Programming #4 –

Comparing Interpreted and Compiled Codes

**GaussWithNumpy.py**

# Name: Tony Maldonado

# Date: September 27, 2020

#

# Description: This program is an implementation of

# the Gaussian Elimination algorithm in Python

# using the Numpy module.

#

# Input: A number n for the size of the matrix

#

# Output: The time t for which it took to compute

# the matrix elimination

#

# Preconditions: The input (for matrix size) must be a

# positive integer

#

# Postconditions: None.

#

# Source code: https://learnche.org/3E4/Assignment\_2\_-\_2010\_-\_Solution/Bonus\_question

import numpy as np

import random

import time

def main():

n = input("Enter a positive integer n for matrix size: ")

n = int(n)

# Fill matrix with random numbers

A = np.random.rand(n,n)

b = np.random.rand(n)

print('')

print("Starting Gaussian Elimination on matrix..")

print("Starting timer...")

# Start timer; BEFORE Gaussian Elimination

start = time.time()

# Run Gaussian Elimination on matrix

x = np.linalg.solve(A,b)

print("Stopping timer...")

print('')

# Stop the timer

end = time.time()

# Output the time solution took

print('Time it took to run was: ', end - start, 'seconds.')

print('')

if \_\_name\_\_ == '\_\_main\_\_':

main()

**GaussNoNumpy.py**

# Name: Tony Maldonado

# Date: September 27, 2020

#

# Description: This program is a modification of the

# implementation of the Gaussian Elimination

# algorithm in Python, /without/ using the Numpy module.

#

# Note: This program uses a no pivoting method.

#

# Input: A number n for the size of the matrix

#

# Output: The time t for which it took to compute

# the matrix elimination

#

# Preconditions: The input (for matrix size) must be a

# positive integer

#

# Postconditions: None.

#

# Source code: https://learnche.org/3E4/Assignment\_2\_-\_2010\_-\_Solution/Bonus\_question

import random

import time

# Calculates the forward part of Gaussian Elimination

def forward\_elimination (A, b, n):

for row in range(0, n-1):

for i in range(row + 1, n):

factor = A[i][row]/A[row][row]

for j in range(row, n):

A[i][j] = A[i][j] - factor \* A[row][j]

b[i] = b[i] - factor \* b[row]

return A, b

# Calculates the back substitution part of Gaussian Elimination

def back\_substitution(a, b, n):

x = [[0. for i in range(n)] for k in range(n)]

x[n-1] = b[n-1] / a[n-1][n-1]

for row in range (n-2, -1, -1):

sums = b[row]

for j in range(row + 1, n):

sums = sums - a[row][j] \* x[j]

x[row] = sums / a[row][row]

return x

# Calculates the Gaussian Elimination without pivoting

def gauss(A,b,n):

A, b = forward\_elimination(A,b,n)

return back\_substitution(A,b,n)

def main():

n = input("Enter a positive integer n for matrix size: ")

n = int(n)

A = []

# Fill matrix with random numbers

for i in range(n):

b = []

for j in range(n):

b.append(random.random())

A.append(b)

print('')

print("Starting Gaussian Elimination on matrix..")

print("Starting timer...")

# Start timer

start = time.time()

gauss(A,b,n) #solving

print("Stopping timer...")

print('')

# Stop the timer

end = time.time()

# Output the time solution took

print('Time it took to run was: ', end - start, 'seconds.')

print('')

if \_\_name\_\_ == '\_\_main\_\_':

main()

**GaussFortan.f90**

! Name: Tony Maldonado

! Date: September 27, 2020

!

! Description: This program is an implementation of

! the Gaussian Elimination algorithm in Fortran.

!

! Input: A number n for the size of the matrix

!

! Output: The time t for which it took to compute

! the matrix elimination

!

! Preconditions: The input (for matrix size) must be a

! positive integer

!

! Postconditions: None.

!

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

program gauss

IMPLICIT NONE

REAL:: start, finish !time vars

INTEGER::i,j,n !loop vars

REAL::s

REAL,allocatable,DIMENSION(:,:)::a !Allocating space

REAL,allocatable,DIMENSION(:)::x !Allocating space

PRINT '("Enter a positive integer n for matrix size: ")'

READ \*, n !Reading in matrix size

allocate(a(n,n+1)) !Allocating space

allocate(x(n)) !Allocating space

CALL RANDOM\_NUMBER(a) !Filling matrix

CALL RANDOM\_NUMBER(x) !Filling matrix

CALL cpu\_time(start) !Starting timer

do j=1,n !Calculations

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 !Calculation part 2

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) !stopping timer

print '("Time to run is ",f6.3," seconds.")',finish-start !printing formatted time

end program gauss

**Tables:**

All times are in milliseconds.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fortran** | **250** | **500** | **1000** | **1500** | **2000** |
|  | 16.00 | 154.00 | 1,266.00 | 7,026.00 | 16,719.00 |
|  | 16.00 | 154.00 | 1,276.00 | 7,040.00 | 16,680.00 |
|  | 15.00 | 154.00 | 1,272.00 | 7,033.00 | 12,632.00 |
|  | 16.00 | 154.00 | 1,269.00 | 7,070.00 | 11,808.00 |
|  | 16.00 | 154.00 | 1,265.00 | 7,101.00 | 16,793.00 |
| **AVG** | 15.80 | 154.00 | 1,269.60 | 7,054.00 | 14,926.40 |
| **STDEV** | 0.45 | - | 4.51 | 31.17 | 2,488.04 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Python No Numpy** | **250** | **500** | **1000** | **1500** | **2000** |
|  | 488.55 | 3,928.63 | 34,122.69 | 194,762.08 | 663,206.39 |
|  | 489.17 | 3,863.75 | 37,016.80 | 205,631.85 | 481,076.08 |
|  | 487.78 | 3,918.82 | 38,749.51 | 216,648.38 | 517,794.55 |
|  | 486.95 | 3,942.84 | 39,152.58 | 215,641.46 | 522,325.99 |
|  | 494.98 | 3,979.33 | 37,893.88 | 215,271.96 | 669,401.51 |
| **AVG** | 489.49 | 3,926.67 | 37,387.09 | 209,591.14 | 570,760.90 |
| **STDEV** | 3.18 | 42.01 | 2,001.46 | 9,410.89 | 88,700.17 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Python With Numpy** | **250** | **500** | **1000** | **1500** | **2000** |
|  | 6.20 | 41.16 | 248.51 | 819.87 | 1,904.80 |
|  | 6.30 | 41.37 | 252.08 | 845.85 | 1,920.52 |
|  | 7.97 | 41.26 | 252.38 | 813.08 | 1,913.84 |
|  | 6.68 | 42.72 | 250.01 | 814.51 | 1,913.90 |
|  | 7.12 | 44.34 | 252.58 | 811.66 | 1,941.14 |
| **AVG** | 6.85 | 42.17 | 251.11 | 820.99 | 1,918.84 |
| **STDEV** | 0.72 | 1.37 | 1.78 | 14.24 | 13.66 |

**Comparison Graph:**

**Comparison and Summary:**

* When looking at the data, and especially the data represented in a 3D bar graph, it is clear which implementations run faster than the others. After a quick analysis, it is very clear that the fastest implementation of the Gaussian Elimination is using Python and NUMPY. While the slowest was Python without NUMPY. Fortran came in at a close second after Python with NUMPY, with promising results. In conclusion, when implementing a Gaussian Elimination, it is much wiser and easier to use Python with the NUMPY module imported.