Fortran

250					
	0.049u	0.000s	0:00.05	80.0%	0+0k 0+0io 0pf+0w
500					
	0.333u	0.000s	0:00.33	100.0%	0+0k 0+0io 0pf+0w
1000					
	2.681u	0.000s	0:02.68	100.0%	0+0k 0+0io 0pf+0w
1500					
	8.883u	0.000s	0:08.88	100.0%	0+0k 0+0io 0pf+0w
2000					
	21.350u	0.000s	0:21.35	100.0%	0+0k 8+0io 0pf+0w

Python NumPy

250					
	6.703u	0.183s	0:06.65	103.4%	0+0k 0+0io 0pf+0w
500					
	52.371u	0.164s	0:52.30	100.4%	0+0k 0+0io 0pf+0w
1000					
	403.263u	0.187s	6:43.21	100.0%	0+0k 0+0io 0pf+0w
1500					
	1451.622u	0.248s	24:11.74	100.0%	0+0k 0+0io 0pf+0w
2000					

```
CS471/p3> gfortran Gaussian1.f
CS471/p3> time ./a.out
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO
0.049u 0.000s 0:00.05 80.0%     0+0k 0+0io 0pf+0w
```

```
CS471/p3> gfortran Gaussian1.f
CS471/p3> time ./a.out
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO
0.333u 0.000s 0:00.33 100.0%    0+0k 0+0io 0pf+0w
```

```
CS471/p3> gfortran Gaussian1.f
CS471/p3> time ./a.out
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO
2.681u 0.000s 0:02.68 100.0%    0+0k 0+0io 0pf+0w
```

```
CS471/p3> gfortran Gaussian1.f
CS471/p3> time ./a.out
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO
8.883u 0.000s 0:08.88 100.0%    0+0k 0+0io 0pf+0w
```

```
CS471/p3> gfortran Gaussian1.f
CS471/p3> time ./a.out
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_DIVIDE_BY_ZERO
21.350u 0.000s 0:21.35 100.0%   0+0k 8+0io 0pf+0w
```

```
CS471/p3> time python Gaussian.py
6.703u 0.183s 0:06.65 103.4%    0+0k 0+0io 0pf+0w
```

```
CS471/p3> time python Gaussian.py
52.371u 0.164s 0:52.30 100.4%   0+0k 0+0io 0pf+0w
```

```
CS471/p3> time python Gaussian.py
403.263u 0.187s 6:43.21 100.0%  0+0k 0+0io 0pf+0w
```

```
CS471/p3> time python Gaussian.py
1451.622u 0.248s 24:11.74 100.0%        0+0k 0+0io 0pf+0w
```

```
CS471/p3> time python Gaussian.py
2572.799u 0.275s 42:52.89 100.0%        0+0k 0+0io 0pf+0w
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
!       Andrew Baca
!       Semptember 14, 2018
!
!       Program: Gaussian1.f
!       Objective: this program will perform Gausian elimination on a square matrix given the input size
!                  and capture UNIX time of these runs.
!
!       Input: Array size 250, 500, 1000, 1500, 2000
!       Output: Solved Array and UNIX runtime for the solve
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        program Gaussian1

        implicit none
        integer i,N,j,k,siz,s
        real, dimension(:), allocatable :: B                    !declare 1-d array for agmented part of matrix
        real, dimension(:,:), allocatable :: A                  !declare 2-d array for system of equations
        real, dimension(:), allocatable :: X                    !declare 1-d array for variable names
        real z
        ALLOCATE(A(N,N))                        !allocate space for array
        AllOCATE(B(N))                          !allocate space for the array
        Allocate(X(N))                          !allcoate space for the array

        z=0
        siz = 500

        do i = 1,siz                            !fill the system of equations with random variables ranging from 1 to 10
         do j=1, siz
           A(i,j) = int(rand(0)*10) + 1
           !print *,A(i,j)
          end do
        end do

        do i = 1, siz - 1                               !Gaussian elimination for the bottom half of the matrix
         do j = i, siz
           A(j,i) = A(j,i)/A(i,i)
            do k = i + 1, siz
             A(j,k) =  A(j,k) - (A(j,i)*A(i,k))
            end do
          end do
        end do
```

```
do i = 1, siz - 1                               !Gaussian elimination for the top haf of the matrix
 do j = i + 1, siz
   A(j,i) = A(j,i)/A(i,i)
    do k = i, siz
     A(j,k) =  A(j,k) - (A(j,i)*A(i,k))
    end do
  end do
end do

do i = 1, siz                                   !fill answer array and variable array with 1's
 b(i) = 1.0
 x(i) = 1.0
end do

do i = 1, siz -1                                !forward elimination using the augmented part of the matrix
 do j = i + 1, siz
  B(j) = B(j) - (A(j,i)*B(i))
 end do
end do

x(siz) = A(siz,siz)

do i = siz, 1                                   !fill in variableswith their answers, ie backward solve
 s = b(i)
 do j = i + 1, siz
  s = s - (A(i,j)*x(j))
 end do
 x(i) = s/a(i,i)
end do

stop
end
```

```
###################################################################################################################
#
# Andrew Baca
# September 14, 2018
#
# Program: Gaussian.py
# Objective: this program will perform Gausian elimination on a square matrix given the input size
#            and capture UNIX time of these runs.
#
# Input: Array size 250, 500, 1000, 1500, 2000
# Output: Solved Array and UNIX runtime for the solve
#
##################################################################################################################

import numpy as np                              #import numPy Libraries
from random import seed                         #import RandomLibraries
import random

seed(1)                         #random seed

size = 1000                     #size entry for the matrix

b = np.ones(size)                       #declare 1-d array for augmented part of matrix
x = np.ones(size)                       #declare 1-d array for variable in Ax = B
a = np.random.randn(size, size)* 10     #include 2d array representing the system of equations


for i in range(size - 1):               #gaussian elimination, echelon form
 for j in range(i , size):
  a[j,i] = a[j,i]/a[i,i]
  for k in range(i + 1, size):
    a[j,k] = a[j,k] - a[j,i]*a[i,k]

for i in range(size - 1):               #gausian elimination reducedechelon form
 for j in range(i + 1, size):
  a[j,i] = a[j,i]/a[i,i]
  for k in range(i, size):
    a[j,k] = a[j,k] - a[j,i]*a[i,k]

for i in range(size - 1):               #forward elimination using the augmented part of the matrix
 for j in range(i + 1, size):
  b[j] = b[j] - a[j,i] * b[i]

for i in range(size -1, 0):             #backward solve into variable part of the matrix
 s = b[i]
 for j in range(i + 1, size):
  s = s - a[i,j]*x[j]
 x[i] = s / a[i,i]
```