The problem for this assignment is to implement Gaussian elimination with back substitution using 3 different implementations: Fortran, Python w/ NumPy, and Python w/o NumPy. Our goal is to compare the time it takes to perform Gaussian elimination on matrix squares with sizes 250, 500, 1000, 1500 and 2000. The program takes the size of the matrix as input and fills it with random numbers. We then start a timer and run Gaussian elimination, then stop the timer upon completion. The program only outputs the time, not the solution.

**Note:** I tried 2 implementations for Python with NumPy. One uses NumPy only to fill the arrays and then performs the elimination using loops. The other used NumPy to fill the arrays and then used the function numpy.linalg.solve to solve the matrix. I chose to not use the linalg function because one, there was a massive time difference between using it and for-loops, and two, because I couldn’t find out exactly how it was working. The NumPy documentation said it’s *similar* to LU decomposition. But I wanted to be absolutely sure that my program performed only Gaussian Elimination and back substitution without pivoting, and I can clearly see that by using traditional for-loops. I was also thrown off by the time disparity, with numpy.linalg.solve running the 2500 square matrix in only minutes compared to the hour with my implementation.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Fortran** | **Python with NumPy** | **Python without NumPy** |
| **Avg Time in Seconds** | | | |
| 250 Matrix | 0.01570280046 s avg | 6.639442 s avg | 2.219504 s avg |
| 500 Matrix | 0.15366939902 s avg | 52.672484 s avg | 17.820722 s avg |
| 1000 Matrix | 1.65147073268 s avg | 421.845636 s avg | 143.25374 s avg |
| 1500 Matrix | 6.16338310242 s avg | 1430.918686 s avg | 488.144448 s avg |
| 2000 Matrix | 14.69540252684 s avg | 3559.784584 s avg | 1139.258002 s avg |
| **Standard Deviation** | | | |
| 250 Matrix | 0.0003694526599503 | 0.074639372291037 | 0.025718965453532 |
| 500 Matrix | 0.00020806425104947 | 0.25829618151262 | 0.078716773155409 |
| 1000 Matrix | 0.740230406715 | 2.535437628013 | 1.6854007409397 |
| 1500 Matrix | 1.1916958243583 | 11.500300559145 | 6.1976725030656 |
| 2000 Matrix | 2.3759854142589 | 373.69900613805 | 10.752734259555 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Fortran** | **Python with NumPy** | **Python without NumPy** |
| **Time In Seconds to Run Gaussian Elimination** | | | |
| 250 Matrix | 0.0180960000  0.0156580005  0.0155510008  0.0155420005  0.0155060003 | 6.75859  6.61317  6.68965  6.55762  6.57818 | 2.22668  2.18790  2.24902  2.19076  2.24316 |
| 500 Matrix | 0.1538799852  0.1534840018  0.1535440087  0.1535570025  0.1535899937 | 52.83777  52.39568  53.09123  52.57846  52.45928 | 17.81674  17.86984  17.93991  17.75913  17.71799 |
| 1000 Matrix | 1.2932420969  1.3175150156  1.2824258804  3.1488640308  1.2904789448 | 426.37494  420.63157  422.81922  419.63487  419.76758 | 142.27477  142.14290  142.57945  146.60248  142.66910 |
| 1500 Matrix | 7.1683769226  4.7121329308  7.1334838867  4.6960039139  7.1069178581 | 1428.37137  1440.19472  1427.67631  1445.79118  1412.55985 | 482.27168  492.74093  491.42062  495.00916  479.27985 |
| 2000 Matrix | 11.8217601776  16.6834888458  16.6385231018  11.7498779297  16.5833625793 | 3406.68932  3356.21022  3364.32780  3365.34332  4306.35226 | 1136.19115  1160.47635  1135.26734  1133.24824  1131.10693 |

As we can see by the data, the Fortran implementation is the quickest, followed by Python without NumPy, and lastly Python with NumPy. Based on the data, we can see that a compiled language runs the quickest, which I would attribute to the compiler being able to optimize certain operations. I think Python without NumPy runs faster because it can directly fill the arrays via for-loops, whereas the NumPy implementation runs slower because the program has to access the NumPy library and the functions may just run slower than a for loop and the random library.

**Fortran**

!Joseph Camacho-Terrazas

!9/26/2020

!Input: Takes a command line argument for the matrix size

!Output: The time it takes for Gaussian Elimination to run on the matrix

!Preconditions: Program must be run with 1 argument. Argument must allow for Gaussian Elimination to run without error.

!Postconditions: Program will output the total time it took to run Gaussian Elimination on the matrix.

PROGRAM program4

    IMPLICIT NONE

    !Matrix declarations

    INTEGER::i,j

    REAL::s

    !Create dynamic arrays to allow for command line args

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

    REAL,DIMENSION(:),allocatable::x

    !Char and int for casting arg to integer

    CHARACTER(100)::numinputchar

    INTEGER::numinput

    !Trackers for CPU time

    REAL::start, finish

    !Get arg1 and cast to int

    CALL getarg(1,numinputchar)

    READ(numinputchar,'(I10)')numinput

    !Construct the arrays using arg1 input

    ALLOCATE(a(numinput,numinput))

    ALLOCATE(x(numinput))

    !Fills the array with random numbers

    CALL random\_number(a)

    !Start timer

    CALL cpu\_time(start)

    !----------Fortran Gaussian Elimination Without Pivoting Source----------

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

    !File related stuff is not needed for this implementation, but I left it to show the full source

    !OPEN(1,FILE='input.txt')

    !OPEN(2,FILE='output.txt')

    !READ(1,\*)((a(i,j),j=1,n+1),i=1,n)

    !WRITE(2,8)"Augmented Matrix",((a(i,j),j=1,numinput+1),i=1,numinput)

    DO j=1,numinput

        DO i=j+1,numinput

            a(i,:)=a(i,:)-a(j,:)\*a(i,j)/a(j,j)

        END DO

    END DO

    !WRITE(2,8)"After Gaussian Elimination",((a(i,j),j=1,numinput+1),i=1,numinput)

    DO i=numinput,1,-1

        s=a(i,numinput+1)

        DO j=i+1,numinput

            s=s-a(i,j)\*x(j)

        END DO

        x(i)=s/a(i,i)

    END DO

    !WRITE(2,9)"X=",(x(i),i=1,numinput)

    !8 FORMAT(a,/,3(4(f7.2,3x),/))

    !9 FORMAT(a,/,3(f7.2,/))

    !End the timer and print result

    CALL cpu\_time(finish)

    print '("Time = ",f20.10," seconds")',finish-start

END PROGRAM

**Python with NumPy**

#Joseph Camacho-Terrazas

#Program4 Python Numpy Implementation

#9/26/2020

#Input: Takes a command line argument for the matrix size

#Output: The time it takes for Gaussian Elimination to run on the matrix

#Preconditions: Program must be run with 1 argument. Argument must allow for Gaussian Elimination to run without error.

#Postconditions: Program will output the total time it took to run Gaussian Elimination on the matrix.

# Imports

import sys

import time

import numpy as np

# Read command line arguments as an int to pass to array size

n = int(sys.argv[1])

#----------Python Gaussian Elimination with Numpy Source----------

# Credit: https://www.codesansar.com/numerical-methods/gauss-elimination-method-python-program.htm

# Making numpy array of n size and initializing

# to zero for storing solution vector

x = np.zeros(n)

# Fill array with random numbers

a = np.random.rand(n, n+1)

# Applying Gauss Elimination

# Start Timer

start = time.time()

for i in range(n):

    if a[i][i] == 0.0:

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

    for j in range(i+1, n):

        ratio = a[j][i]/a[i][i]

        for k in range(n+1):

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

# Back Substitution

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]

#End Timer and print results

finish = time.time()

print('Time = %.5f seconds'%(finish-start))

**Python without NumPy**

#Joseph Camacho-Terrazas

#Program4 Python Numpy Implementation

#9/26/2020

#Input: Takes a command line argument for the matrix size

#Output: The time it takes for Gaussian Elimination to run on the matrix

#Preconditions: Program must be run with 1 argument. Argument must allow for Gaussian Elimination to run without error.

#Postconditions: Program will output the total time it took to run Gaussian Elimination on the matrix.

# Imports

import sys

import time

import random

#Read command line arguments as an int to create the array dimensions

n = int(sys.argv[1])

#Set the size of the matrix

cols = n+1

rows = n

#Create 2 dimensional array of zeroes

x = [[0 for \_ in range(cols)]for \_ in range(rows)]

#Create 2 dimensional array of random floats

a = [[random.uniform(1.50,10.50) for \_ in range(cols)]for \_ in range(rows)]

#----------Python Gaussian Elimination with Numpy Source----------

# Credit: https://www.codesansar.com/numerical-methods/gauss-elimination-method-python-program.htm

# ----Numpy code is not used in this implementation, I just left it in to show the full source-----

# Making numpy array of n size and initializing

# to zero for storing solution vector

#x = np.zeros(n)

# Fill array with random numbers

#a = np.random.rand(n, n+1)

# Applying Gauss Elimination

# Start Timer

start = time.time()

for i in range(n):

    if a[i][i] == 0.0:

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

    for j in range(i+1, n):

        ratio = a[j][i]/a[i][i]

        for k in range(n+1):

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

# Back Substitution

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]

#End Timer and print results

finish = time.time()

print('Time = %.5f seconds'%(finish-start))