Jack Rellamas
Dr. Saleem
MATH 143M, T/TH
25 September 2023

Project 1: Gaussian Elimination

Results for n = 11

- Exact Solution: transpose of
  - $[1.0, 1.0, \ldots, 1.0_{11}]$
- Computed Solution: transpose of
  - [-6.1039, 57.4389, -148.5615, 274.3009, -320.1991, -73.1111, 305.9167, -14.5, -24.3333, -226. , 208.5]
- My Error: transpose of
  - [7.1039, -56.4389, 149.5615, -273.3009, 321.1991, 74.1111, -304.9167, 15.5, 25.3333, 227. , -207.5]
- Infinity Norm: 320.1991
- Euclidean Norm: 630.4414
- Number of Multiplications: 297
- Number of Multiplications for n = 11 according to formula (where the formula is given by $num\ multiplications = n^3/3 + n^2 - n/3$): 561

Results for n = 12

- Exact Solution: transpose of
  - $[1.0, 1.0, \ldots, 1.0_{12}]$
- Computed Solution: transpose of
  - [-3.200000e-03, -2.068050e+01, 5.441310e+01, 8.200930e+01, -1.574907e+02, 1.043889e+02, -1.378333e+02, -1.920000e+02, -2.433330e+01, 1.290000e+02, 3.860000e+02, -1.775000e+02]
- My Error: transpose of
  - [1.0032, 21.6805, -53.4131, -81.0093, 158.4907, -103.3889, 138.8333,
  - 193. , 25.3333, -128. , -385. , 178.5]
- Infinity Norm: 386.0
- Euclidean Norm: 547.1853
- Number of Multiplications: 354

- Number of Multiplications for n = 12 according to formula (where the formula is given by $num\ multiplications = n^3/3 + n^2 - n/3$): 716

Results for n = 13

- Exact Solution: transpose of
  - $[1.0, 1.0, \ldots, 1.0_{13}]$
- Computed Solution: transpose of
  - [1.4116, -30.5978, 78.418, 51.0123, -148.1543, 32.6852, -90.7778, 9.6667, 2.5556, -32.3333, 184.3333, -96.8333, 80.6667]
- My Error: transpose of
  - [-0.4116, 31.5978, -77.418, -50.0123, 149.1543, -31.6852, 91.7778, -8.6667, -1.5556, 33.3333, -183.3333, 97.8333, -79.6667]
- Infinity Norm: 184.3333
- Euclidean Norm: 303.2433
- Number of Multiplications: 416
- Number of Multiplications for n = 13 according to formula (where the formula is given by $num\ multiplications = n^3/3 + n^2 - n/3$): 897

Sources of Error:

The main source of error in this program would have to be rounding to the 4 decimal spot. Another source of error would be an optimized algorithm. If we also included partial pivoting in our program, we would have a smaller error at the end of the computation. Additionally, we chose to do our computation on the Hilbert Matrix, which we know is notorious for being an ill-conditioned matrix.

Computer Program: (done in python)

```
import numpy as np
import math

n = 13     # size of hilbert matrix
num_multiplications = 0
```

```
# Create the Hilbert matrix
hilbert_matrix = np.zeros((n, n))
for i in range(1, n + 1):
      for j in range(1, n + 1):
      hilbert_matrix[i - 1, j - 1] = 1.0 / (i + j - 1)
      num_multiplications += 1

# augment matrix with all 1's
augmented_matrix = np.column_stack((hilbert_matrix, np.ones(n)))
augmented_matrix = np.round(augmented_matrix, 4) # keep 4 digit
rounding property

# Perform Gaussian elimination
for i in range(n):
      # Eliminate entries below the current pivot
      for j in range(i + 1, n):
      pivot = augmented_matrix[i, i]
      multiplier = augmented_matrix[j, i] / pivot
      augmented_matrix[j, i:] -= multiplier * augmented_matrix[i, i:]
      num_multiplications += 2

      # Round the values in the augmented matrix
      augmented_matrix = np.round(augmented_matrix, 4)


# Backward substitution
x = np.zeros(n)  # initialize solution vector x
# loop from n-1 -> 0, decremeting by 1 every step
for i in range(n - 1, -1, -1):
      sum = 0
      for j in range(n, i, -1):
      if (j != n):
            sum -= augmented_matrix[i, j] * x[j]
            num_multiplications += 1
      # first loop through special case (augmented column)
      else:
            sum += augmented_matrix[i, j]
      # whatshere = augmented_matrix[i,i]
```

```
      x[i] = sum / augmented_matrix[i, i]
      num_multiplications += 1
# Round the values in the x vector
x = np.round(x, 4)

# Error Vector
error_vector = np.subtract(np.ones(n), x)
error_vector = np.round(error_vector, 4)

# Euclidean Norm
euclidean_norm = 0
for i in x:
      euclidean_norm += i * i
euclidean_norm = math.sqrt(euclidean_norm)
euclidean_norm = round(euclidean_norm, 4)

# Infinity Norm
absolute_X = abs(x)
infinity_norm = np.max(absolute_X)

# Print DATA
print("Solution Vector:")
print(x)
print("Error Vector:")
print(error_vector)
print(f"Number of Multiplications: {num_multiplications}")
print(f"Euclidean Norm: {euclidean_norm}")
print(f"Infinity Norm: {infinity_norm}")
```