Facultad de Estudios Superiores

Acatlán

Matemáticas Aplicadas y Computación.

Proyecto Final.

Arriaga Vázquez Luis Fernando

García Pérez Salomón

Orozco Pacheco Moisés

Ramírez Valdespino Alan

Método de Newton

$def\ \ Método\ Newton(N, f, a, cen)$

Función General: Realiza el método de Newton de manera lineal, evalúa la función y su derivada en una iteración inicial con valores del intervalo de inicio, y posteriormente con sus aproximaciones.

Parámetros:

N : Número máximo de iteraciones, definido por $def\ \ Número\ Iteraciones$

f : Función en la que el método será aplicado.

a : Valor inicial.

cen : Valor centinela, que valida, que la función sea trigonométrica o no.

Se define n como la variable iterativa.

- Iteración $n = 0$
  1. Encontramos $f'(x)$
  2. Se inicializan las siguientes variables:

variable	significado
$var = a$	$x_0 = a$
$var1 = 0$	Valida la que la raíz anterior no se haya repetido
$error = 0$	ε

- Iteración $n = 1,\ n = 2,\ \ldots,\ n = k$
  1. Se verifica que $n < N$

     $True,$ se continua el algoritmo
     $False$, se corta el algoritmo

  2. Se verifica que $val1 == val$

     $True$, se ha llegado a la raíz aproximada con m cifras significativas.
     $False$, se continua el algoritmo.

  3. Se verifica que $cen! = 0$ para evaluar $f(x_n)$ y $f'(x_n)$

     $True$, la función es trigonometrica

```
xn=round(f.evalf(subs={x:val}),10)
xpn=round(g.evalf(subs={x:val}),10)
```

4. Se determina la siguiente $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

   val1=val donde: x_{n-1}
   val=round(val-xn/xpn,10) donde: x_n

5. Se encuentra el error relativo $\epsilon = \left|\frac{x_n - x_{n-1}}{x_n}\right|$

   error=round(abs((val1-val)/val),10)

6. Se imprime la tabla, y se realiza $n = n + 1$ repitiendo la secuencia de pasos.

Conclusión: Concluimos que el método de newton es bastante importante ya que es una de las técnicas numéricas más poderosas y conocidas, para determinar las raíces de una ecuación y esto a través de aproximaciones El método casi siempre converge muy rápidamente si x0 está "suficientemente cerca" de la raíz., pero en la programación esta cambia un poco debido a las limitantes

Def Metodo de Secante (N,f,a,b, cen,TOL,cifra)

Función General: es el análisis numérico, para encontrar las raíces de una función, mediante interacciones.

Parámetros:
N= número de interacciones
f= función
a= intervalo inicial
b= intervalo final
cen= función con identidad trigonometrica
TOL= Tolerancia
cifra

1. *Primero empezamos a usar algunas funciones para hacer los cálculos primarios para la realización del ejercicio(iteraciones)*

$$def\ Numero\ Iteraciones(a, b, TOL):$$
$$val = (b - a)/TOL$$
$$N = (math.log10(val)/math.log10(2))$$
$$print\ (N)$$

2. *.Después hacemos la definición de la función ya sea esta trigonométrica o no, esto dado por una serie de ecuaciones programadas para que el programa responda de manera satisfactoria,*

   Que es equivalente a hacerlo de manera manual con las formulas del método

$$
\begin{gathered}
n = 0 \\
m = 5000 \\
xaxis = np.linspace(-10, 10, m) \\
x = sym.symbols('x') \\
val = xaxis[n] \\
xn = round(f.evalf(subs = \{x : val\}), 10) \\
np\_yaxis = np.array([xn]) \\
while\ (n <= m - 2) : \\
val = xaxis[n] \\
xn = round(f.evalf(subs = \{x : val\}), 10) \\
yaxis = np.append(np\_yaxis, xn) \\
np\_yaxis = yaxis \\
n \mathrel{+}= 1 \\
Grafica(xaxis, yaxis)
\end{gathered}
$$

3.Ahora graficamos usando estas funciones que son equivalentes a hacerlo de manera manual

$$
\begin{gathered}
def\ Grafica(x, y) : \\
fig = plt.figure() \\
ax = fig.add\_subplot(1, 1, 1) \\
ax.spines['left'].set\_position('zero') \\
ax.spines['bottom'].set\_position('zero') \\
ax.spines['right'].set\_color('none') \\
ax.spines['top'].set\_color('none')
\end{gathered}
$$

Conclusión: Concluimos que el método de la secante es bastante importante ya que es unaes una variante del método de Newton,La ventaja es que no hay que calcular la derivada de f; esto es de gran ayuda en un caso en que f′ sea difícil de calcular, pero en la programación esta cambia un poco el proceso debido a las limitantes de la computadora así como las utilidades reales

$def \quad Gauss\,Jordan(A, n, arr, par, order)$

Función General: Diagonaliza o neutraliza las particiones, dependiendo del subíndice en el que se encuentren.

Parámetros:

$A:$ Matriz original de la cual se derivan las particiones, también llamada $arreglo\ Principal$

$n:$ El subíndice de la partición a iterar.

$arr:$ Vector, que guarda las dimensiones de las particiones de la matriz, para su reconstrucción en la función $GenerarMatriz()$

$par: \lfloor \frac{m}{2} \rfloor$ donde $m =$ no. De particiones

$order:$ Vector, que guarda la secuencia de los pasos a seguir del algoritmo.

$k:$ iteración

k	$order[k]$
0	0
1	1
2	3
3	2

Dentro del $arreglo\ Principal$, se obtiene el siguiente orden:

A_{11}	A_{12}
0	1
A_{21}	A_{22}
2	3

Para cada subíndice se definen las siguientes tareas:

- Subíndice 0:
  1. Se verifica que pertenezca a la diagonal de la matriz, siendo:

$$n \ mod \ (par + 1) == 0$$

En donde, se tienen los siguientes supuestos:

$n =$ Es el número dentro del subíndice k de la matriz A

$par + 1 = \lfloor \frac{m}{2} \rfloor$ donde $m =$ no. De particiones más 1

  2. Se convierte a la matriz identidad encontrando su matriz inversa, y multiplicando todo el renglón por $arreglo\ Principal[n] = Aaux$

```
Ainv =np.linalg.inv(Aaux)
I = np.around(np.matmul(Aaux,Ainv),6).astype(int)
```

Por el uso excesivo de decimales dentro del lenguaje Python, de debe redondear en la sexta cifra significativa, además de expresar el resultando en enteros.

$$I = Aaux * Ainv$$

$$I = A_{11}A_{11}^{-1}$$

  3. Se verifica que n no sea el último subíndice de $arreglo\ Principal$ de modo que:

$$n! = par + 1$$

De modo, que al verificar la premisa anterior confirmamos implícitamente el subíndice 0. Y entonces, realizamos la multiplicación de $Aaux$ por $Amod = arreglo\ Principal[n + 1]$

```
Amod = Generar Matriz(A,n+1,arr)
Amod = np.around(np.matmul(Aaux,Amod),6).astype(int)
```

En donde:

$$Amod = arregloPrincipal[0 + 1] = A_{12}$$

$$Aaux = arregloPrincipal[0] = A_{11}$$

Finalmente:

$$Amod = A_{12}A_{11} = A'_{12}$$

Actualizando la tabla tenemos:

I	A'_{12}

0	1
A_{21}	A_{22}
2	3

- Subíndice 1
  1. Se verifica que no pertenezca a la diagonal de la matriz, siendo:

$$\neg(n \ mod \ (par + 1) == 0)$$

Se definen los siguientes parámetros para los subíndices de la matriz:

$$arregloPrincipal[n] = Aaux = A_{12}$$

$$arregloPrincipal[n + ppp] = Aaux0$$

$$arrelgloPrincipal[n + pp] = Aaux0$$

$$arregloPrincipal[n + p] = Aaux1$$

I	A'_{12}
0	1
A_{21}	A_{22}
2	3

En donde $p, \ pp$ serán los subíndices adecuados para $n = 1$, definidos en las siguientes fórmulas:

$$p = -1^{n+1} = -1^{1+1} = -1^2 = 1$$

Para pp se definen dos casos:

$pp = -1^i(i)$ en donde $i = 1$
$pp = 2 - i$ en donde $i = 2$

De manera que:

i	pp	A
1	-1	A_{11}
2	0	A_{12}

Para ppp se tiene:

$$ppp = -1^{n+1}(i) \text{ en donde } i \in [1,2]$$

i	ppp	A
1	1	A_{11}
2	2	A_{22}

Realizando las operaciones básicas:

$$A_{21} = A_{21} - I * A_{21} = 0$$
$$A_{22} = A_{22} - A_{12}A_{21} = A'_{22}$$

Exponemos $arreglo\ Principal$ como arr:

$$arr[n + ppp] = arr[n + ppp] - arr[n + pp] * arr[n + p]$$

i	p	pp	ppp	A
1	1	-1	1	$A_{21} = A_{21} - A_{11}A_{21}$
2	1	0	2	$A_{22} = A_{22} - A_{12}A_{21}$

Actualizamos la tabla:

I	A'_{12}
0	1
0	A'_{22}
2	3

- Subíndice 2

  Se verifica que no pertenezca a la diagonal de la matriz, siendo:

$$\neg(n \ mod \ (par + 1) == 0)$$

Se definen los siguientes parámetros para los subíndices de la matriz:

$$arregloPrincipal[n] = Aaux = A_{22}$$

$$arregloPrincipal[n + ppp] = Aaux0$$

$$arrelgloPrincipal[n + pp] = Aaux0$$

$$arregloPrincipal[n + p] = Aaux1$$

I	A'_{12}
0	1
A_{21}	A_{22}
2	3

En donde $p, \ pp$ serán los subíndices adecuados para $n = 2$, definidos en las siguientes fórmulas:

$$p = -1^{n+1} = -1^{2+1} = -1^{3} = -1$$

Para pp se definen dos casos:

$pp = -1^{n}(i)$ en donde $i = 1$
$pp = 2 - i$ en donde $i = 2$

De manera que:

i	pp	A
1	1	A_{11}

2	0	A_{12}

Para ppp se tiene:

$$ppp = -1^{n+1}(i) \text{ en donde } i \in [1,2]$$

i	ppp	A
1	-1	A_{11}
2	-2	A_{22}

Realizando las operaciones básicas:

$$A_{21} = A_{11} - 0 * A_{12} = 0$$
$$A_{22} = A_{22} - I * A_{21} = A'_{22}$$

Exponemos $arregloPrincipal$ como arr:

$$arr[n + ppp] = arr[n + ppp] - arr[n + pp] * arr[n + p]$$

i	p	pp	ppp	A
1	-1	1	-1	$A_{12} = A_{12} - A_{22}A_{12}$
2	-1	0	-2	$A_{11} = A_{11} - A_{21}A_{12}$

```
B = np.around(np.matmul(Aaux1,Aaux0),6).astype(int)
C= Aaux0 – B
```

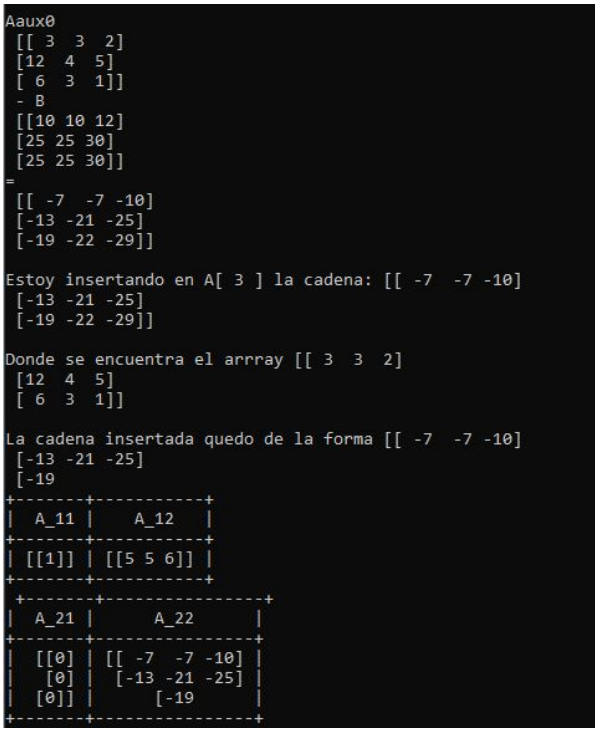
- Subíndice 3
  Son repetidos los mismos pasos a seguir en el subíndice 0, con la única distinción en la que:

$$\neg(n! = par + 1)$$

Por lo que al confirmar, que es el último, retornará.

Problemas que se necesitan resolver:

1. La secuencia correcta del algoritmo debe se $[0, 1, 3, 2]$ haciendo la principal diferencia, la matriz identidad, que al formarse debe restar y no retornar.
2. La inclusión del vector recurso.
3. Pulir la inclusión de string en ciertas particiones, como es el ejemplo de la partición 1,n, en la que inserta de manera equivocada.



Conclusión: Concluimos que el método de Gauss Jordan es bastante importante para los métodos numéricos ya que es un método bastante accesible y medianamente simple de usar en casos prácticos debido a que no sólo elimina los términos debajo de la diagonal principal sino también los que están sobre de ella., pero en la programación esta cambia un poco el proceso debido a las limitantes de la computadora
así como las utilidades reales

Def Método de Doolittle (u,l)

Función General: Genera las matrices y aplica el principio de del método de Doolitle , pidiendo un tamaño para la matriz

Parámetros:

u = matriz
l=matriz
m= número de renglones y columnas
r= renglón
c= columna
k= contador del pivote

1.Primero empezamos definiendo las matrices

$$m = int(input("Introduce\ el\ número\ de\ renglones"))$$
$$matriz = np.zeros([m, m])$$
$$u = np.zeros([m, m])$$
$$l = np.zeros([m, m])$$

2. Usamos una ecuación para que pueda visualizar e introducir los elementos de una manera agradable

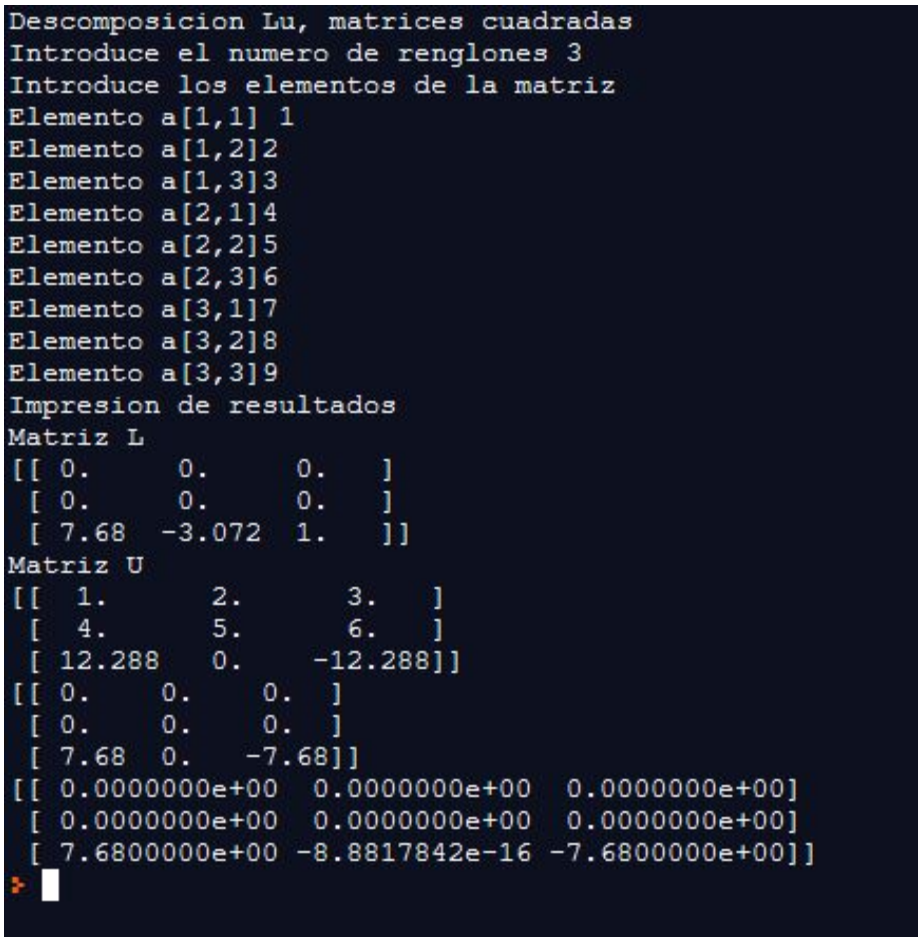
$$matriz[r, c] = (input("Elemento\ a[" + str(r + 1) + "," + str(c + 1) + "]"))$$

3. Ahora hacemos las operaciones debajo de la diagonal

$$matriz[r, c] = matriz[r, c] - (factor * matriz[k, c])$$

4. Es momento de comprobar

$$matrizr[r, c] \mathrel{+}= (l[r, k] * u[k, c])$$



Conclusión: Concluimos que el método de Doolitle es un método bastante accesible y simple de usar en casos prácticos y teóricos debido a que Aún cuando las matrices L y U pueden obtenerse por el método de Gauss,aunque es deseable encontrar un método más directo para su determinación, podemos decir que es un método muy práctico de usar pero en la programación esta cambia un poco el proceso