CS 3200

Assignment 03

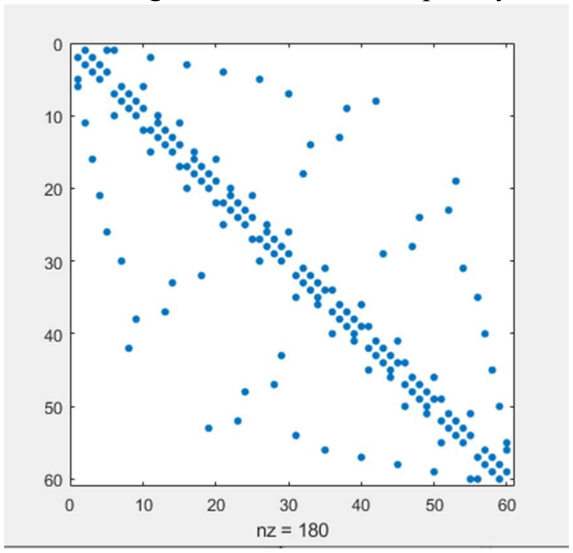
Lianrui Geng

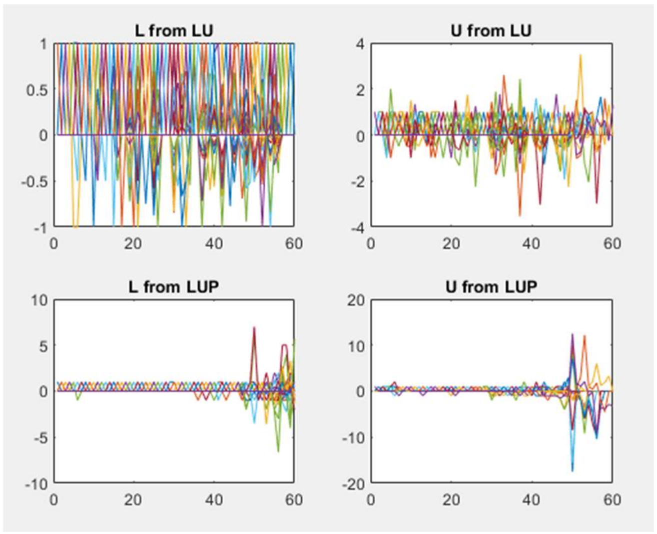
U1346008

Q1:

b)

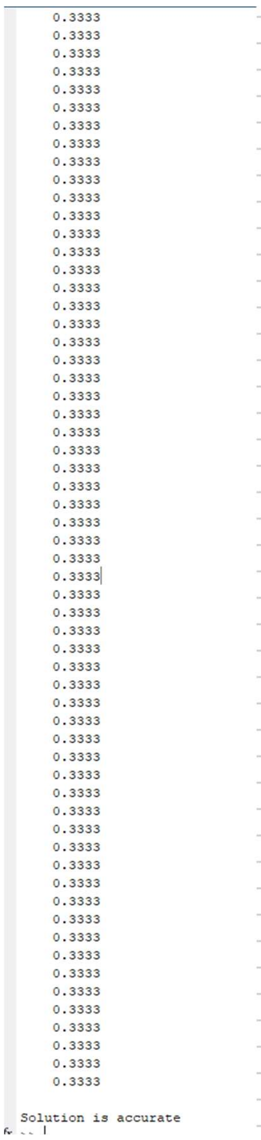
The matrix B is a sparse matrix, which means most of its entries are 0. Advantages of storing a sparse matrix in compressed format: reduce the memory use, more efficient.
Disadvantages: increase the complexity and limited usefulness.



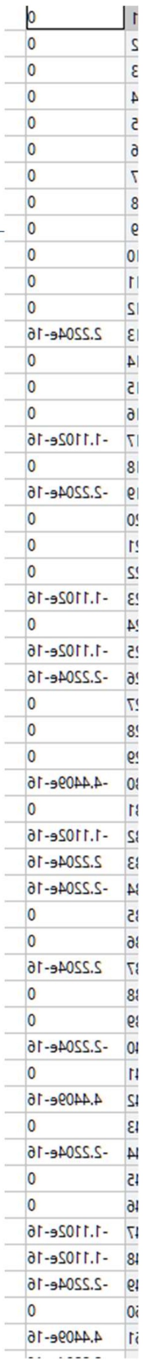


e)

LU decomposition involves implicit permutation matrix P while LUP decomposition involves explicit permutation matrix P. LUP decomposition provides additional flexibility and numerical stability, while LU decomposition is generally simpler to implement.



```
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333
    0.3333

Solution is accurate
>>
```



```
0
0
0
0
0
0
0
0
0
0
0
0
2.2204e-16
0
0
0
-1.1102e-16
0
-2.2204e-16
0
0
0
-1.1102e-16
0
-1.1102e-16
-2.2204e-16
0
0
0
-4.4409e-16
0
-1.1102e-16
2.2204e-16
-2.2204e-16
0
0
2.2204e-16
0
0
-2.2204e-16
0
4.4409e-16
0
-2.2204e-16
0
0
-1.1102e-16
-1.1102e-16
-2.2204e-16
0
4.4409e-16
```

g)

I don’t think I need to do iterative refinement, because its result is really close to my expectation. As we know, iterative refinement is designed to improve accuracy. In this case, I don’t think we need to do this.

Q2:

This is jacobi's method:

```
   1.0e+92 *


   -0.8629

   -1.4447

   -1.9043

   -2.1426

   -2.1426

   -1.9043

   -1.4447

   -0.8629
```

Itr of jacobi's method: 100

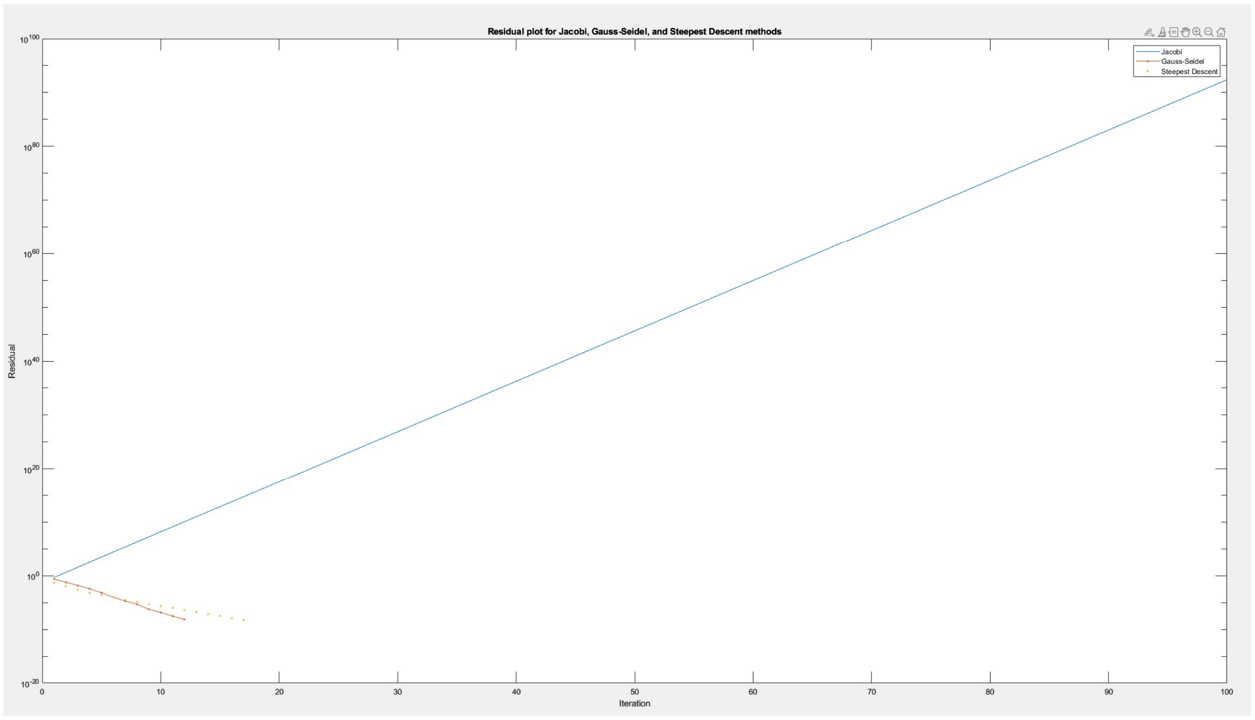
This is normVal of jacobi's method:

```
   1.0e+92 *


  Columns 1 through 12


    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000


  Columns 13 through 24


    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000
```

```
  Columns 25 through 36

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000

  Columns 37 through 48

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000

  Columns 49 through 60

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000

  Columns 61 through 72

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000

  Columns 73 through 84

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000

  Columns 85 through 96

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
0.0001    0.0004
```

```
  Columns 97 through 100


    0.0037    0.0322    0.2776    2.3913


This is gauss_seidel's method:
    0.0667
    0.0504
    0.0484
    0.0504
    0.0504
    0.0484
    0.0504
    0.0667


Itr of gauss_seidel's method: 11
This is normVal of gauss_seidel's method:
    0.2513    0.0628    0.0155    0.0038    0.0007    0.0001    0.0000    0.0000    0.0000    0.0000
0.0000    0.0000


Jacobi solution: -
86290108262385574531044021377323057541429777312472870310738408034936835379916155061150941184.000000 -
14446696107754237457449981822248958104289252237566229059171071526457340281126803403013567283 2.000000 -
19043109077409172417083822662136536141010406977019779860591678729313746597885583527395511500 8.000000 -
21425634073227828404315916176663686103546609572037698990982243668132177151449437011205056102 4.000000 -
21425634073227828404315916176663686103546609572037698990982243668132177151449437011205056102 4.000000 -
19043109077409172417083822662136536141010406977019779860591678729313746597885583527395511500 8.000000 -
```

```
1444669610775423745744998182224895810428925223756622905917107152645734028112
68034030135672832.000000 -
8629010826238557453104402137732305754142977731247287031073840803493683537991
6155061150941184.000000

Gauss-Seidel solution: 0.066712 0.050359 0.048382 0.050408 0.050408 0.048382 0.050359
0.066712

steepest_descent solution: 0.066712 0.050359 0.048382 0.050408 0.050408 0.048382
0.050359 0.066712

>>
```



c)

The Jacobi algorithm updates all components of the solution vector simultaneously, while the Gauss-Seidel algorithm updates each component sequentially. The Gauss-Seidel algorithm typically converges faster but requires less memory and can be easier to implement in parallel.

d)

A sparse matrix has most of its entries as zero. Gauss-Seidel and Jacobi algorithms are useful for sparse matrices because they only process the nonzero entries. This makes them much more efficient and allows for parallelization, making them well-suited for scientific and engineering applications that involve large sparse matrices.

Q3:

Example:

Run one time:

Iteration	x (1)	x (2)	d (1)	d (2)	alpha
1	0.423	0.368	-204.289	234.772	0.001
2	1.084	-0.392	58.764	51.134	-0.003
3	1.140	-0.343	-27.346	31.426	0.001
4	1.229	-0.445	7.866	6.845	-0.003
5	1.236	-0.438	-3.660	4.207	0.001
6	1.248	-0.452	1.053	0.916	-0.003
7	1.249	-0.451	-0.490	0.563	0.001
8	1.251	-0.453	0.141	0.123	-0.003
9	1.251	-0.452	-0.066	0.075	0.001
10	1.251	-0.453	0.019	0.016	-0.003
11	1.251	-0.453	-0.009	0.010	0.001

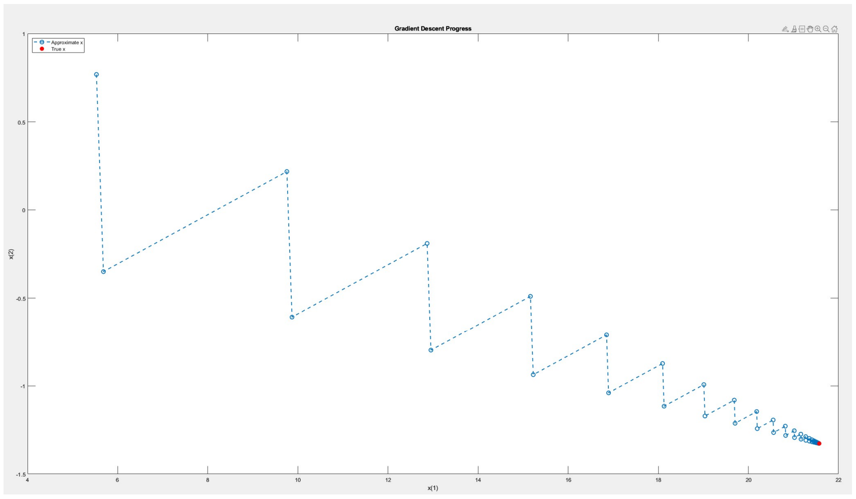
Solution of the system is:

1.251046   -0.452648

residual 0.000018 0.000016

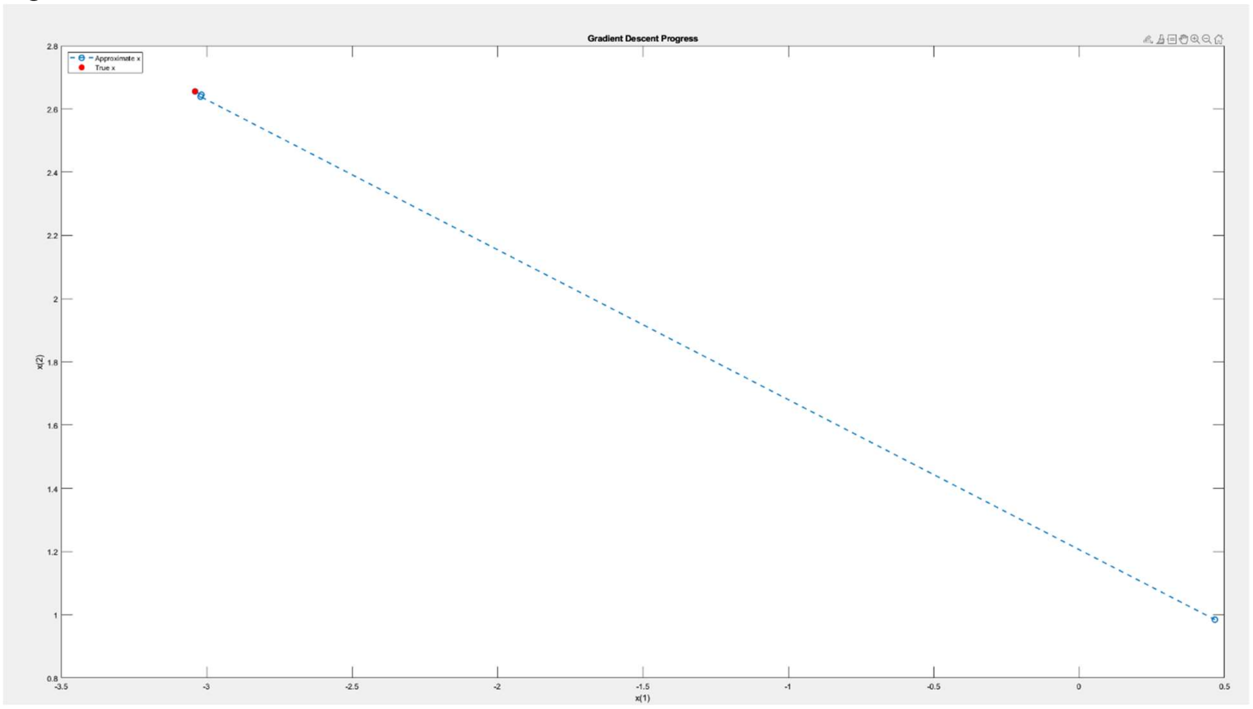
in 11 iterations

True x value: 1.251081 -0.452683



e)

I don’t believe this graph always shows converge. Sometimes, it won’t converge, and it seems like the line is far away from true value. The plot of approximate x values may exhibit plateauing, diverging, or slow convergence. Plateauing appears as a horizontal or wavy portion, divergence as a diagonal or curved portion, and slow convergence as a very gradual slope or irregular fluctuations. Factors such as the condition number of the matrix, step size, and choice of direction vector can all influence the convergence rate and accuracy of the algorithm.



f)

The learning rate parameter (alpha) determines the step size of each iteration of the optimization algorithm, such as gradient descent. It is important to choose an appropriate value for α because it affects the convergence rate of the optimization algorithm and the final solution.

When α is too small, the optimization algorithm will take a long time to converge to the minimum, and the algorithm may get stuck in the local minimum instead of reaching the global minimum. This is because the algorithm takes very small steps and may not make significant progress towards the minimum.

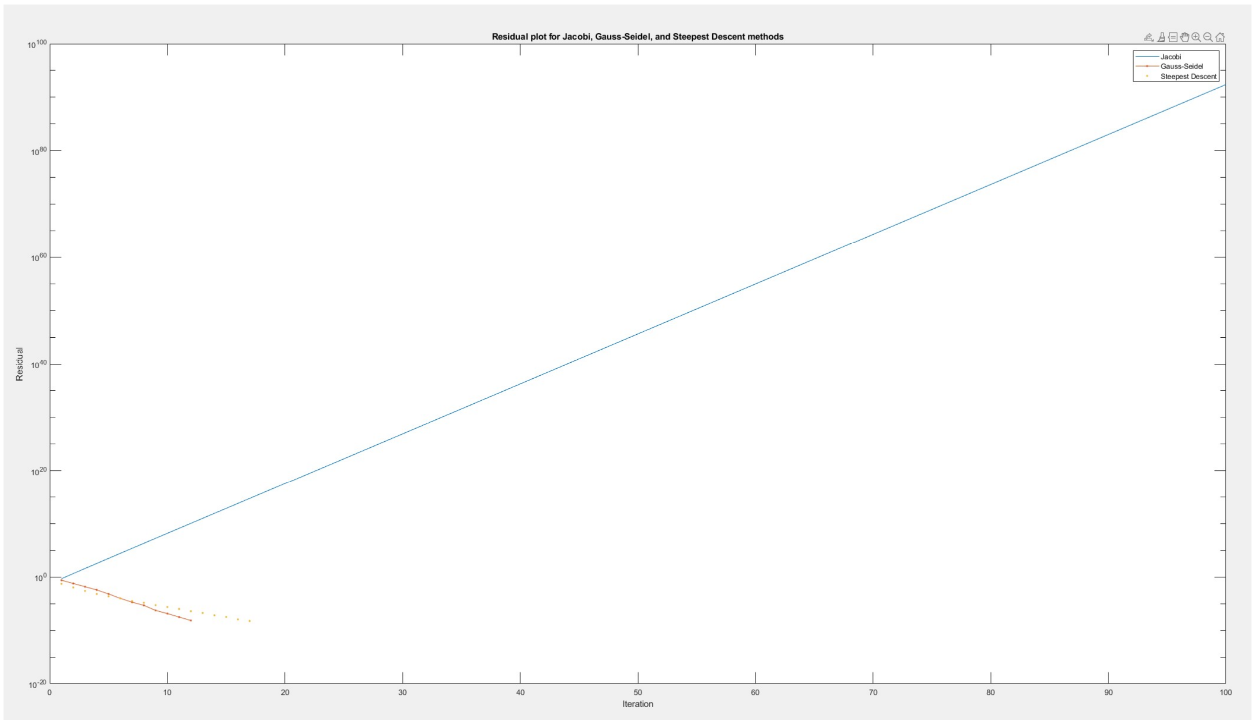
On the other hand, when the value of α is too large, the optimization algorithm may overfulfill the minimum and diverge. This is because the algorithm takes too many steps and can oscillate or diverge instead of converging towards a minimum.

The alpha formula commonly used in optimization algorithms is a decreasing function of the number of iterations. This means that as the algorithm approaches the minimum, the step size is reduced, which helps prevent overshot and oscillation. Conversely, a constant value of α may not adapt to changing conditions of the optimization problem and may lead to suboptimal solutions.

In conclusion, choosing an appropriate value for α is critical to the success of an optimization algorithm in machine learning. Choosing a small value will make the algorithm too slow, while choosing a large value will make the algorithm unstable. Using a decreasing alpha formula can help balance these two extremes and ensure convergence to an optimal solution.

Q4:

Remind: the code is in Q2 code, I put three functions in one file.



c)

From the figure we can see that the Gauss-Seidel method and the steepest descent method converge faster than the Jacobian method. This is because the Jacobian method updates each component of the solution vector independently, which leads to slower convergence compared to the other two methods that update the solution vector more efficiently.

We can also see that the Gauss-Seidel method converges faster than the steep descent method, at least for a given system of linear equations. This is because the Gauss-Seidel method uses more information about new components of the solution vector at each update, which leads to faster convergence than the steepest descent method, which only uses gradient information.

Furthermore, the steepest descent method requires more iterations to converge than the Gauss-Seidel method, and the convergence behavior is more volatile. This is because the steepest descent method chooses the search direction based on the gradient, which can lead to slow progress towards the minimum if the gradient is not a good indicator of the direction of the steepest descent.