Programming #4 Comparing interpreted and compiled codes

Keller Sedillo-Garrido

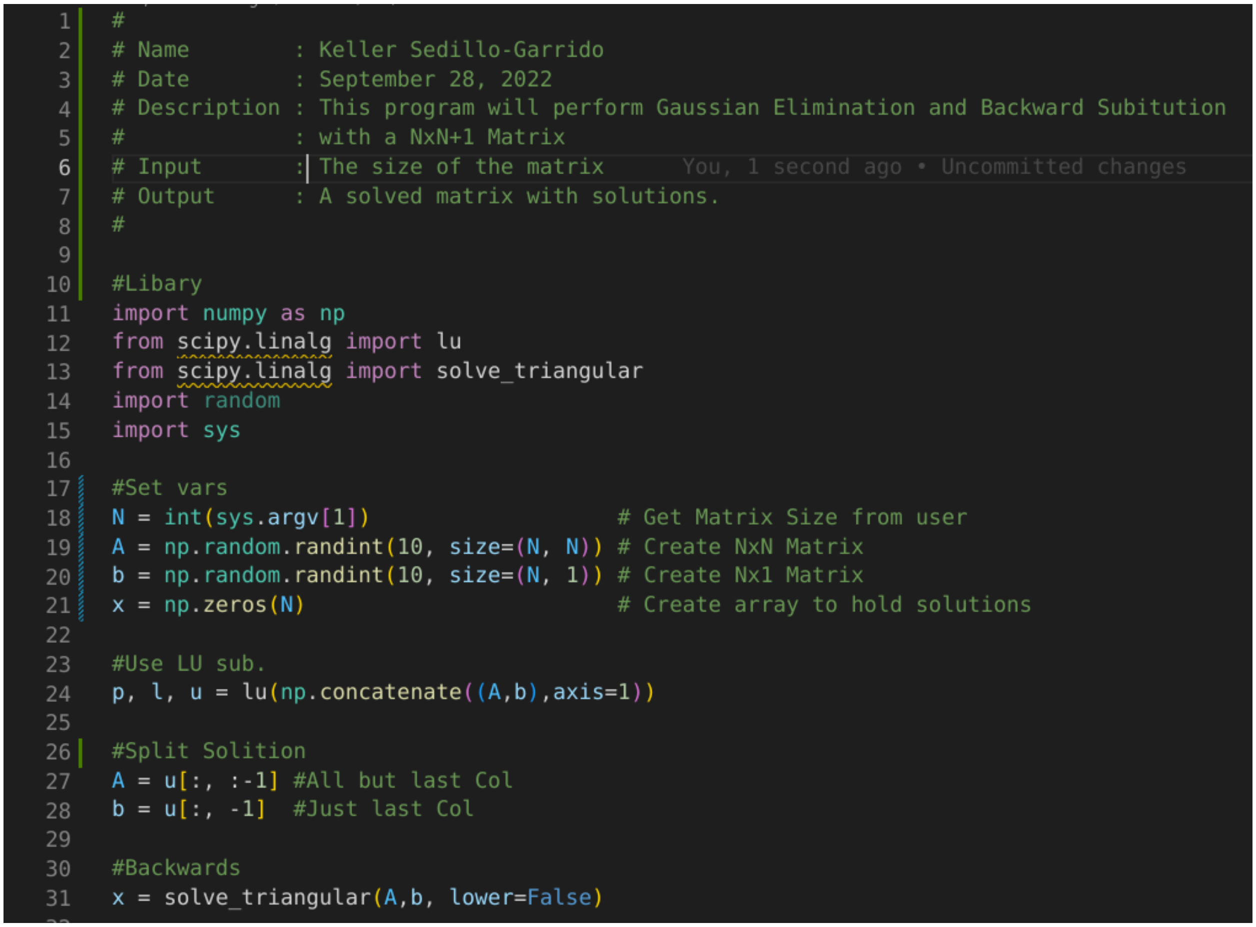
09/29/22

Description:

For the programming assignment, I have created three different programs in order to see which language can perform Gaussian Elimination with backward substitution while also creating a random set of NxN+1 matrices.

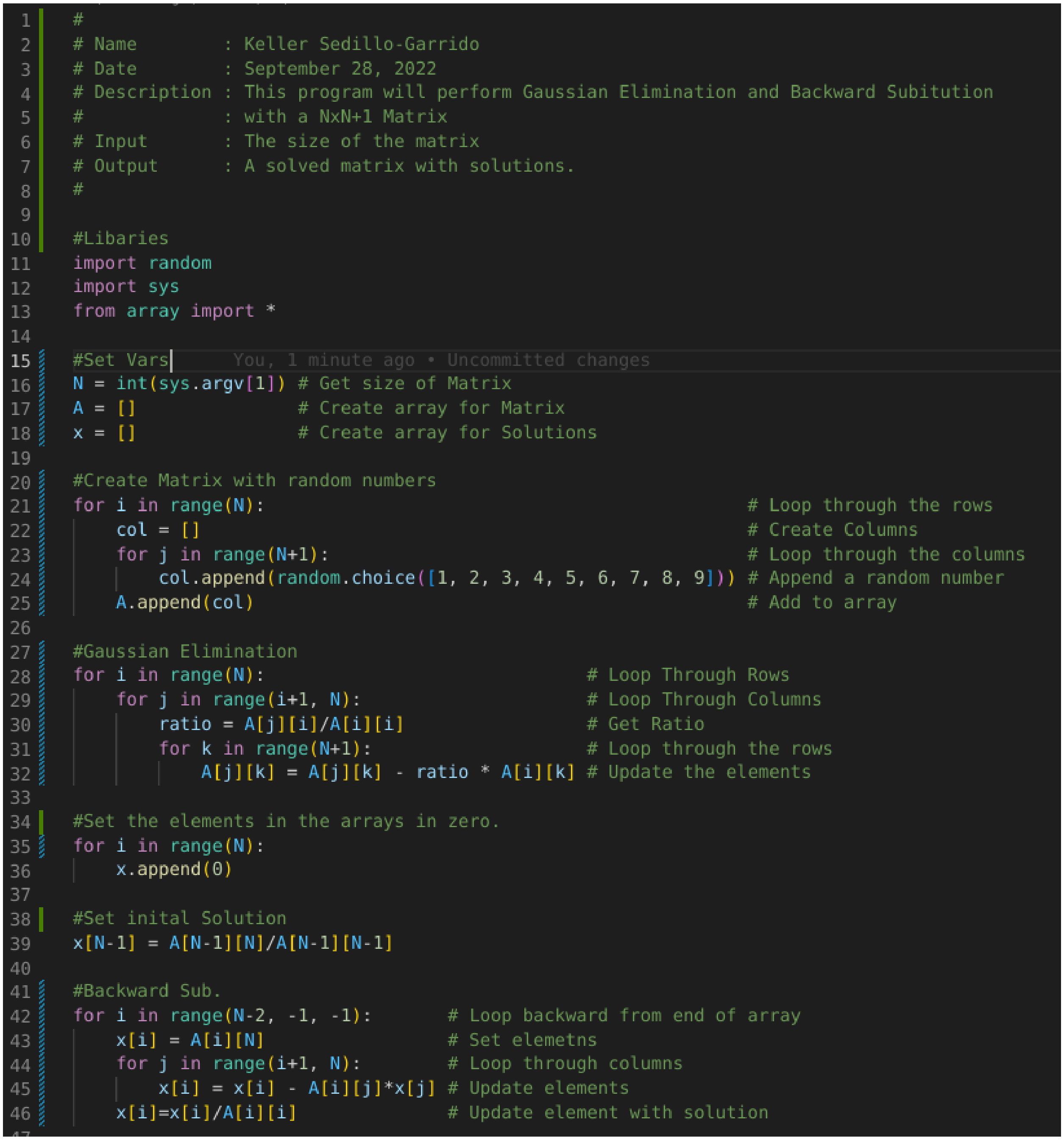
Code:

Numpy:



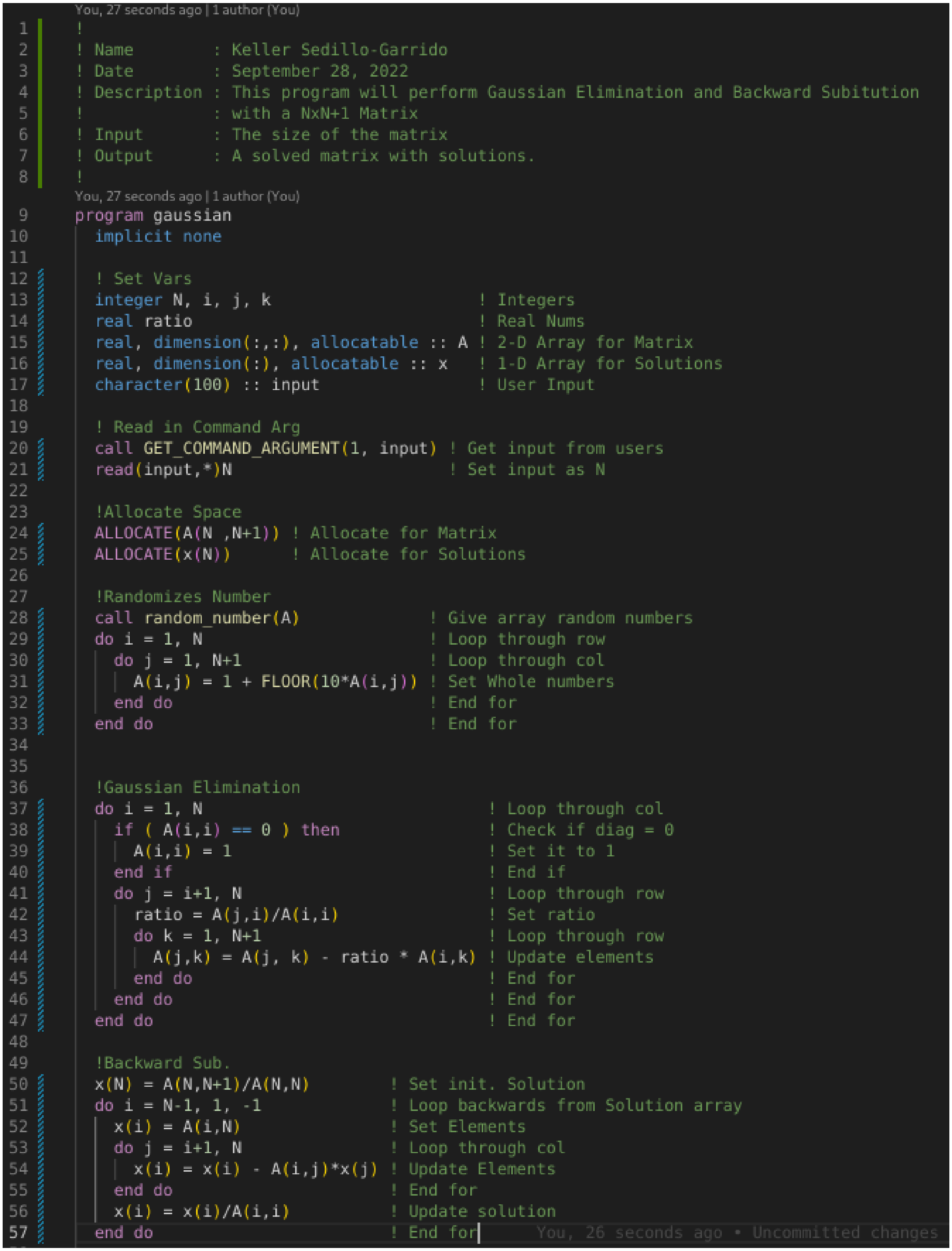
```
#
# Name        : Keller Sedillo-Garrido
# Date        : September 28, 2022
# Description : This program will perform Gaussian Elimination and Backward Subitution
#             : with a NxN+1 Matrix
# Input       : The size of the matrix
# Output      : A solved matrix with solutions.
#

#Libary
import numpy as np
from scipy.linalg import lu
from scipy.linalg import solve_triangular
import random
import sys

#Set vars
N = int(sys.argv[1])                   # Get Matrix Size from user
A = np.random.randint(10, size=(N, N)) # Create NxN Matrix
b = np.random.randint(10, size=(N, 1)) # Create Nx1 Matrix
x = np.zeros(N)                        # Create array to hold solutions

#Use LU sub.
p, l, u = lu(np.concatenate((A,b),axis=1))

#Split Solition
A = u[:, :-1] #All but last Col
b = u[:, -1]  #Just last Col

#Backwards
x = solve_triangular(A,b, lower=False)
```

Python:



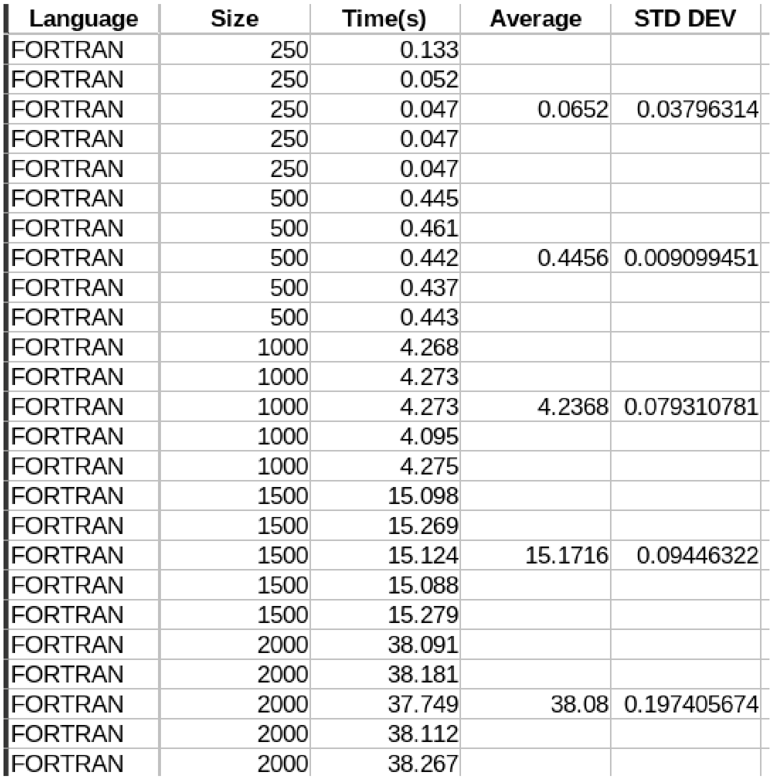
```python
#
# Name        : Keller Sedillo-Garrido
# Date        : September 28, 2022
# Description : This program will perform Gaussian Elimination and Backward Subitution
#             : with a NxN+1 Matrix
# Input       : The size of the matrix
# Output      : A solved matrix with solutions.
#

#Libaries
import random
import sys
from array import *

#Set Vars      You, 1 minute ago • Uncommitted changes
N = int(sys.argv[1]) # Get size of Matrix
A = []               # Create array for Matrix
x = []               # Create array for Solutions

#Create Matrix with random numbers
for i in range(N):                                             # Loop through the rows
    col = []                                                   # Create Columns
    for j in range(N+1):                                       # Loop through the columns
        col.append(random.choice([1, 2, 3, 4, 5, 6, 7, 8, 9])) # Append a random number
    A.append(col)                                              # Add to array

#Gaussian Elimination
for i in range(N):                              # Loop Through Rows
    for j in range(i+1, N):                     # Loop Through Columns
        ratio = A[j][i]/A[i][i]                 # Get Ratio
        for k in range(N+1):                    # Loop through the rows
            A[j][k] = A[j][k] - ratio * A[i][k] # Update the elements

#Set the elements in the arrays in zero.
for i in range(N):
    x.append(0)

#Set inital Solution
x[N-1] = A[N-1][N]/A[N-1][N-1]

#Backward Sub.
for i in range(N-2, -1, -1):       # Loop backward from end of array
    x[i] = A[i][N]                 # Set elemetns
    for j in range(i+1, N):        # Loop through columns
        x[i] = x[i] - A[i][j]*x[j] # Update elements
    x[i]=x[i]/A[i][i]              # Update element with solution
```

Fortran:

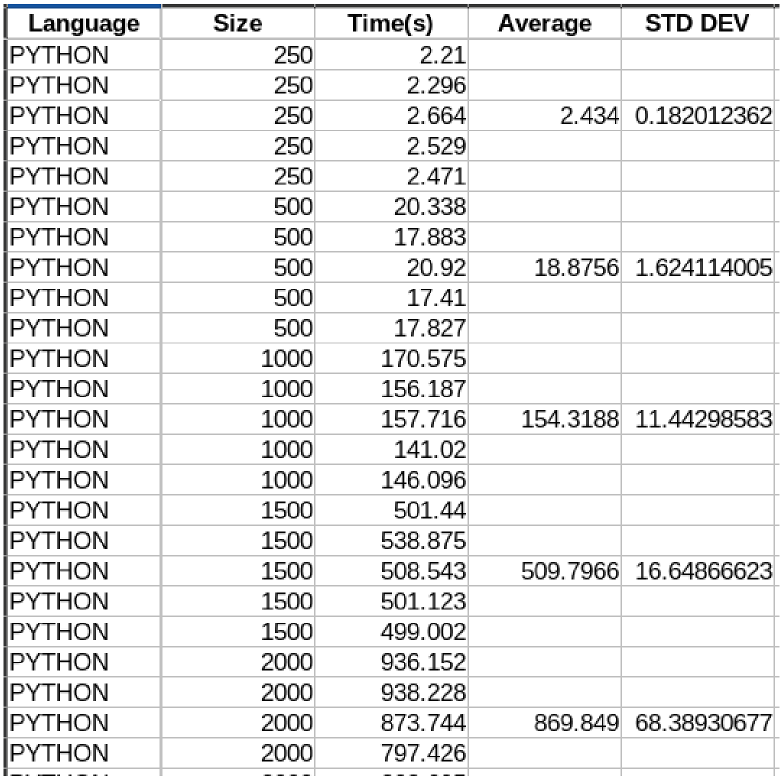


```
You, 27 seconds ago | 1 author (You)
!
! Name        : Keller Sedillo-Garrido
! Date        : September 28, 2022
! Description : This program will perform Gaussian Elimination and Backward Subitution
!             : with a NxN+1 Matrix
! Input       : The size of the matrix
! Output      : A solved matrix with solutions.
!
You, 27 seconds ago | 1 author (You)
program gaussian
  implicit none

  ! Set Vars
  integer N, i, j, k                     ! Integers
  real ratio                             ! Real Nums
  real, dimension(:,:), allocatable :: A ! 2-D Array for Matrix
  real, dimension(:), allocatable :: x   ! 1-D Array for Solutions
  character(100) :: input                ! User Input

  ! Read in Command Arg
  call GET_COMMAND_ARGUMENT(1, input) ! Get input from users
  read(input,*)N                      ! Set input as N

  !Allocate Space
  ALLOCATE(A(N ,N+1)) ! Allocate for Matrix
  ALLOCATE(x(N))      ! Allocate for Solutions

  !Randomizes Number
  call random_number(A)               ! Give array random numbers
  do i = 1, N                         ! Loop through row
    do j = 1, N+1                     ! Loop through col
      A(i,j) = 1 + FLOOR(10*A(i,j))   ! Set Whole numbers
    end do                            ! End for
  end do                              ! End for


  !Gaussian Elimination
  do i = 1, N                               ! Loop through col
    if ( A(i,i) == 0 ) then                 ! Check if diag = 0
      A(i,i) = 1                            ! Set it to 1
    end if                                  ! End if
    do j = i+1, N                           ! Loop through row
      ratio = A(j,i)/A(i,i)                 ! Set ratio
      do k = 1, N+1                         ! Loop through row
        A(j,k) = A(j, k) - ratio * A(i,k)   ! Update elements
      end do                                ! End for
    end do                                  ! End for
  end do                                    ! End for

  !Backward Sub.
  x(N) = A(N,N+1)/A(N,N)        ! Set init. Solution
  do i = N-1, 1, -1             ! Loop backwards from Solution array
    x(i) = A(i,N)               ! Set Elements
    do j = i+1, N               ! Loop through col
      x(i) = x(i) - A(i,j)*x(j) ! Update Elements
    end do                      ! End for
    x(i) = x(i)/A(i,i)          ! Update solution
  end do                        ! End for      You, 26 seconds ago • Uncommitted changes
```

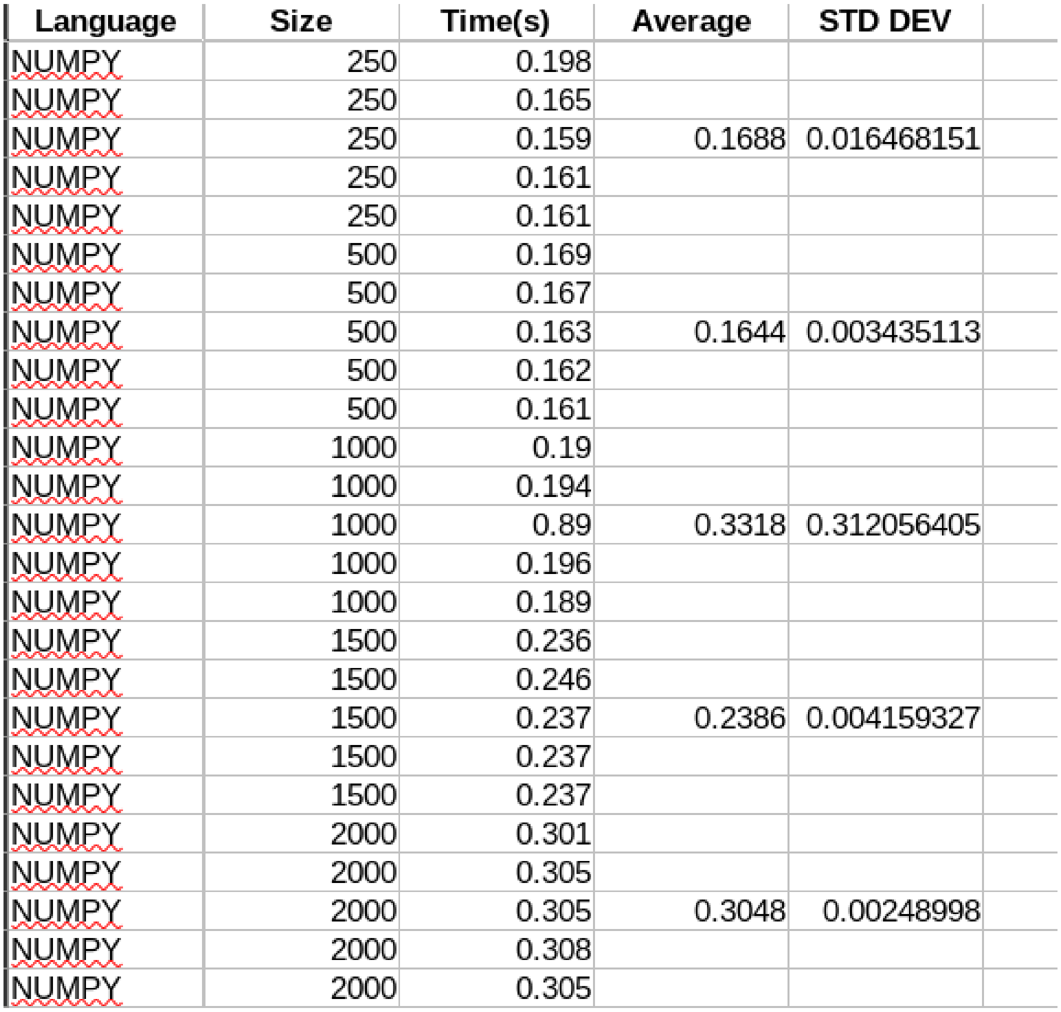
Table:



Language	Size	Time(s)	Average	STD DEV
FORTRAN	250	0.133		
FORTRAN	250	0.052		
FORTRAN	250	0.047	0.0652	0.03796314
FORTRAN	250	0.047		
FORTRAN	250	0.047		
FORTRAN	500	0.445		
FORTRAN	500	0.461		
FORTRAN	500	0.442	0.4456	0.009099451
FORTRAN	500	0.437		
FORTRAN	500	0.443		
FORTRAN	1000	4.268		
FORTRAN	1000	4.273		
FORTRAN	1000	4.273	4.2368	0.079310781
FORTRAN	1000	4.095		
FORTRAN	1000	4.275		
FORTRAN	1500	15.098		
FORTRAN	1500	15.269		
FORTRAN	1500	15.124	15.1716	0.09446322
FORTRAN	1500	15.088		
FORTRAN	1500	15.279		
FORTRAN	2000	38.091		
FORTRAN	2000	38.181		
FORTRAN	2000	37.749	38.08	0.197405674
FORTRAN	2000	38.112		
FORTRAN	2000	38.267		

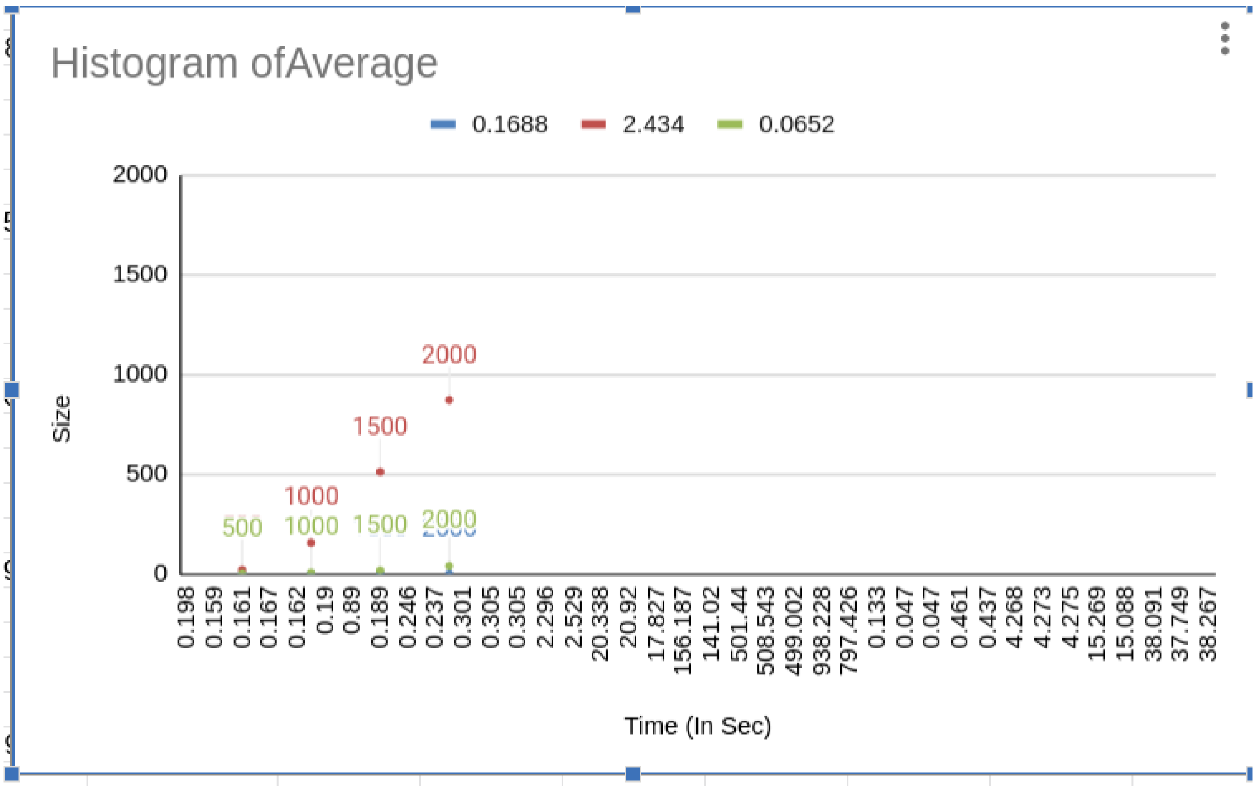


Language	Size	Time(s)	Average	STD DEV
PYTHON	250	2.21		
PYTHON	250	2.296		
PYTHON	250	2.664	2.434	0.182012362
PYTHON	250	2.529		
PYTHON	250	2.471		
PYTHON	500	20.338		
PYTHON	500	17.883		
PYTHON	500	20.92	18.8756	1.624114005
PYTHON	500	17.41		
PYTHON	500	17.827		
PYTHON	1000	170.575		
PYTHON	1000	156.187		
PYTHON	1000	157.716	154.3188	11.44298583
PYTHON	1000	141.02		
PYTHON	1000	146.096		
PYTHON	1500	501.44		
PYTHON	1500	538.875		
PYTHON	1500	508.543	509.7966	16.64866623
PYTHON	1500	501.123		
PYTHON	1500	499.002		
PYTHON	2000	936.152		
PYTHON	2000	938.228		
PYTHON	2000	873.744	869.849	68.38930677
PYTHON	2000	797.426		



Language	Size	Time(s)	Average	STD DEV
NUMPY	250	0.198		
NUMPY	250	0.165		
NUMPY	250	0.159	0.1688	0.016468151
NUMPY	250	0.161		
NUMPY	250	0.161		
NUMPY	500	0.169		
NUMPY	500	0.167		
NUMPY	500	0.163	0.1644	0.003435113
NUMPY	500	0.162		
NUMPY	500	0.161		
NUMPY	1000	0.19		
NUMPY	1000	0.194		
NUMPY	1000	0.89	0.3318	0.312056405
NUMPY	1000	0.196		
NUMPY	1000	0.189		
NUMPY	1500	0.236		
NUMPY	1500	0.246		
NUMPY	1500	0.237	0.2386	0.004159327
NUMPY	1500	0.237		
NUMPY	1500	0.237		
NUMPY	2000	0.301		
NUMPY	2000	0.305		
NUMPY	2000	0.305	0.3048	0.00248998
NUMPY	2000	0.308		
NUMPY	2000	0.305		

Graph:



Analysis:

As we see with the data points, Numpy was the quicker program compared to the other two with Fortran averaging second and python ending last place. My guess to these results are due to the fact that Numpy uses both C code and python. This means that numpy is a hybrid of both interpreted language and compiled classes making processing calculations faster. I think python ends up last because of its interpitated nature and the fact that it needs to go through line by line compared to compiled code.