UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Tarea: Sistemas de ecuaciones lineales

Néstor Heli Aponte Ávila
Métodos Númericos
Cód. 20182167052
2022 - I

14 de junio de 2022

1. Ejercicios

1. Construya un código lo más sencillo posible para hacer la sustitución hacia atrás de una matriz triangular superior.

```
import numpy as np # libreria util para cosas matematicas

def backsus(A,B): # def de la funcion que recibe matriz A, vector independiente B
    A = np.array(A); B = np.array(B) # Convierto esas listas a matrices numericas
    # matriz aumentada
    AB = np.concatenate((A,B),axis=1) #.concatenate() junta vectores, axis (dimension de la la matriz)
    # parametros para recorrer la matriz por filas y columnas
    tam = np.shape(AB) # .shape(AB) devuelve tamano matriz AB fila x columna
    n = tam[0] # tamano fila
    m = tam[1] # tamano columna
    endrow = n-1 # ultima fila, (python accede a indice posicion desde [0,1,...])
    endcol = m-1 # ultima columna
    X = np.zeros(n) # vector de ceros que recoge las soluciones

    for i in range(endrow,-1,-1): # i para recorrer filas
        suma = 0 # variable que recoge la suma que debo restar (en la formula)
        for j in range(i+1,endcol,1): # j para recorrer columnas
            suma = suma + AB[i,j]*X[j] # calcula la suma que debo restar en mi iteracion pues evalua los x_j que ya he encontrado en la coordenada a_ij
        b = AB[i,endcol] # ultimo elemento del vector independiente b en la fila i
        X[i] = (b-suma)/AB[i,i] # formulilla para la variable x_i

    X = np.transpose([X]) # Para ponerlo en vertical que se ve mas bonito

    return X
```

Listing 1: Sustitución hacia atrás

2. Construya un código sencillo del método de Gauss haciendo primero reducción a matriz triangular superior usando pivoteo parcial y luego usando la función de sustitución hacia atrás construida en el primer punto.

```
import numpy as np # libreria util para cosas matematicas
from backsus import * # me traigo la funcion del anterior punto

def gaussprince(A,B): # def de la funcion que recibe matriz A, vector independiente B
    A = np.array(A); B = np.array(B) # Convierto esas listas a matrices numericas
    # matriz aumentada
    AB = np.concatenate((A,B),axis=1) #.concatenate() junta vectores, axis (dimension de la la matriz)
    # parametros para recorrer la matriz por filas y columnas
    tam = np.shape(AB) # .shape(AB) devuelve tamano matriz AB fila x columna
    n = tam[0] # tamano fila
    m = tam[1] # tamano columna

    # eliminacion hacia adelante
    for i in range(0,n-1,1): # recorrido por filas
        pivote   = AB[i,i] # asignacion del pivote a_ii
        adelante = i + 1 # filas que se veran afectadas
        for k in range(adelante,n,1): # Rango para recorrer los que estan debajo del pivote
            factor  = AB[k,i]/pivote # calculo del factor para fila k columna i
            AB[k,:] = AB[k,:] - AB[i,:]*factor # formulilla
    newA = AB[:,:-1] # Todo menos la ultima columna, matriz pivoteada
    newB = []
    for i in range(len(newA)): # len() mide la longitud del vector
```

```python
        newB.append([AB[i,-1]]) # necesito que B este de la forma [[b1],[b2],...] si no la otra funcion me bota error por la dimension --- .append(x) anade a x al final de la lista
    X = backsus(newA,newB) # vector solucion sustitucion hacia atras
    print("X:\n",X)
```

Listing 2: Gauss con pivoteo y sustitución hacia atrás

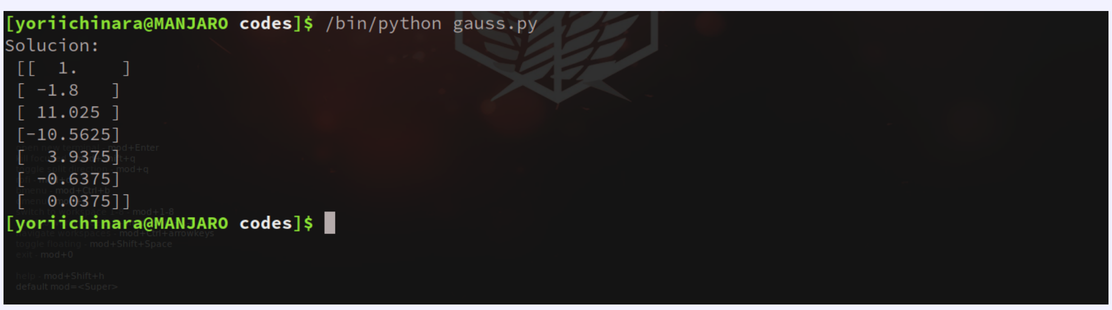
3. Use el código para resolver el siguiente problema: Hallar el polinomio de grado 6

$$y = a_1 + a_2x + a_3x^2 + a_4x^3 + a_5x^4 + a_6x^5 + a_7x^6$$

que pasa por los puntos $(0,1),(1,3),(2,2),(3,1),(4,3),(5,2),(6,1)$.

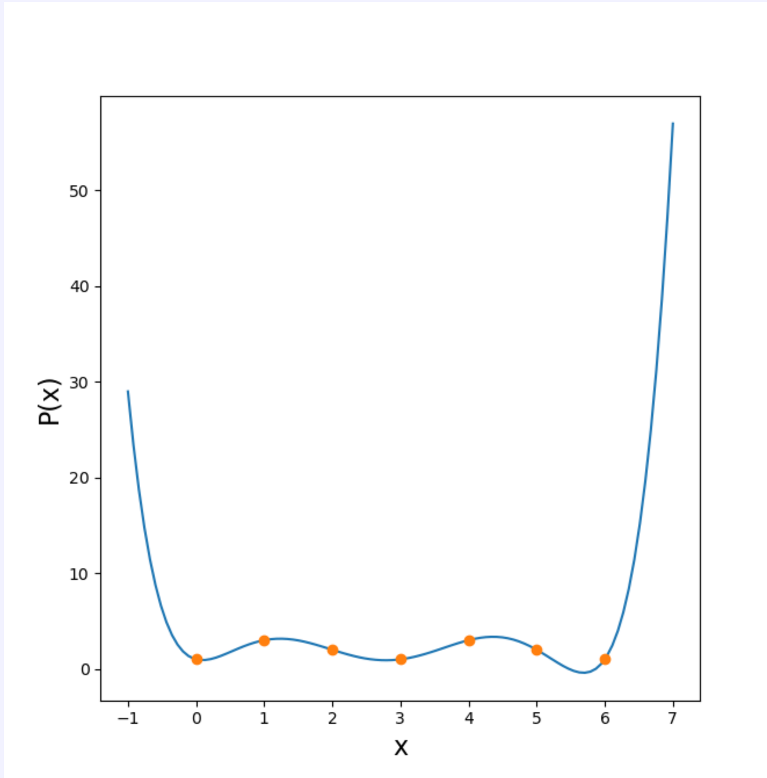
$$\begin{cases} a_1 + a_2(0) + a_3(0)^2 + a_4(0)^3 + a_5(0)^4 + a_6(0)^5 + a_7(0)^6 = 1 \\ a_1 + a_2(1) + a_3(1)^2 + a_4(1)^3 + a_5(1)^4 + a_6(1)^5 + a_7(1)^6 = 3 \\ a_1 + a_2(2) + a_3(2)^2 + a_4(2)^3 + a_5(2)^4 + a_6(2)^5 + a_7(2)^6 = 2 \\ a_1 + a_2(3) + a_3(3)^2 + a_4(3)^3 + a_5(3)^4 + a_6(3)^5 + a_7(3)^6 = 1 \\ a_1 + a_2(4) + a_3(4)^2 + a_4(4)^3 + a_5(4)^4 + a_6(4)^5 + a_7(4)^6 = 3 \\ a_1 + a_2(5) + a_3(5)^2 + a_4(5)^3 + a_5(5)^4 + a_6(5)^5 + a_7(5)^6 = 2 \\ a_1 + a_2(6) + a_3(6)^2 + a_4(6)^3 + a_5(6)^4 + a_6(6)^5 + a_7(6)^6 = 1 \end{cases}$$

```python

gaussprince(
  [[1,0,0,0,0,0,0],
  [1,1,1**2,1**3,1**4,1**5,1**6],
  [1,2,2**2,2**3,2**4,2**5,2**6],
  [1,3,3**2,3**3,3**4,3**5,3**6],
  [1,4,4**2,4**3,4**4,4**5,4**6],
  [1,5,5**2,5**3,5**4,5**5,5**6],
  [1,6,6**2,6**3,6**4,6**5,6**6]],
  [[1],[3],[2],[1],[3],[2],[1]]
)

```



Elabore un plot donde se vea el polinomio y los puntos.

```python

import numpy as np # Libreria util para cosas matematicas
from numpy.polynomial.polynomial import Polynomial # Para hacer polinomios facil
import matplotlib.pyplot as plt # Libreria para plotear

X = [1,-1.8,11.025,-10.5625,3.9375,-0.6375,0.0375] # Vector de coeficientes del polinomio
polinomio = Polynomial(coef=X) # planteamiento polinomio

Px = [0,1,2,3,4,5,6] # Puntos coordenados (x,y)
Py = [1,3,2,1,3,2,1] #

dom = np.linspace(start=-1, stop=7,num=100) # domino donde voy a evaluar el polinomio linspace() construye un vector de num-puntos equidistantes entre start y stop
plt.plot(dom,polinomio(dom)) + plt.plot(Px,Py,"o") # plot polinomio + puntos
plt.xlabel("x",size=16) # indicacion eje x
plt.ylabel("P(x)",size=16) # indicacion eje y
plt.show() # lanza el grafico para poder ver el plot

```

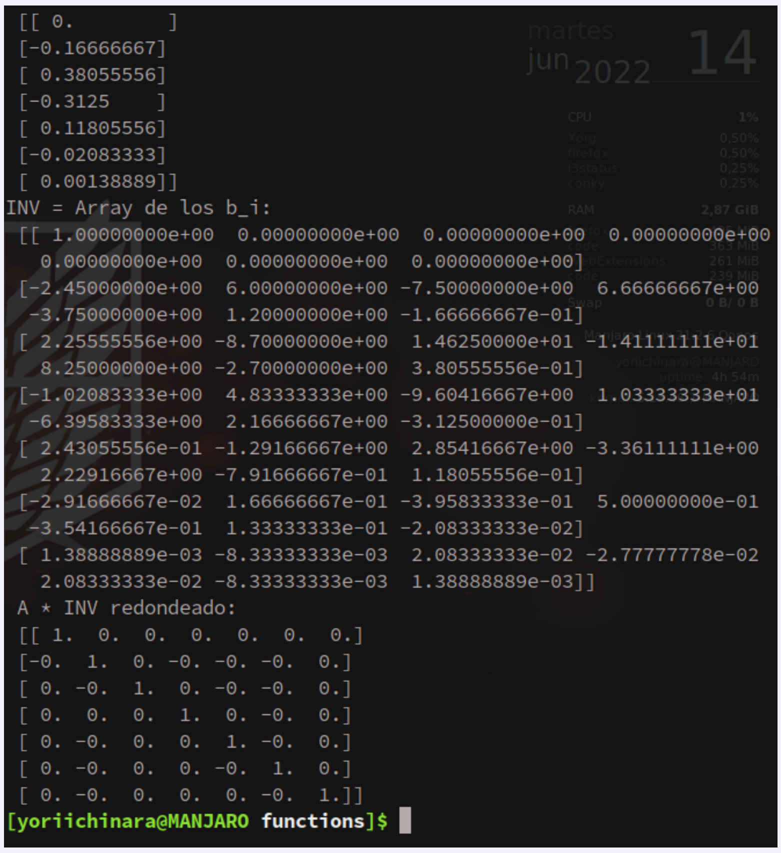


4. Considere la matriz

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 & 32 & 64 \\ 1 & 3 & 9 & 27 & 81 & 243 & 729 \\ 1 & 4 & 16 & 64 & 256 & 1024 & 4096 \\ 1 & 5 & 25 & 125 & 625 & 3125 & 15625 \\ 1 & 6 & 36 & 216 & 1296 & 7776 & 46656 \end{bmatrix}$$

y cálcule la descomposición LU, luego considere $\vec{b_i}$ al vector canonico 7×1 que tiene 1 en la coordenada i-ésima y ceros en en el resto. Para cada $\vec{b_i}$, $i = 1, 2, \cdots, 7$ resuelva el sistema $Ax = b_i$ usando LU. Construya una matrix X cuyas columnas sean los vectores solución obtenidos en el debido orden. Por último realice la multiplicación de $A * X$ y redacte una conclusión al respecto.

```python

# matrix asociada
A = np.array([[ 1, 0, 0, 0, 0, 0, 0], # np.array() para verla y trabajarla como matrix
              [ 1, 1, 1, 1, 1, 1, 1],
              [ 1, 2, 4, 8, 16, 32, 64 ],
              [ 1, 3, 9, 27, 81, 243, 729],
              [ 1, 4, 16, 64, 256, 1024,4096],
              [ 1, 5, 25, 125, 625, 3125, 15625],
              [ 1, 6, 36, 216, 1296, 7776, 46656]], dtype=float)

#Resolviendo el sistema para cada vector
b1 = lumatrix(A, np.array([1,0,0,0,0,0,0],dtype=float))  # lumatrix(A,B) solucion la matrix A para el vector independiente B
b2 = lumatrix(A, np.array([0,1,0,0,0,0,0],dtype=float))
b3 = lumatrix(A, np.array([0,0,1,0,0,0,0],dtype=float))
b4 = lumatrix(A, np.array([0,0,0,1,0,0,0],dtype=float))
b5 = lumatrix(A, np.array([0,0,0,0,1,0,0],dtype=float))
b6 = lumatrix(A, np.array([0,0,0,0,0,1,0],dtype=float))
b7 = lumatrix(A, np.array([0,0,0,0,0,0,1],dtype=float))

INV = np.concatenate((b1,b2,b3,b4,b5,b6,b7),axis=1) # concatena los vector en una sola matrix

Id = np.round(np.dot(A,INV)) # Que me lo redondee con np.round() porque hay cabida a error pequeno --- np.dot() para el producto de matrix

print("INV = Array de los b_i:\n",INV,"\n A * INV redondeado: \n",Id)

```



Reflexión. Al juntar esas soluciones conseguimos la matriz inversa pues si lo pienso con detenimiento, para mi matriz A y su inversa A^{-1} (bajo existencia) tengo que

$$A * A^{-1} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix} = I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

De donde

$$\begin{cases} a_{11}b_{11} + a_{12}b_{21} + \cdots + a_{1n}b_{n1} = 1 \\ a_{21}b_{11} + a_{22}b_{21} + \cdots + a_{2n}b_{n1} = 0 \\ \vdots \\ a_{n1}b_{11} + a_{n2}b_{21} + \cdots + a_{nn}b_{n1} = 0 \end{cases} \equiv \quad A * \begin{bmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

El análisis resulta análogo para cada vector componente de la matriz inversa. Por todo ello, si, puedo realizar el cálculo de la matriz inversa, resolviendo los sistemas que van a los vectores canonicos de la matriz identidad.

Other. Era mas fácil en MATLAB :((

2. Programas en Python

```python
from decimal import Decimal
import numpy as np

# solucion con el metodo LU

# input
def lumatrix(A,B): # B es el vector de valores independientes
    # previa
    B  = np.transpose([B]) # acomoda el B como vector vertical
    AB = np.concatenate((A,B),axis=1) # matriz aumentada

    # Pivoteo parcial por filas
    tamano = np.shape(AB) # tamano matriz aumentada nxm
    n = tamano[0] # num filas
    m = tamano[1] # num columnas

    # eliminacion gaussiana hacia adelante
    L = np.identity(n,dtype=float) # inicia L como matriz identidad nxn
    for i in range(0,n-1,1): # recorrido por filas
        pivote = AB[i,i] # calcula pivote
        adelante = i+1 # solo filas debajo del pivote
        for k in range(adelante,n,1): # cada elemento debajo del pivote
            factor = AB[k,i]/pivote # factor
```

```
            AB[k,:] = AB[k,:] - AB[i,:]*factor # formulilla
            L[k,i] = factor # recoge los factores

    U = np.copy(AB[:,:m-1]) # Matriz U, toda la anterior menos el vector de val
    independientes

    # Resolver LY = B1
    LB =np.concatenate((L,B),axis=1) #matriz aumentada

    # sustitucion hacia adelante
    Y = np.zeros(n,dtype=float) # vector ceros que recoge las soluciones
    Y[0] = LB[0,n] # primer valor independiente que arranca el despeje
    for i in range(1,n,1): # recorrido por filas
        suma = 0 # inicializacion suma
        for j in range(0,i,1): # rango columnas (variables a reemplazar para luego despejar)
            suma = suma + LB[i,j]*Y[j] # calculo suma para el despeje
        b = LB[i,n]  # valor independiente del vector B
        Y[i] = (b-suma)/LB[i,i] # formulilla

    Y = np.transpose([Y]) # dispone Y como un vector vertical

    # Resolver UX = Y
    UY =np.concatenate((U,Y),axis=1) # matrix aumentada

    # sustitucion hacia atras
    ultfila = n-1 # ultima fila
    ultcolumna = m-1 # ultima columna
    X = np.zeros(n,dtype=float) # vector ceros que recoge soluciones

    for i in range(ultfila,0-1,-1): # recorrido filas
        suma = 0 # inicia variable
        for j in range(i+1,ultcolumna,1): # recorrido columnas
            suma = suma + UY[i,j]*X[j] # calcula la suma evaluando x_j encontrados
        b = UY[i,ultcolumna] # valor independiente vector B
        X[i] = (b-suma)/UY[i,i] # formulilla

    X = np.transpose([X]) # dispone X como vector

    print("solucion: \n", X)

    return X
```

Listing 3: Método LU

Referencias

[1] Jhon H. Mathews - Kurtis D. Fink, *Métodos Númericos con MATLAB*, Pretince Hall, (2000)

[2] ESPOL, *Métodos Númericos - Curso con Python*, Guayaquil - Ecuador, (link)