PA 3: Comparing interpreted and compiled codes

Name: Humesh Reddy Venkatapuram

Aggie ID: 800810540

Date: 03/05/2023

Problem Description: Build three programs that perform Gaussian elimination with back substitution. Use system calls to measure time to capture the time it takes to run to program with different data sizes. Use an appropriate measuring tool that gives you responsible time granularity. Python + NumPy can use pivoting as LU decomposition which does this by default. The other 3 implementations should not have to pivot.

Input: Size of the matrix (250, 500, 1000, 1500, 2000), used random number generator to populate the matrix Output: The total execution time of the program in seconds.

Python code with Numpy:

#Name: Humesh Reddy Venkatapuram

#Date: 03/05/2023

#Description: This program performs Gaussian Elimination for solving matrix equation of the form AX = B

import numpy as np

from numpy import zeros

import time

import scipy

def Elements  (A):

    P, L, U = scipy.linalg.lu(A)

def main():

    N = int(input('Enter size of the Matrix (250, 500, 1000, 1500, 2000): '))

    start\_time = time.time()

# Populating the matrix

    A= np.random.randint(-100, 100, size=(N, N+1))

# Function call to start Gaussian Elimination

    Elements (A)

# Timer just after gaussian elimination is done

    end\_time = time.time()

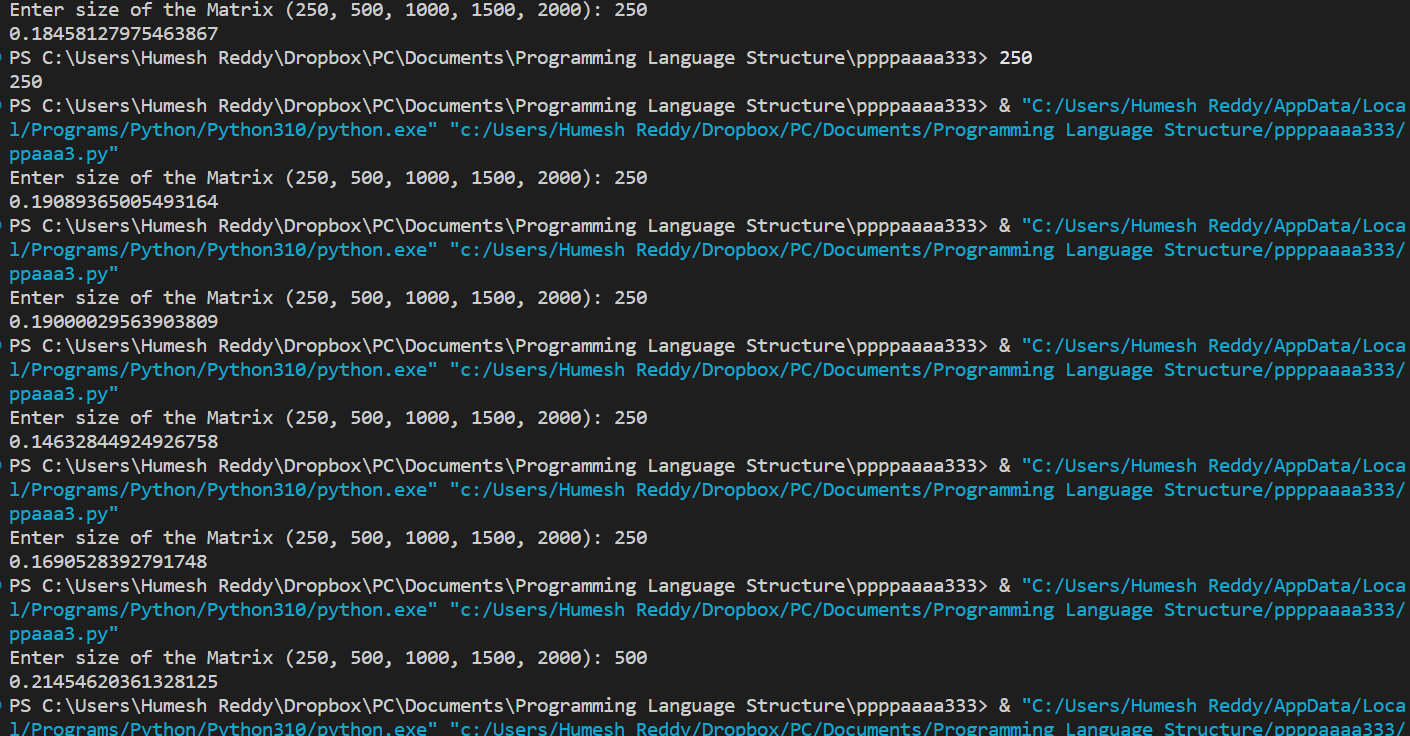
    total\_time = end\_time-start\_time

    print (total\_time)

if \_\_name\_\_ =="\_\_main\_\_":

    main()

Output:



Python code without Numpy:

#Name: Humesh Reddy Venkatapuram

#Date: 03/05/2023

#Description: This program performs Gaussian Elimination for solving matrix equation of the form AX = B

import random

import time

def gaussianElimination (A, x, N):

# Applying Gaussian Elimination

        for i in range(N):

            if A[i][i] == 0.0:

                continue

            for j in range(i+1, N):

                alpha= A[j][i]/A[i][i]

                for k in range(N+1):

                    A[j][k]=A[j][k]-alpha\*A[i][k]

#Back Substution

            x[N-1] = A [N-1] [N]/A [N-1] [N-1]

            for i in range(N-2, -1, -1):

                x[i] = A[i] [N]

                for j in range(i+1,N):

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

                x[i] = x[i]/A[i] [i]

def main():

#Taking in the inpht size of matrix

    N= int(input('Enter size of the Matrix (250, 500, 1000, 1500, 2000): '))

#Timer starts after taking in the input size of matrix

    start\_time = time.time()

# Populating the matrix A without numpy library # The matrix b has been taken cared in matrix A only

    A=[]

    for i in range(N):

        col=[]

        for j in range(N+1):

            col.append(random.randint(-100, 100))

        A.append(col)

    x = []

    for i in range(N):

        col = []

        for j in range(1):

            col.append(0.0)

        x.append(col)

# Function call to start gaussian elimination

    gaussianElimination (A, x, N)

# Timer just after gaussian elimination is done

    end\_time = time.time()

    Total\_time = end\_time-start\_time

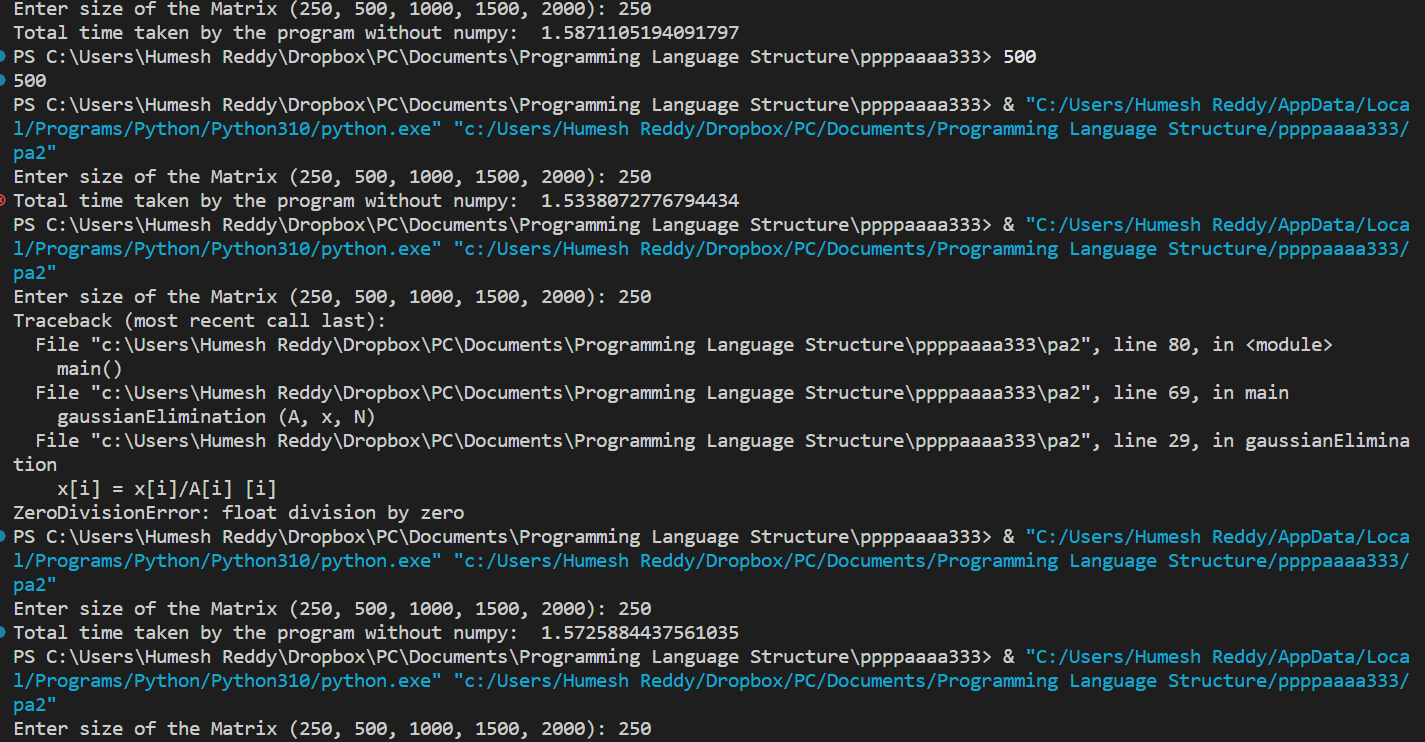
# Print result

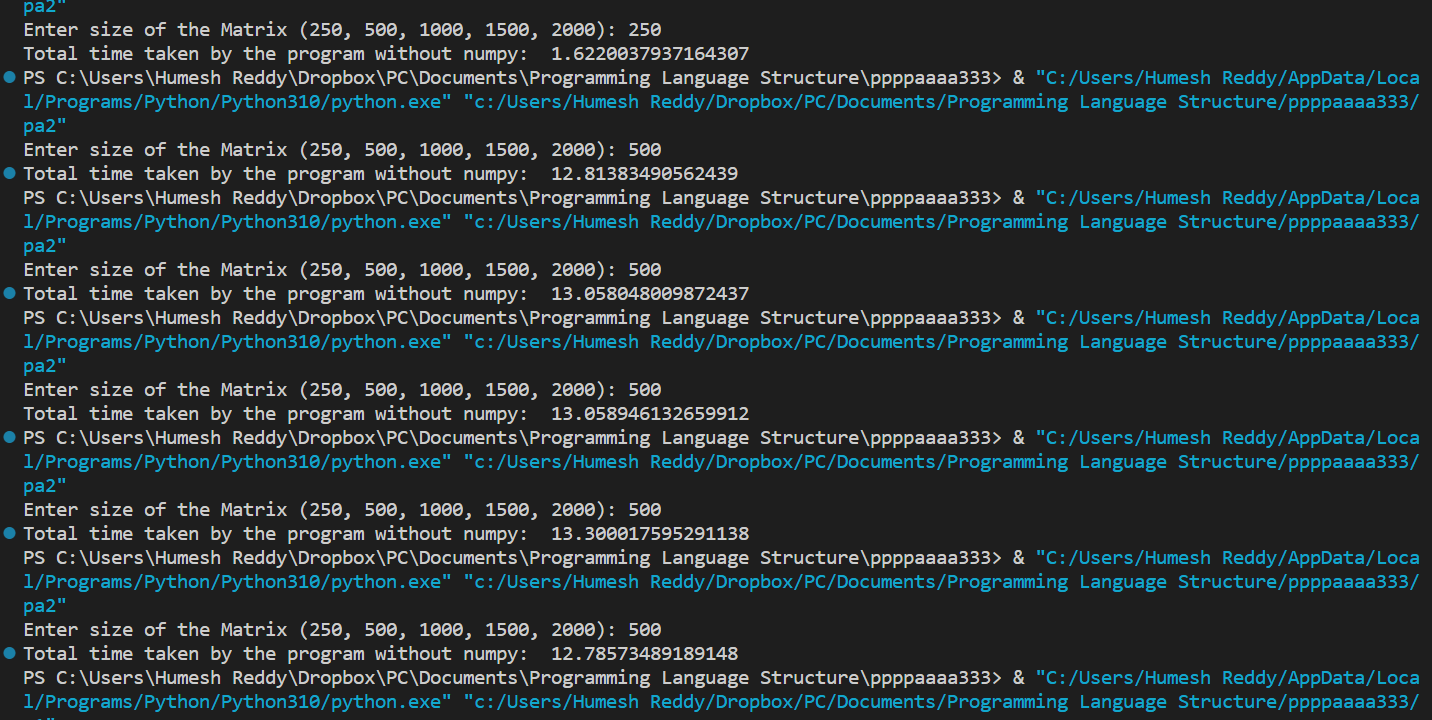
    print ("Total time taken by the program without numpy: ", Total\_time)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Output:





Fortran Code:

Program Gaussian\_elimination

implicit none

Integer::n=2000

Integer i,j,k

Real (8), Allocatable :: a(:,:),b(:),x(:),c(:)

Real(8) d

Real start, end

print \*, "Enter size of the Matrix (250, 500, 1000, 1500, 2000):"

read\*, n

Allocate(a(1:n, 1:n+1), b(1:n),x(1:n),c(1:n))

! Populating the matrices

call RANDOM\_NUMBER(a)

call RANDOM\_NUMBER(b)

! Timer start after taking in the input size of matrix

call cpu\_time(start)

do k=1,n-1,1

do i=k+1,n,1

if(a(k,k) /= 0) then

a(n,i)=a(n,i)-a(i,k)/a(k,k)\*a(n,k)

else

goto 100

endif

d=a(i,k)

do j=1,n,1

a(i,j)=a(i,j)-a(k, j)\*(d/a(k,k))

enddo

enddo

enddo

do i=n,1,-1

do j=1,n,1

if(j/=i) then

c(i)=c(i)+a(i,j)\*x(j)

else

cycle

endif

enddo

x(i)=(a(n,i)-c(i))/a(i,i)

enddo

! Timer end just after performing gaussian elimination.

call cpu\_time(end)

print \*, end-start

100 stop

end

Python code with numpy:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **Time 1** | **Time 2** | **Time 3** | **Time 4** | **Time 5** | **Average** | **Stdev** |
| 250 | 0.1845 | 0.1908 | 0.1900 | 0.1463 | 0.1690 | 0.17612 | 0.016842 |
| 500 | 0.2145 | 0.1700 | 0.1937 | 0.2167 | 0.2233 | 0.20364 | 0.019521 |
| 1000 | 0.2062 | 0.2369 | 0.2569 | 0.2779 | 0.2027 | 0.23612 | 0.028948 |
| 1500 | 0.2721 | 0.3910 | 0.4734 | 0.3288 | 0.2473 | 0.3425200 | 0.082090 |
| 2000 | 0.3094 | 0.5011 | 0.5468 | 0.5115 | 0.3249 | 0.438739 | 0.100546 |

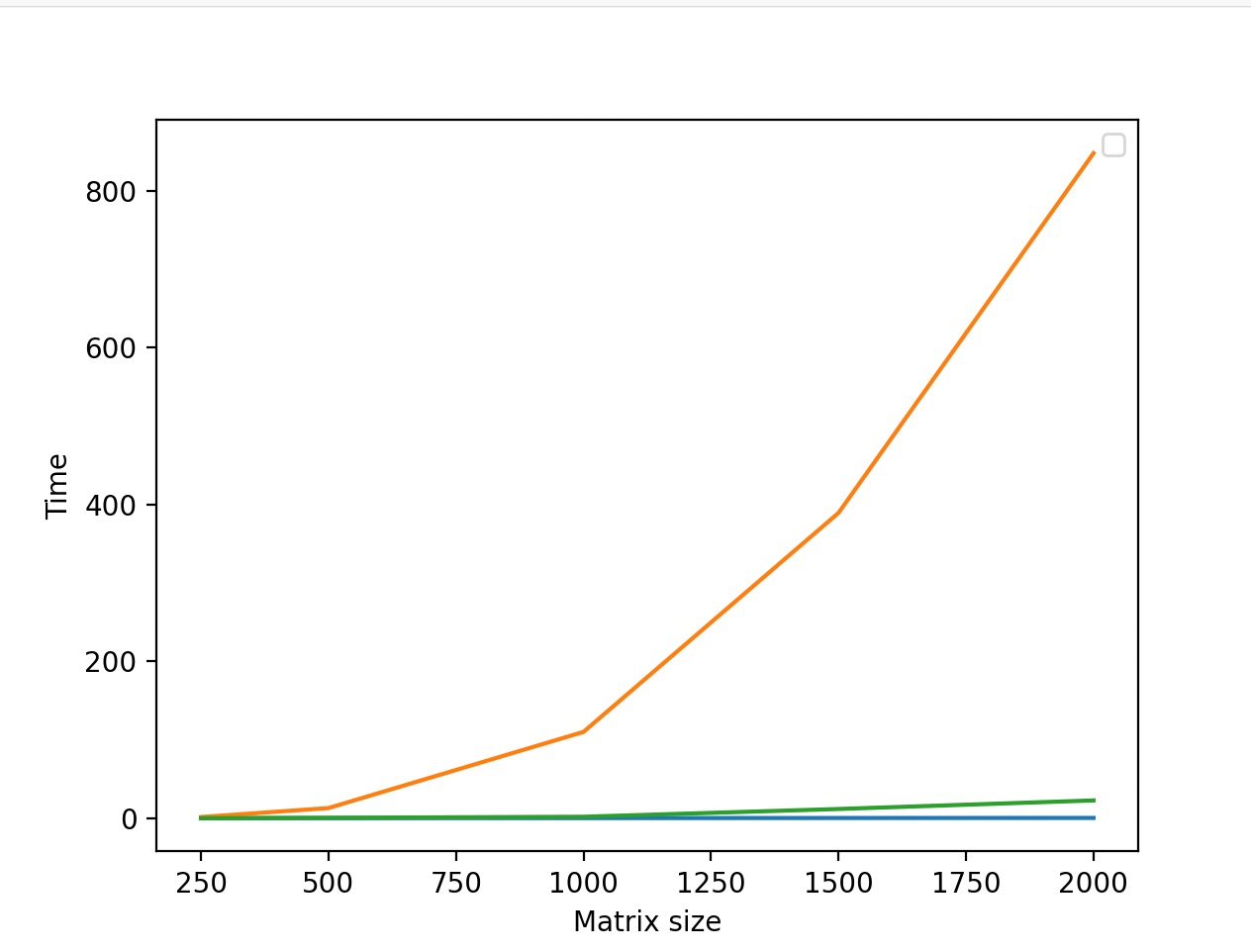
Python code without numpy:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **Time 1** | **Time 2** | **Time 3** | **Time 4** | **Time 5** | **Average** | **Stdev** |
| 250 | 1.5871 | 1.5338 | 1.5725 | 1.6147 | 1.6220 | 1.58602 | 0.0317 |
| 500 | 12.8138 | 13.0580 | 13.0589 | 13.3000 | 12.7857 | 13.0033 | 0.1883 |
| 1000 | 104.3571 | 105.8630 | 108.8426 | 126.2478 | 105.1795 | 110.098 | 8.2155 |
| 1500 | 380.2209 | 351.6018 | 492.6352 | 350.9742 | 370.8865 | 389.2653 | 52.8952 |
| 2000 | 873.8692 | 784.0470 | 819.1984 | 900.9371 | 860.9944 | 847.8092 | 41.3757 |

Fortran code:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | **Time 1** | **Time 2** | **Time 3** | **Time 4** | **Time 5** | **Average** | **Stdev** |
| 250 | 0.0718 | 0.0889 | 0.0554 | 0.0821 | 0.0923 | 0.0781 | 0.0133 |
| 500 | 0.4176 | 0.6894 | 0.5123 | 0.6784 | 0.6111 | 0.5817 | 0.1034 |
| 1000 | 1.8909 | 1.9456 | 1.8794 | 1.9223 | 2.0033 | 1.9282 | 0.0441 |
| 1500 | 10.6663 | 11.3423 | 11.6587 | 12.9098 | 12.5657 | 11.8285 | 0.8159 |
| 2000 | 22.6985 | 23.4523 | 22.0003 | 23.8796 | 21.6698 | 22.7400 | 0.8366 |

Used MatPlot python library to generate the graph taking above data points.



Explanation of graph:

For better comparison, all the values are plotted in one graph.

X-axis is the matrix size and Y-axis is the time in seconds.

Python with NumPy plot is marked in blue color.

Python without NumPy plot is marked in green color.

The Fortran plot is marked in orange color.

So, from this experiment, we can conclude that Interpreted language (python) generally takes more time than compiled language (Fortran).