**Numerical methods**

**Assignment 3**

**\*\*All the outputs/results along with code have been added as attachment in a html file to refrain from lot of editing and probably wrong assumptions/mistakes**

1. Assume any 4 X 4 matrix such that the diagonal is dominant. Let the coefficients in each equation be widely differing (e.g. 100 x1+ 40 x2+10 x3 + x4). Assume any vector say {1,1,1,1} and generate the right-hand vector. Now write a Gauss elimination algorithm without partial pivoting and solve the above problem. Now take the same matrix and rearrange the rows such that the diagonal dominance is destroyed. Re-solve the problem and obtain the solution, and compare the obtained solution with that of the actual answer.

Code:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import numpy as np

import sys

# ### First half of question that is to generate B

# In[2]:

A = np.array([[1.0,2.0,1.0],

             [2.0,3.0,3.0],

             [3.0,-1.0,2.0]])

A

# In[3]:

x=np.array([ 2., -1.,  3.])

x

# In[4]:

np.dot(A, x) #expected B

# In[ ]:

# ### Remaining part to derive x from Ax=B

# In[5]:

# Reading number of unknowns

n = 3 #int(input('number of unknowns: '))

# In[6]:

ab = np.zeros((n,n+1))

#taking input for A|B

print('Enter "A|B" matrix:')

for \_ in range(n):

    for \_\_ in range(n+1):

        ab[\_][\_\_] = float(input(f'ab{\_+1}{\_\_+1}='))

# In[7]:

ab = np.array([[1.0,2.0,1.0,3.0],

             [2.0,3.0,3.0,10.0],

             [3.0,-1.0,2.0,13.0]])

# In[8]:

ab

# In[ ]:

# In[9]:

x = np.zeros(n) #solution matrix

# In[10]:

#gauss elimination

def ge():

    for i in range(n):

        if ab[i][i] == 0.0:

            sys.exit('Divide by zero detected!')

        for j in range(i+1, n):

            ratio = ab[j][i]/ab[i][i]

            for k in range(n+1):

                ab[j][k] = ab[j][k] - ratio \* ab[i][k]

    return ab

# In[11]:

#back substitution

def bs():

    x[n-1] = ab[n-1][n]/ab[n-1][n-1]

    for i in range(n-2,-1,-1):

        x[i] = ab[i][n]

        for j in range(i+1,n):

            x[i] = x[i] - ab[i][j]\*x[j]

        x[i] = x[i]/ab[i][i]

    return x

# In[12]:

ge()

# In[13]:

bs()

# In[14]:

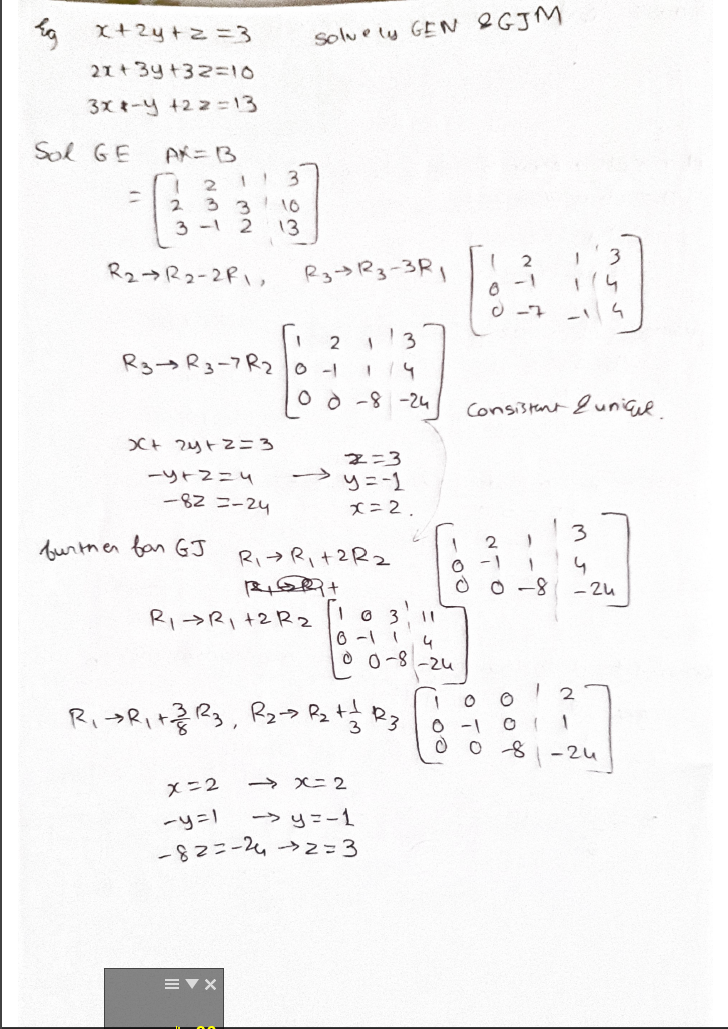
# Printing solution martix "x"

print('\nValues of unknown are as follows: ')

for i in range(n):

    print(f'x{i} = {x[i]}')

Question used for checking algorithm:



1. Write a problem to solve Problem 1 using Crout’s (LU decomposition) method.

Code:

#!/usr/bin/env python

# coding: utf-8

# # LU decomposition

# In[1]:

from numpy import dot

from numpy import zeros,array,product,diagonal

import pprint

import scipy

import scipy.linalg   # SciPy Linear Algebra Library

import numpy

import numpy as np

# In[2]:

#array from question 1

a = np.array([[1.0,2.0,1.0],

             [2.0,3.0,3.0],

             [3.0,-1.0,2.0]])

# In[3]:

b=np.array([ 3., 10.0,  13.0])

# In[4]:

def LUdecomp(a):

    n = len(a)

    for k in range(0,n-1):

        for i in range(k+1,n):

            if a[i,k] != 0.0:

                lam = a [i,k]/a[k,k]

                a[i,k+1:n] = a[i,k+1:n] - lam\*a[k,k+1:n]

                a[i,k] = lam

    return a

# In[5]:

a

# In[6]:

a[0][1]

# In[7]:

a = LUdecomp(a)

# In[8]:

a

# In[9]:

rows=len(a)

cols=len(a[0])

# In[10]:

def mzm(rows: int, cols: int): #make zero matrix

    matrix = np.zeros((rows,cols))

    for i in range(rows):

        for j in range(cols):

            if i==j:

                matrix[i][j]=1

            else:

                matrix[i][j]=0

    m = numpy.array(matrix)

    return m

# In[11]:

L=mzm(3,3)

# In[12]:

U=mzm(3,3)

# In[13]:

U

# In[14]:

def makel(a,L,U):

    dc=0 #diagonal counter

    for i in range(rows):

        dc=dc+1

        for j in range(cols):

            if j<dc-1:

                L[i][j]=a[i][j]

            else:

                U[i][j]=a[i][j]

    return a,L,U

# In[15]:

makel(a,L,U)

# In[16]:

v = np.linalg.solve(L, b)

v

# In[17]:

x = np.linalg.solve(U, v)

x

1. Solve the following problem by the tri-diagonal matrix method (Thomas algorithm).

Note that the value of each unknown is 1.

Code:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import numpy as np

# # Tri Diagonal Matrix Algorithm(Thomas algorithm)

# In[2]:

#driver code / question

A = np.array([[-2,1,0,0,0,0,0,0,0],

              [1,-2,1,0,0,0,0,0,0],

              [0,1,-2,1,0,0,0,0,0],

              [0,0,1,-2,1,0,0,0,0],

              [0,0,0,1,-2,1,0,0,0],

              [0,0,0,0,1,-2,1,0,0],

              [0,0,0,0,0,1,-2,1,0],

              [0,0,0,0,0,0,1,-2,1],

              [0,0,0,0,0,0,0,1,-2]],dtype=float)

# In[3]:

alen=9#alen = len(A)

c=[]

d=[]

e=[]

for d1 in range(alen-1):

    c.append(1)

for d2 in range(alen):

    d.append(-2)

for d3 in range(alen-1):

    e.append(1)

# In[4]:

B = np.array([-1,0,0,0,0,0,0,0,-1])

# In[5]:

def thomasalgo(a, b, c, d):

    nf = len(d) # number of equations

    ac, bc, cc, dc = map(np.array, (a, b, c, d)) # copy arrays

    for it in range(1, nf):

        mc = ac[it-1]/bc[it-1]

        bc[it] = bc[it] - mc\*cc[it-1]

        dc[it] = dc[it] - mc\*dc[it-1]

    xc = bc

    xc[-1] = dc[-1]/bc[-1]

    for il in range(nf-2, -1, -1):

        xc[il] = (dc[il]-cc[il]\*xc[il+1])/bc[il]

    return xc

# In[10]:

thomasalgo(c, d, e, B)

# ### comparing answers only for reference

# In[11]:

np.linalg.solve(A, B)

# In[ ]:

1. Solve the following problem using Gauss-Jacobi and Gauss-Seidel methods.

The exact solution is {1,1,1,1,1}. Solve the problem in single and double precision and comment on the results

Code:

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import pandas as pd

# In[2]:

a = [[1/1, 1/2, 1/3, 1/4, 1/5],

     [1/2, 1/3, 1/4, 1/5, 1/6],

     [1/3, 1/4, 1/5, 1/6, 1/7],

     [1/4, 1/5, 1/6, 1/7, 1/8],

     [1/5, 1/6, 1/7, 1/8, 1/9]]

b = [[1/1+1/2+1/3+1/4+1/5],

     [1/2+1/3+1/4+1/5+1/6],

     [1/3+1/4+1/5+1/6+1/7],

     [1/4+1/5+1/6+1/7+1/8],

     [1/5+1/6+1/7+1/8+1/9]]

# In[3]:

a

# In[4]:

b

# In[5]:

a[0][1]

# In[6]:

def f1(x2,x3,x4,x5):

    x1 = ((1/1+1/2+1/3+1/4+1/5)-(((1/2)\*x2)+((1/3)\*x3)+((1/4)\*x4)+((1/5)\*x5)))/(1)

    return x1

# In[7]:

def f2(x1,x3,x4,x5):

    x2 = ((1/2+1/3+1/4+1/5+1/6)-(((1/2)\*x1)+((1/4)\*x3)+((1/5)\*x4)+((1/6)\*x5)))/(1/3)

    return x2

# In[8]:

def f3(x1,x2,x4,x5):

    x3 = ((1/3+1/4+1/5+1/6+1/7)-(((1/3)\*x1)+((1/4)\*x2)+((1/6)\*x4)+((1/7)\*x5)))/(1/5)

    return x3

# In[9]:

def f4(x1,x2,x3,x5):

    x4 = ((1/4+1/5+1/6+1/7+1/8)-(((1/4)\*x1)+((1/5)\*x2)+((1/6)\*x3)+((1/8)\*x5)))/(1/7)

    return x4

# In[10]:

def f5(x1,x2,x3,x4):

    x5 = ((1/5+1/6+1/7+1/8+1/9)-(((1/5)\*x1)+((1/6)\*x2)+((1/7)\*x3)+((1/8)\*x4)))/(1/9)

    return x5

# In[32]:

# In[11]:

#setting solytion to 0

x1=0

x2=0

x3=0

x4=0

x5=0

#list for storing values

x1l=[]

x2l=[]

x3l=[]

x4l=[]

x5l=[]

# In[12]:

#gauss seidal

for i in range(50000):

    count=0

#     print("i=",i)

    x1l.append(x1)

    x2l.append(x2)

    x3l.append(x3)

    x4l.append(x4)

    x5l.append(x5)

#     print("added to list")

    x1=f1(x2,x3,x4,x5)

#     print(x1)

    x2=f2(x1,x3,x4,x5)

#     print(x2)

    x3=f3(x1,x2,x4,x5)

#     print(x3)

    x4=f4(x1,x2,x3,x5)

#     print(x4)

    x5=f5(x1,x2,x3,x4)

#     print(x5)

#     print("executed expressions")

# In[13]:

df = pd.DataFrame(list(zip(x1l, x2l, x3l, x4l, x5l)),

               columns =['x1', 'x2', 'x3', 'x4','x5'])

# In[14]:

df.tail(5)

# In[16]:

print(df.iloc[5000]) #for checking any particular iteration

# In[ ]:

# In[104]:

x1=0

x2=0

x3=0

x4=0

x5=0

x1l=[]

x2l=[]

x3l=[]

x4l=[]

x5l=[]

# In[17]:

# f1(x2,x3,x4,x5)

# In[18]:

# f2(x1,x3,x4,x5)

# In[19]:

# f3(x1,x2,x4,x5)

# In[20]:

# f4(x1,x2,x3,x5)

# In[21]:

# f5(x1,x2,x3,x4)

# In[22]:

#gauss jacobi

import math

for i in range(5000):

    count=0

#     print("\n\n")

#     print("i=",i)

#     print("x1t",x1t)

#     print("x2t",x2t)

#     print("x3t",x3t)

#     print("x4t",x4t)

#     print("x5t",x5t)

    x1t=f1(x2,x3,x4,x5)

    x2t=f2(x1,x3,x4,x5)

    x3t=f3(x1,x2,x4,x5)

    x4t=f4(x1,x2,x3,x5)

    x5t=f5(x1,x2,x3,x4)

    if abs(x1t-x1)<10\*\*-5:

        print("reached solution")

        break

#     print("executed expressions")

#     print("x1",x1)

#     print("x2",x2)

#     print("x3",x3)

#     print("x4",x4)

#     print("x5",x5)

#     print("current values")

    x1=x1t

    x2=x2t

    x3=x3t

    x4=x4t

    x5=x5t

#     print("x1",x1)

#     print("x2",x2)

#     print("x3",x3)

#     print("x4",x4)

#     print("x5",x5)

#     print("temp val assigned")

    x1l.append(x1)

    x2l.append(x2)

    x3l.append(x3)

    x4l.append(x4)

    x5l.append(x5)

#     print("added to list")

    if abs(x1)==math.inf:

        print("non converging")

        break

# In[23]:

df2 = pd.DataFrame(list(zip(x1l, x2l, x3l, x4l, x5l)),

               columns =['x1', 'x2', 'x3', 'x4','x5'])

# In[24]:

df2

\*\*Please refer html files for result