0 and exp!=0):
                last = exp-1-i
                i+=1
            aux = 0
            for x_i in xAsArray:
                aux += x_i**(exp - last)#
            A.itemset(exp, aux)
    def bfunc():
        for exp in range(n):
            temp = [(a**exp) * b for a, b in zip(xAsArray, yAsArray)]
            aux = sum(temp)
            b.itemset(exp, aux)
    Afunc(0,0)
    bfunc()
    s = np.matmul(np.linalg.inv(A),b)
    print("a:",math.e**s.item(0))
    print("b:", s.item(1))
    xPlotter = [x.item(i) for i in range(np.prod(x.shape))]
    yPlotter = [y.item(i) for i in range(np.prod(y.shape))]
    newX = np.linspace(-25,100)#np.linspace(int(x.min())-1,int(x.max()
    newY = 0
    if isExp==True:
        newY = math.e**(s.item(0)) * math.e**(newX*s.item(1))
        return [xPlotter, yPlotter,  newX, newY, s]
```

Figura 10: Código para regresión exponencial

Tenemos que acepta los parámetros enviados por la función $generalizedExponential()$, es decir, n, los valores de x y de los logaritmos de y. Inicializamos las matrices A y b con ceros, hacemos que los valores de x y los logaritmos de y tomen forma de lista para su manipulación mas sencilla.

Mandamos llamar la función $Afunc()$ en la cual calculamos la matriz A, es decir, recorremos cada fila aumentando un exponente a la respectiva suma de x's dependiendo de la fila en la que se

encuentra, de esta forma calculamos n, $\sum x_i$ y $\sum x_i^2$. Una vez hecho eso calculamos el vector $\vec{y}$ de una forma tal que cada fila sea igual a la suma de la miltiplicación del logaritmo natural de y_i por x_i elevado al exponente correspondiente a la fila

Una vez hecho esto hacemos $\vec{v} = A^{-1}\vec{y}$, el cual nos da la matriz s de dimensiones 2×2, posteriormente imprimimos los valores, generamos los valores de los puntos dados por el problema en forma de lista para generar su grafico de dispersión, los cuales son $xPlotter$ y $yPlotter$, establecemos el rango de la gráfica como $newX$ y por último hacemos que $newY$ tome los valores de la nueva función con los puntos de $newX$, los valores de la nueva función están en s, sin embargo se nos da el $ln(a)$, por lo tanto, para encontrar a tenemos que $e^{lna} = a$, lo que nos deja con la ecuación $e^{s(0)}e^{s(1)x}$.

Por último retornamos los valores para los gráficos de dispersión y los valores para graficar la nueva función (tanto de x como de y), por último tenemos la gráfica y los resultados, los cuales son: Entonces la ecuación de la regresión exponencial es la siguiente:

```
a: 100.32625084061956
b: -0.1314021163132879
error: 0.0008209581996106567
Material despues de 50 semanas: 0.14062219797914957
Semanas para que queden 20 gramos de material: 12.272976689065333
```

Figura 11: Resultados de la regresión exponencial

$$y = 100.32625084061956e^{-0.1314021163132879x}$$

Y su gráfica está en la siguiente página:

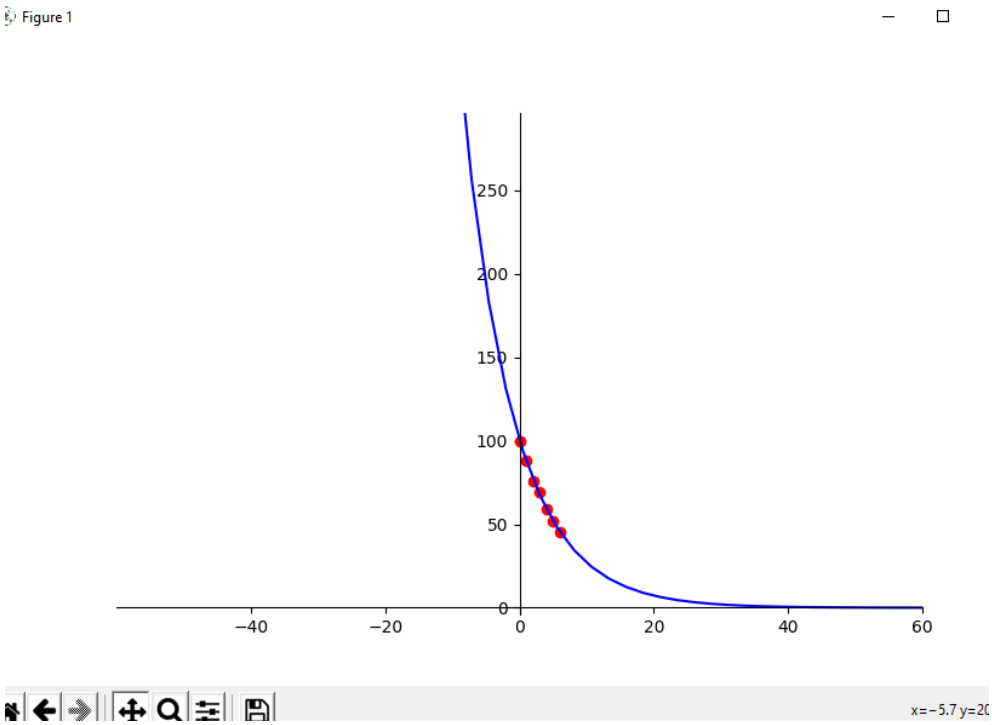

Figura 12: Gráfica de regresián exponencial y dispersión

Por último, para calcular el material radioactivo despues de 50 semanas tenemos:

$$100.32625084061956e^{-0.1314021163132879(50)} = 0.14062219797914957$$

Por lo que se tienen 0.14062219797914957 gramos despues de 50 semanas
Y para calcular cuantas semanas son necesarias para tener 20 gramos de material radioactivo es necesario hacer lo siguiente:

$$\frac{ln(20) - ln(100.32625084061956)}{-0.1314021163132879} = x = 12.272976689065333$$

Entonces se requieren 12.3 semanas para tener 20 gramos de material radioactivo.

Conclusión

Podemos ver que claramente los métodos numéricos son bastante utiles para el cálculo de soluciones de situaciones muy diversas, su aplicación no se centra simplemente en situaciones especificas, sino que puede ser generalizado para ser utlizado en cualquier momento, por ejemplo, la regresión permite encontrar funciones que se ajusten con el menor error a conjuntos de puntos, siendo estas funciones de distintas formas, cuyo calculo puede generalizarse bajo un mismo algoritmo en el cual lo único que cambia es el formato de entradas y salidas, haciendo que los cálculos y complejidad computacional se vea reducida.

En cuanto a las matrices y las formas en que los sistemas de ecuaciones lineales pueden ser resueltos son de suma importancia para nosotros como ingenieros en ciencias computacionales sobre todo porque nuestro trabajo requiere del analisis y desarrollo de algoritmos eficientes y sencillos para la resolución de problemas, y dado que la mayoria de los problemas consisten en distintas etapas de los mismos valores, es posible utilizar estos métodos para encontrar o aproximar soluciones a dichos problemas.

Un ejemplo concreto es el hecho del cálculo de corrientes y voltajes, pues tal y como vimos durante las presentaciones, es posible calcularlos mediante los métodos vistos, lo que facilita el trabajo, pues antes de analizar dichos métodos, durante mis años de bachiller, el mismo problema lo resolvía mediante métodos algebráicos o mediante matrices expandidas, pero siempre a papel y lápiz, por lo que saber programar estos procedimientos de forma sencilla y poder utilizarlos para resolver problemas de forma sencilla es bastane útil y satisfactorio.

Referencias

Burden, R. y Faires, J. (2002). *Análisis Numérico 7ma edición*. México: Thomson Learning

Chapra, C., y Raymond, P. (2015). *Numerical Methods for Engineers*. Estados Unidos: McGraw-Hill Education.

Gergonne, J. D. (1974). *The application of the method of least squares to the interpolation of sequences*. Historia Mathematica. 1(4). pp. 439–447.