Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería



**MÉTODOS NUMÉRICOS**

**Inversa**

**Docente:** Itzel Barriba Cazares

**Alumno:** Gómez Cárdenas Emmanuel Alberto

Matrícula: 1261509

Primero copie y pegue las funciones obtener entero, obtener flotante de un programa anterior, posteriormente cree las funciones imprimir matriz e imprimir respuestas.  
Cree unas matrices con valores predeterminados, ya que para hacer pruebas no es buena idea estar creando matrices cada vez que quieras correr el programa.  
Procedí a crear la función solveMat que resuelve la matriz por el método de Gauss-Jordan con matriz aumentada, después cree los métodos generar matriz aumentada, agregar matriz aumentada y quitar matriz aumentada, todo esto debido a que para el método de Gauss Jordan necesitamos manipular una matriz aumentada.  
Mas tarde, cree los métodos multiplicar matriz y multiplicar AxB, me vi forzado a crear dos funciones diferentes, ya que una matriz y una lista no funcionan igual en cuanto a Python.

La función multMat sirve para multiplicar dos matrices. En mi caso, la utilice para comprobar que la ecuación se satisfaga multiplicando la matriz inversa por la matriz A, lo cual da la matriz identidad

Últimamente cree la función crear matriz la cual tiene como parámetro el tamaño de la matriz y va pidiendo cada valor de esta.

Codigo

#Capturar entero

def getInt(str):

    return int(input(str + ": "))

#Capturar flotante

def getFloat(str):

    return float(input(str + ": "))

#Imprime los valores de X

#si el valor esta dentro de cierto rango, este se redondea

def printAns(name, mat):

    print("\n\t" + name + ":")

    for i in range(len(mat)):

        if((mat[i][0] < 1e-7) & (mat[i][0] > -1e-7)):

            print("\tx%d = %15.10f" %(i, round(mat[i][0])),  end = '')

        else: print("\tx%d = %15.10f" %(i, mat[i][0]), end = '')

        print("")

#Imprime Matriz

def printMat(name, mat):

    print("\n\t" + name + ":")

    for i in range(len(mat)):

        for j in range(len(mat)):

            if((mat[i][j] < 1e-7) & (mat[i][j] > -1e-7)):

                print("%15.10f" %(round(mat[i][j])),  end = '')

            else: print("%15.10f" %(mat[i][j]), end = '')

        print("")

def createMat(size):

    #Crea e inicializa la matriz con 0's flotantes

    mat = [[float(0) for j in range(size)] for i in range(size)]

    #inicializa la matriz de respuestas con 0's flotantes

    b = [float(0) for i in range (size)]

    for i in range(size):

        for j in range(size):

            mat[i][j] = getFloat("Valor de [" + str(i) + "][" + str(j) + "]")

    for i in range(size):

        b[i] = getFloat("Valor de x" + str(i))

    return mat,b

#Genera matriz aumentada inizializada en 0's flotantes

def genAumentedMat(size):

    aumgmentedMat = [[float(0) for j in range(2\*size)] for i in range(size)]

    return aumgmentedMat

#Agrega la matriz identidad del lado derecho

def addIdentityMat(mat):

    size = len(mat)

    aumgmentedMat = genAumentedMat(size)

    for i in range(size):

        for j in range(2\*size):

            if(j<size): aumgmentedMat[i][j] = mat[i][j]

            if(j==size+i): aumgmentedMat[i][j] = float(1)

    return aumgmentedMat

#Elimina la matriz identidad que queda despues de emplear la funcion solveMat

def subIdentityMat(aumgmentedMat):

    size = len(aumgmentedMat)

    inverseMat = [[float(0) for j in range(size)] for i in range(size)]

    for i in range(size):

        for j in range(2\*size):

            if(j>=size): inverseMat[i][j-size] = aumgmentedMat[i][j]

    return inverseMat

#Metodo Gauss-Jordan con matriz aumentada

def solveMat(mat):

    size = len(mat)

    #Crea la matriz aumentada

    aumgmentedMat = addIdentityMat(mat)

    for i in range(size):

        for j in range(size):

            if(j!=i):

                #transforma matriz a matriz identidad

                temp = aumgmentedMat[j][i]/aumgmentedMat[i][i]

                for k in range(size\*2):

                    aumgmentedMat[j][k] -= aumgmentedMat[i][k] \* temp

    for i in range(size):

        temp = aumgmentedMat[i][i]

        for j in range(size\*2):

            aumgmentedMat[i][j] /= temp

    return aumgmentedMat

def getInverseMatrix(mat):

    #Comprueba que la determinante sea diferente de 0

    #De lo contrario la matriz es no invertible

    if (getDetMat(mat) == 0):

        print("\n\nMatriz no invertible")

        exit(1)

    return subIdentityMat(solveMat(mat))

#Obtiene la subMatriz necesaria para sacar determinantes

def getSubMat(Mat, col):

    size = len(Mat)

    subMat = [[float(0) for j in range(size-1)] for i in range(size-1)]

    count = 0

    for j in range(size):

        if (j == col):

            pass

        else:

            for i in range(size):

                if(i == 0): pass

                else: subMat[i-1][count] = Mat[i][j]

            count+=1

    return subMat

#Obtiene la determinante de la matriz

def getDetMat(mat):

    det = 0

    rows = len(mat)

    cols = len(mat[0])

    sign = 1

    if (rows == 1 & cols == 1):

        return mat[0][0]

    for col in range(cols):

      subMat = getSubMat(mat, col)

      det += (sign \* mat[0][col] \* getDetMat(subMat))

      sign \*= -1

    return det

#Multiplicacion de dos matrices

def multMat(matA, matB):

    colsA = len(matA[0])

    rowsB = len(matB)

    colsAxB = len(matB[0])

    rowsAxB = len(matA)

    if (colsA == rowsB):

        matAxB = [[float(0) for j in range(rowsAxB)] for i in range(colsAxB)]

        for i in range(rowsAxB):

            for j in range (colsAxB):

                temp = float(0)

                for k in range(rowsB):

                    temp+= matA[i][k] \* matB[k][j]

                matAxB[i][j] = temp

    return matAxB

#Multiplicacion de una matriz por una lista

def multAxB(matA, B):

    sizeA = len(matA)

    sizeB = len(B)

    colsAxB = 1

    rowsAxB = sizeA

    if (sizeA == sizeB):

        matAxB = [[float(0) for j in range(colsAxB)] for i in range(rowsAxB)]

        for i in range(rowsAxB):

            for j in range (colsAxB):

                temp = float(0)

                for k in range(sizeB):

                    temp+= matA[i][k] \* B[k]

                matAxB[i][j] = temp

    return matAxB

size = getInt("introduzca el tamaño de la matriz")

temp = createMat(size)

matA = temp[0]

B = temp[1]

inversa = getInverseMatrix(matA)

MatrizAxI = multMat(matA, inversa)

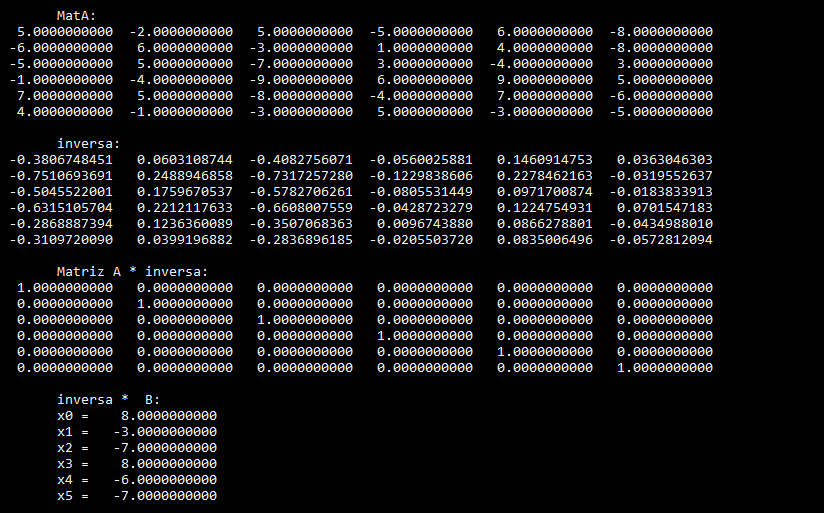
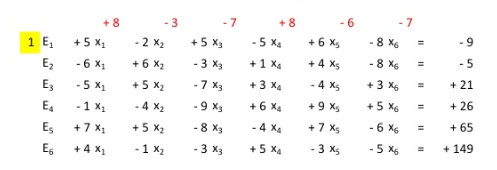
MatrizAxB = multAxB(inversa, B)

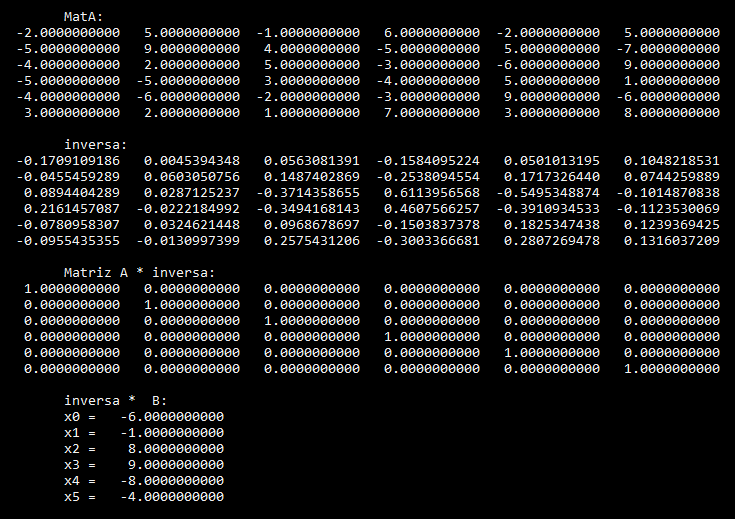
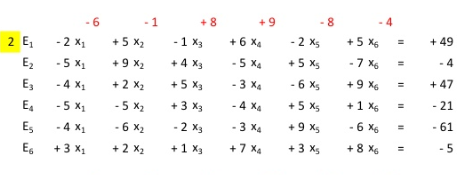
printMat("MatA", matA)

printMat("inversa", getInverseMatrix(matA))

printMat("Matriz A \* inversa", MatrizAxI)

printAns("inversa \*  B", MatrizAxB)





Referencias

<https://www.slideshare.net/licmata/sistmas-de-6x6>  
<https://stackoverflow.com/questions/6473679/transpose-list-of-lists><https://stackoverflow.com/questions/216972/what-does-it-mean-if-a-python-object-is-subscriptable-or-not>  
<https://stackoverflow.com/questions/24026950/rounding-off-small-numbers-in-python><https://stackoverflow.com/questions/23643144/typeerror-object-of-type-function-has-no-len-in-python>