Xiana Lara
September 28, 2020
Programming #4
CS 471

## Programming #4 Comparing interpreted and compiled codes

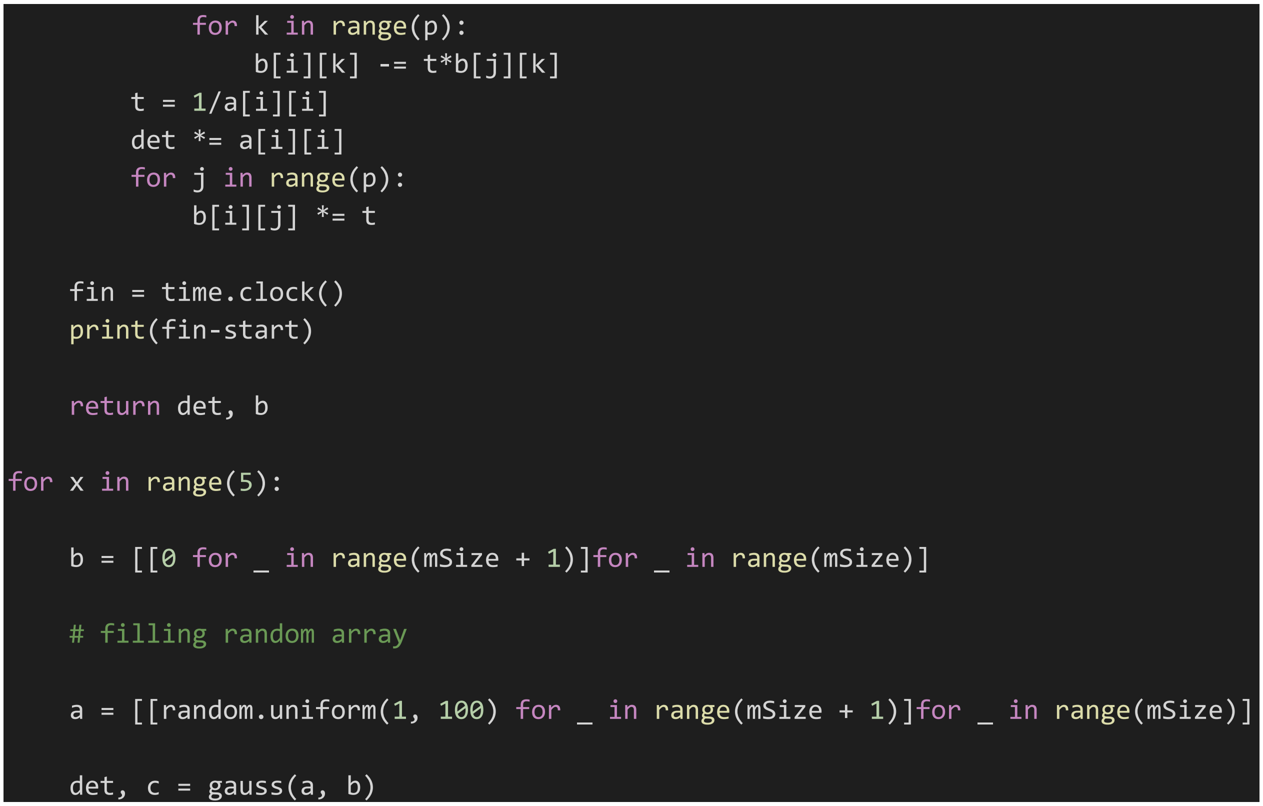
In this assignment, the task was to have a program allocate and populate a matrix using random numbers. The program should then start a clock, run Gaussian Elimination and back substitution, and then the stop time. After those steps are completed the program should print the time it took to run the code. This has allowed me to compare interpreted and compiled languages.

```
# Xiana Lara
# September 28, 2020
# Programming 4: Comparing interpreted and compiled codes

# Original Sources
# https://numpy.org/learn/

# Current Purpose
# Comparing interpreted and compiled languages by measuring
# the time it takes to run the programs with different data sizes
# Current Language: python

# Input: Square matrices size
# Output: The amount of time in seconds each program ran for

import numpy
import time

# read matrix size from user
mSize = int(input("Square Matrix Size: "))

for x in range(5):
    a, b= numpy.random.rand(mSize, mSize), numpy.random.rand(mSize, 1)

    start = time.clock()
    mat = numpy.linalg.solve(a, b)
    fin = time.clock()

    print(fin - start)
```

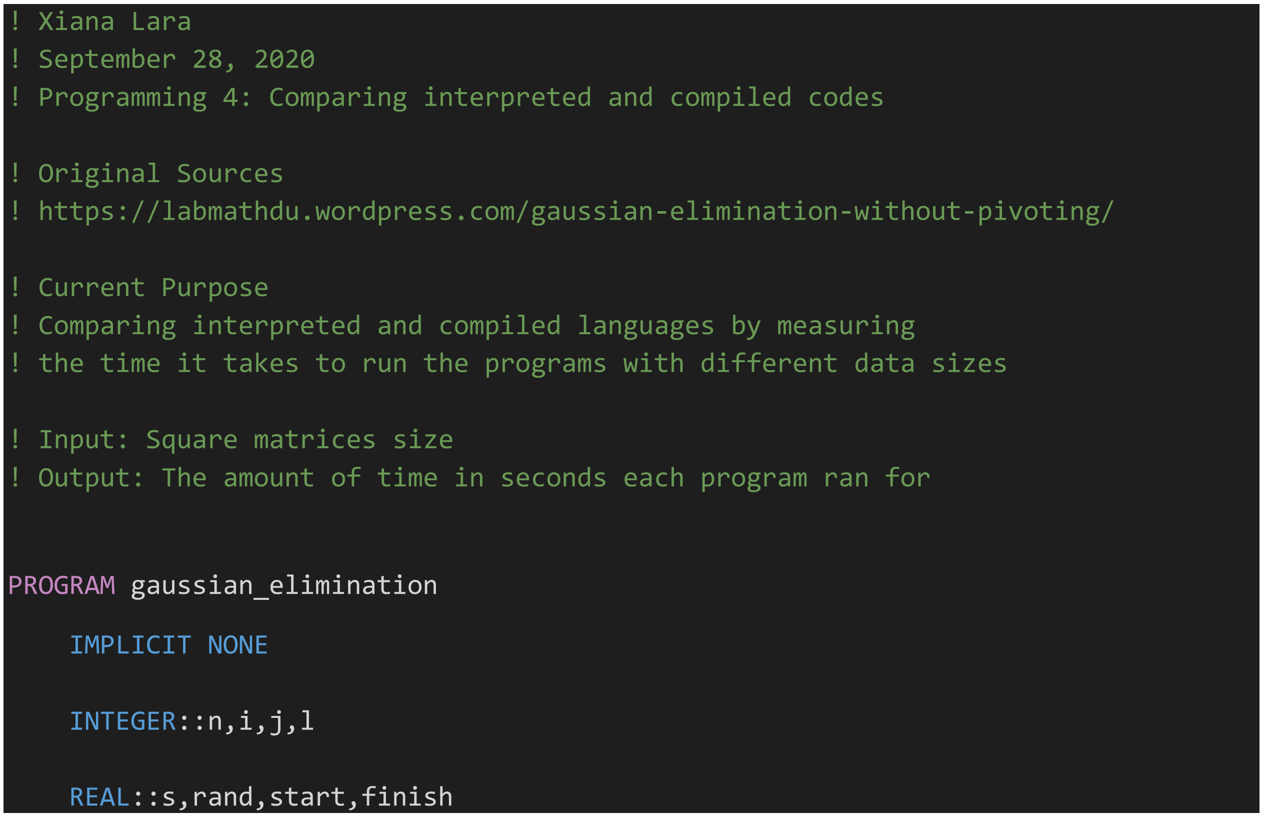
```
# Xiana Lara
# September 28, 2020
# Programming 4: Comparing interpreted and compiled codes

# Original Sources
# https://rosettacode.org/wiki/Gaussian_elimination#Python
```

Xiana Lara
September 28, 2020
Programming #4
CS 471

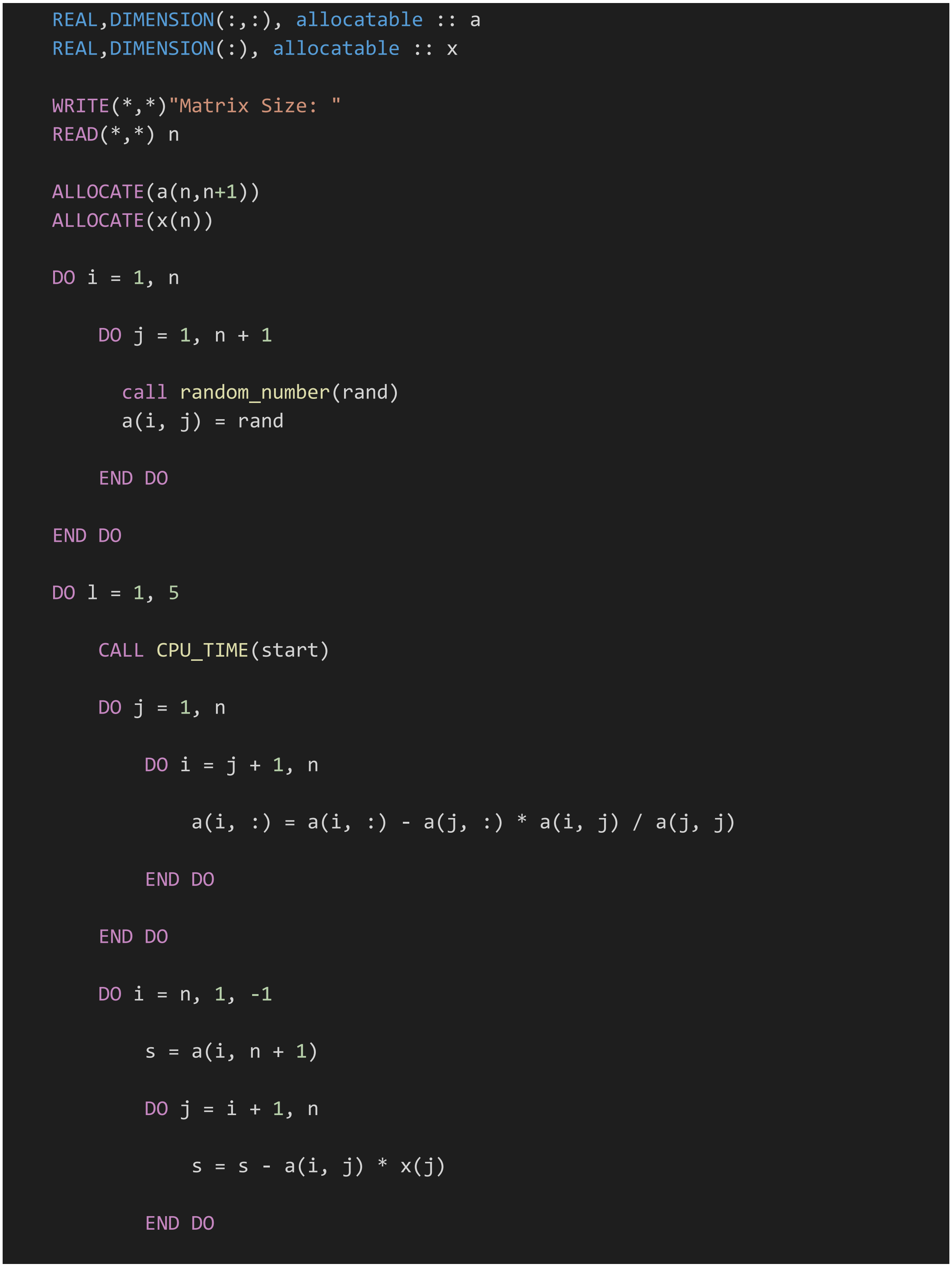
```python
# Current Purpose:
# Comparing interpreted and compiled languages by measuring
# the time it takes to run the programs with different data sizes
# Current Language: python

# Input: Square matrices size
# Output: The amount of time in seconds each program ran for

import copy
from fractions import Fraction
import random
import time


mSize = int(input("Square Matrix Size: "))

def gauss(a, b):

    start = time.clock()
    a = copy.deepcopy(a)
    b = copy.deepcopy(b)
    n = len(a)
    p = len(b[0])
    det = 1
    for i in range(n - 1):
        k = i
        for j in range(i + 1, n):
            if abs(a[j][i]) > abs(a[k][i]):
                k = j
        if k != i:
            a[i], a[k] = a[k], a[i]
            b[i], b[k] = b[k], b[i]
            det = -det

        for j in range(i + 1, n):
            t = a[j][i]/a[i][i]
            for k in range(i + 1, n):
                a[j][k] -= t*a[i][k]
            for k in range(p):
                b[j][k] -= t*b[i][k]

    for i in range(n - 1, -1, -1):
        for j in range(i + 1, n):
            t = a[i][j]
```



```
            for k in range(p):
                b[i][k] -= t*b[j][k]
        t = 1/a[i][i]
        det *= a[i][i]
        for j in range(p):
            b[i][j] *= t

    fin = time.clock()
    print(fin-start)

    return det, b

for x in range(5):

    b = [[0 for _ in range(mSize + 1)]for _ in range(mSize)]

    # filling random array

    a = [[random.uniform(1, 100) for _ in range(mSize + 1)]for _ in range(mSize)]

    det, c = gauss(a, b)
```



```
! Xiana Lara
! September 28, 2020
! Programming 4: Comparing interpreted and compiled codes

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

! Current Purpose
! Comparing interpreted and compiled languages by measuring
! the time it takes to run the programs with different data sizes

! Input: Square matrices size
! Output: The amount of time in seconds each program ran for


PROGRAM gaussian_elimination

    IMPLICIT NONE

    INTEGER::n,i,j,l

    REAL::s,rand,start,finish
```



```
REAL,DIMENSION(:,:), allocatable :: a
REAL,DIMENSION(:), allocatable :: x

WRITE(*,*)"Matrix Size: "
READ(*,*) n

ALLOCATE(a(n,n+1))
ALLOCATE(x(n))

DO i = 1, n

    DO j = 1, n + 1

      call random_number(rand)
      a(i, j) = rand

    END DO

END DO

DO l = 1, 5

    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
```

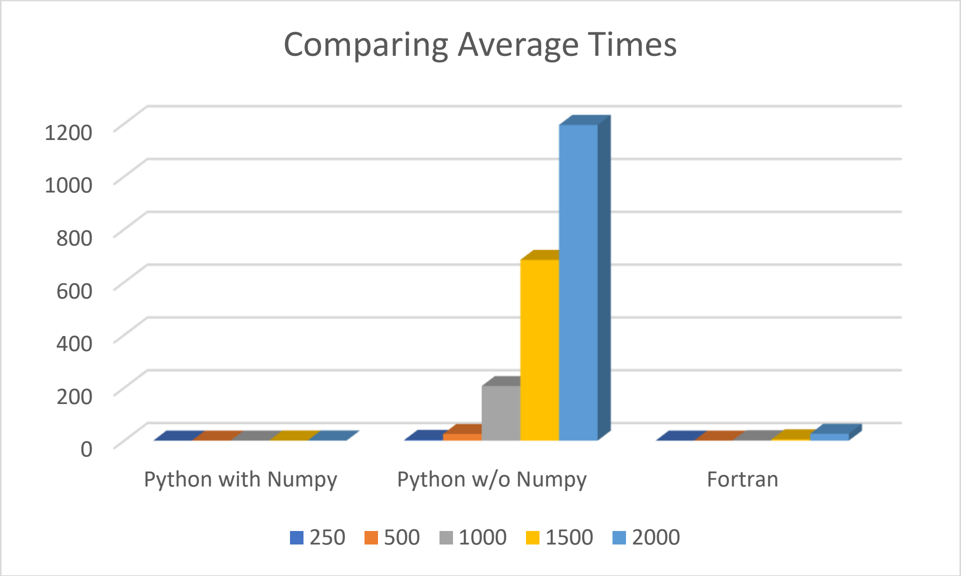
```
            x(i) = s / a(i, i)

        END DO

        CALL CPU_TIME(finish)

        WRITE (*,*) (finish-start)

    END DO

END PROGRAM
```

Python with Numpy					
	250	500	1000	1500	2000
Test 1	0.028146	0.061285	0.140235	0.324478	0.647227
Test 2	0.02566	0.066294	0.164031	0.374531	0.633534
Test 3	0.025611	0.066556	0.163474	0.377881	0.649288
Test 4	0.025324	0.066338	0.164135	0.373115	0.638521
Test 5	0.025721	0.066435	0.162656	0.377732	0.647484
Average	0.026092	0.065382	0.158906	0.365547	0.643211
Standard D	0.001158	0.002292	0.010454	0.02305	0.006837

Python w/o Numpy					
	250	500	1000	1500	2000
Test 1	2.916339	23.94309	196.4518	666.283	1208.987
Test 2	2.913794	24.10212	197.1202	684.9095	1197.087
Test 3	2.930442	25.13861	206.679	721.4952	1178.075
Test 4	2.931379	25.61127	214.2105	653.1022	1201.638
Test 5	2.934495	26.15521	217.6647	695.1361	1189.753
Average	2.92529	24.99006	206.4253	684.1852	1195.108
Standard D	0.009495	0.955263	9.657318	26.45337	11.8107

Fortran					
	250	500	1000	1500	2000
Test 1	0.03125	0.171875	1.3125	4.84375	22.5
Test 2	0.015625	0.15625	1.265625	4.734375	22.23438
Test 3	0.03125	0.15625	1.3125	5.015625	29.98438
Test 4	0.015625	0.171875	1.3125	5.046875	29.40625
Test 5	0.03125	0.171875	1.28125	4.875	25.625
Average	0.02500	0.165625	1.296875	4.903125	25.95
Standard D	0.008558	0.008558	0.022097	0.128563	3.675454

Xiana Lara
September 28, 2020
Programming #4
CS 471



Though python using NumPy is the fastest, Fortran is significantly faster than python without NumPy. This is interesting because python, as a language, is more advanced than Fortran. Fortran is faster because it there is no need to interpret it since it is a compiled language you skip a whole step, which apparently saves a lot of time.