For this assignment we were given the task to create 3 programs which could be used to measure the time it takes each program with data sets of different sizes. The three programs we developed was python with no use of np arrays and without the use of partial pivoting. The second program was also python with the use of Numpy and it's libraries. The third and final program was Fortran 90 without the use of partial pivoting using the naïve approach. This was done in the effort to identify the differences between interpreted and compiled languages. The data sets quantity selected for this assignment were of 250, 500, 1000, 1500, 2000. Finally, we had to document the results in a table and graph the averages with a total sample set of 75 test 25 from each program with 5 tests for each data set.

```python
#Author: Manuel Holguin
#Date: 09-24-2022
#Description: The code runs Naive Gaussian Elimination without pivoting while recording
#   start time of the program and the ending time. Then does simple addition
#   to print out the output of the time is took to run the program.
#   Additionally the program creates a Matrix of size nXn which is the input
#   from the user. Then is fills the Matrix with random numbers.
from ast import main
import time
import random

main
#take input from user and use it to set the array parameters.
total = input("Input number for n: ")
n = int(total)
#creates and array of zeros to store the solution but not needed for this project
x = [0.0]*n
#Creates and matrix with NXN
B = [[random.randint(0.0,300) for p in range(n)] for q in range(n)]

#Time Starter
start_time = time.time()*1000
#forward elimination
for k in range(n):#Traverses rows
    for i in range(k+1 , n):#Traverses rows fractioning the columns by the previous one
        f = B[i][k] / B[k][k]
        for j in range(n): #k , n-1
            B[i][j] = B[i][j] - f*B[k][j]
#Backward substitution to set the values of the solution
x[n-1] = B[n-2][n-1] / B[n-2][n-2]
for i in range(n-2 , -1 ,-1):
    Sum = B[i][n-1]
    for j in range(i + 1, n):
        Sum = Sum - B[i][j]*x[j]
    x[i] = Sum/B[i][i]
#Print the result time
end_time = time.time()*1000
result = end_time - start_time
print(result)
```

Figure 1 Python code without partial pivoting

python2.py > ...

```python
#Author: Manuel Holguin
#Date: 09-25-2022
#Description: The code runs Gaussian Elimination and pivoting with Numpy while recording
#   start time of the program and the ending time. Then does simple addition
#   to print out the output of the time is took to run the program.
#   Additionally the program creates a Matrix of size n*n+n which is the input
#   from the user. Then is fills the Matrices with random numbers.
from ast import main
#import the numpy library for the assignment
import numpy as np
import time
#numpy fabs used for the absolute value running without numpy linalg library
from numpy import fabs

main
#Get the input from the user and insert into n for array parameters.
total = input("Input number for my n: ")
n = int(total)
#Creates an np array of NxN+1 dimensions and fills it with random ints
B = np.random.uniform(1.0,150.0,(n,n))
#creates an np array of N length for augmented values
C = np.random.uniform(1.0,150.0,(n))
#x vector for storing solutions
x = np.zeros(n)
#Time Starter
start_time = time.time()*1000
#used to solve the system of linear equations but result is not stored not required for this project
#only used to calcuate the time of running gaussian elimination this function uses LU and partial pivoting
np.linalg.solve(B,C)

end_time = time.time()*1000
#calculate the end time of the program
result = end_time - start_time
#Print the result
print("The time is: ")
print(result)

```

Figure 2 Numpy Code with LU and partial pivoting using numpy library linalg.solve

```
!Author: Manuel Holguin
!Date: 09/26/2022
!Description: Purpose of the program is to test the runtime speed of the lexical analyzer
!              and the parser. This fortran code runs Gaussian elimination on matrices
!               of 250,500,1000,1500,2000. Could no implemente input from user was
!               getting a traceback error and the code would not execute. Had to directly
!               input the values. System Time is taken in Miliseconds.
program final

    INTEGER, PARAMETER::n=2000
    INTEGER::i,j
    CHARACTER(100) :: num1char

    real::start,finish
    real :: num1
    REAL::s
    REAL,DIMENSION(n,n+1)::a
    REAL,DIMENSION(n)::x

call get_COMMAND_ARGUMENT(1,num1char)
call random_number(a)!input random numbers into array a

    !READ(1,*)num1
    CALL cpu_time(start)
    DO j=1,n
        DO i=j+1,n
            a(i,:)=a(i,:)-a(j,:)*a(i,j)/a(j,j)
        END DO
    END DO


    DO i=n,1,-1
        s=a(i,n+1)
        DO j=i+1,n
            s=s-a(i,j)*x(j)
        END DO
        x(i)=s/a(i,i)
    END DO

    CALL cpu_time(finish)
    print '("Time = "f12.3, " Milliseconds. ")', (finish-start)*1000
END PROGRAM final
```

Figure 3 Snippet of Fortran code does not use partial pivoting could not fix issue with the compiler to allow system calls to implement user input.

Manuel Holguin 09/26/2022

The times in the following Table are in milliseconds for all tests.

n	code	time 1	Time 2	Time 3	Time 4	Time 5	Average	STDEV.S
250	python	3431.276	2877.395	3239.113	3258.793	3428.425	3247.001	225.656
500	python	27527.005	26825.184	27251.275	27815.879	25544.515	26992.772	888.154
1000	python	237141.661	233067.947	212999.354	222109.551	221669.384	225397.579	9684.841
1500	python	785902.529	753945.053	718700.003	701608.476	808479.432	753727.099	44655.269
2000	python	1961149.456	1935294.537	1908607.089	2114848.723	1955070.831	1974994.127	80825.765
250	Numpy	11.604	5.760	4.419	5.617	6.019	6.684	2.818
500	Numpy	28.508	48.031	39.373	43.810	41.689	40.282	7.313
1000	Numpy	188.292	212.786	207.238	197.524	207.138	202.596	9.700
1500	Numpy	674.783	637.563	629.122	673.502	675.481	658.090	22.799
2000	Numpy	1437.740	1441.377	1445.552	1501.939	1458.698	1457.061	26.308
250	ForTran	13.490	12.774	12.898	13.601	13.620	13.277	0.408
500	ForTran	121.937	123.898	119.730	136.384	124.799	125.350	6.469
1000	ForTran	1141.112	1125.958	1119.075	1105.657	1193.081	1136.977	33.866
1500	ForTran	21310.705	20738.453	19335.648	21278.730	22460.982	21024.904	1134.027
2000	ForTran	46410.609	59943.910	64179.391	63572.617	59220.824	58665.470	7186.930

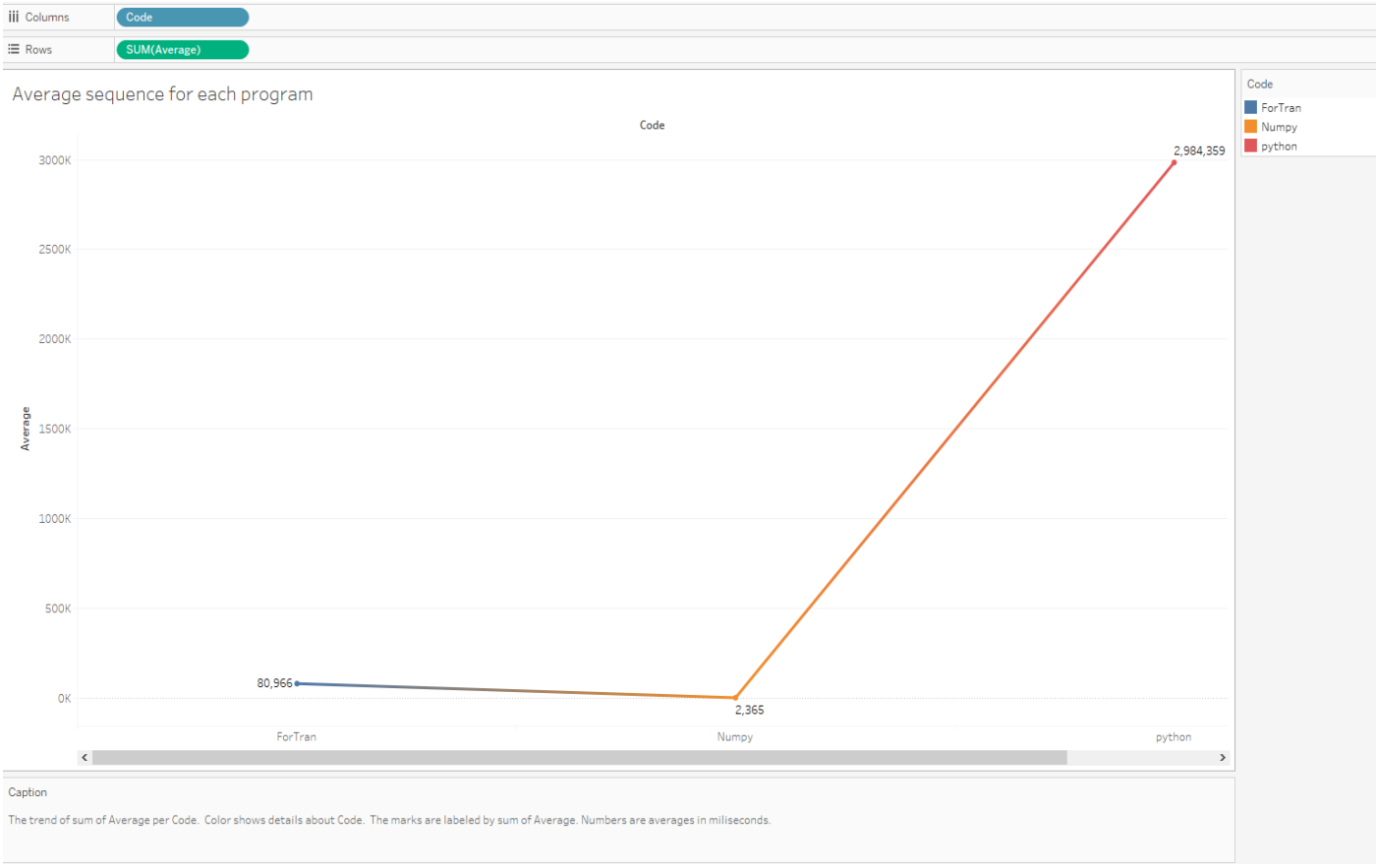

Figure 3 graph potting the sums of averages per program. Showing relativity to each other.

The information which I can gather from the data is although Fortran is a compiled language it can be seen by the plotting of the three averages that the utilization of Numpy for scientific computing has been shown to be faster than the other two implementations. Regardless to believe in which compiled languages are faster as a direct result or not having an interpreter has been proven by contradiction. In this citation Numpy held the lowest average in performing the selected tasks. In conclusion it can be said that it will not be in every case in which a compiled language will be able to outperform an interpreted language as we can see from the proof in the data table and graph.

Appendix

https://numpy.org/doc/stable/reference/routines.linalg.html

https://www.tutorialspoint.com/fortran/fortran_basic_input_output.htm

https://gcc.gnu.org/onlinedocs/gfortran/RANDOM_005fNUMBER.html

https://docs.oracle.com/cd/E19957-01/805-4939/6j4m0vn8a/index.html

https://stackoverflow.com/questions/42544397/generating-an-array-of-random-numbers-in-the-range-of-1-1

https://techgoggler.com/computer-engineering/linear-equations-python-gauss-elimination-method/

https://labmathdu.wordpress.com/gaussian-elimination-without-pivoting/

https://riptutorial.com/fortran/example/26615/passing-command-line-arguments

https://gcc.gnu.org/onlinedocs/gfortran/CPU_005fTIME.html