1(a) Implement gaussian elimination

1. (a) Using only 3 significant bits

$$\left[\begin{array}{ccc|c} 2.51 & 1.48 & 4.53 & 0.05 \\ 1.48 & 0.93 & -1.30 & 1.03 \\ 2.68 & 3.04 & -1.48 & -0.53 \end{array}\right]$$

$$\Rightarrow \left[\begin{array}{ccc|c} 2.51 & 1.48 & 4.53 & 0.0500 \\ 0 & 0.0574 & -3.97 & 1.00 \\ 0 & 1.46 & -6.32 & -26.1 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 2.51 & 1.48 & 4.53 & 0.0500 \\ 0 & 0.0574 & -3.97 & 1.00 \\ 0 & 0 & 94.8 & -26.1 \end{array}\right]$$

$X = 1.46$ , $Y = -1.60$, $Z = -0.275$

(b) Do partial pivoting

(b) Chance 1,3 row first

$$\left[\begin{array}{ccc|c} 2.68 & 3.04 & -1.48 & -0.53 \\ 1.48 & 0.93 & -1.30 & 1.03 \\ 2.51 & 1.48 & 4.53 & 0.05 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 2.68 & 3.04 & -1.48 & -0.53 \\ 0 & -0.749 & -0.483 & 1.32 \\ 0 & -1.37 & 5.92 & 0.546 \end{array}\right]$$

$\Rightarrow$ Change 2,3 row $\Rightarrow$ $\left[\begin{array}{ccc|c} 2.68 & 3.04 & -1.48 & -0.53 \\ 0 & -1.37 & 5.92 & 0.546 \\ 0 & -0.749 & -0.483 & 1.32 \end{array}\right]$

$$= \left[\begin{array}{ccc|c} 2.68 & 3.04 & -1.48 & -0.530 \\ 0 & -1.37 & 5.92 & 0.546 \\ 0 & 0 & -3.72 & 1.02 \end{array}\right]$$

$X = 1.44$ , $Y = -1.58$ , $Z = -0.274$

(c)Chop the number rather than rounding

(C) Chop the number

From(b), we know the exchange rule

$$\xrightarrow[2 \leftrightarrow 3]{1 \leftrightarrow 3} \left[\begin{array}{ccc|c} 2.68 & 3.04 & -1.48 & -0.53 \\ 2.51 & 1.48 & 4.53 & 0.05 \\ 1.48 & 0.93 & -1.30 & 1.03 \end{array}\right] \Rightarrow \left[\begin{array}{ccc|c} 2.68 & 3.04 & -1.48 & -0.53 \\ 0 & -1.36 & 5.91 & 0.546 \\ 0 & -0.748 & -0.482 & 1.32 \end{array}\right]$$

$$\Rightarrow \left[\begin{array}{ccc|c} 2.68 & 3.04 & -1.48 & -0.53 \\ 0 & -1.36 & 5.91 & 0.546 \\ 0 & 0 & -3.73 & 1.01 \end{array}\right] \quad X = 1.43 \quad Y = -1.57 \quad Z = -0.270$$

(d) The result show that the Gaussian elimination best matches the original right-hand sides. Surprisingly, using partial pivoting does not do better. I think it may be because we only keep 3 significant numbers, which lead to the result. Also, part(c) match the right-hand side worse, since we always discard all the number after three significant numbers, which may lead to bigger error.

(d) The table show the error value of each method for each 3 equation.

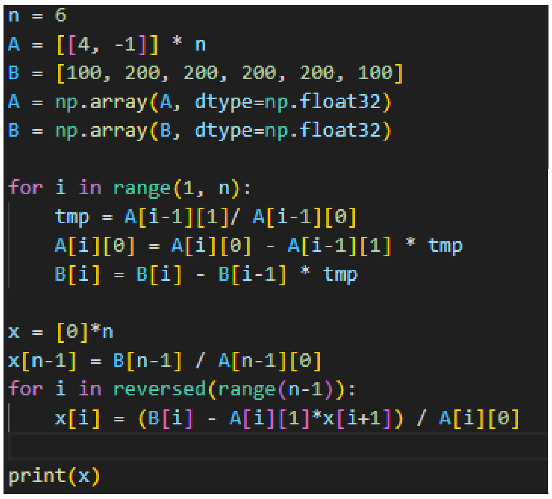
| Method \ Equation | 1st | 2nd | 3rd | Total error |
|---|---|---|---|---|
| (a) | 0.00085 | 0.0003 | 0.0142 | 0.01535 |
| (b) | 0.01522 | 0.012 | 0.00848 | 0.0357 |
| (C) | 0.0074 | 0.0227 | 0.0108 | 0.0409 |

2(a)

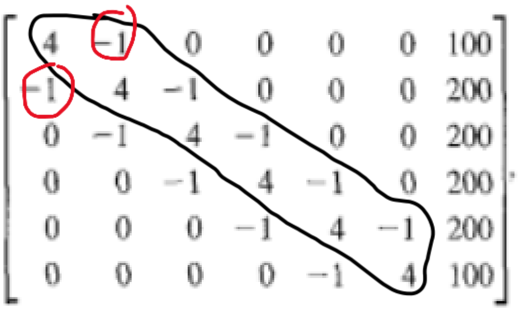
Since the matrix is symmetric, we can only use three columns for each equation, one for b and the other two for A. To solve this equation, we just need to use simple Gaussian elimination. For each row, we need to count the ratio A[i+1][i] / A[i][i] for gaussian elimination to eliminate the (-1) and update b[i+1], A[i+1][i+1]. The information we need for this process is enough to store in an n*3 matrix.

After finishing gaussian elimination, we use back substitution to get all the answers.

(b) Implementation



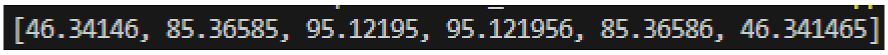
```
n = 6
A = [[4, -1]] * n
B = [100, 200, 200, 200, 200, 100]
A = np.array(A, dtype=np.float32)
B = np.array(B, dtype=np.float32)

for i in range(1, n):
    tmp = A[i-1][1]/ A[i-1][0]
    A[i][0] = A[i][0] - A[i-1][1] * tmp
    B[i] = B[i] - B[i-1] * tmp

x = [0]*n
x[n-1] = B[n-1] / A[n-1][0]
for i in reversed(range(n-1)):
    x[i] = (B[i] - A[i][1]*x[i+1]) / A[i][0]

print(x)
```



$$\left[\begin{array}{cccccc|c} 4 & -1 & 0 & 0 & 0 & 0 & 100 \\ -1 & 4 & -1 & 0 & 0 & 0 & 200 \\ 0 & -1 & 4 & -1 & 0 & 0 & 200 \\ 0 & 0 & -1 & 4 & -1 & 0 & 200 \\ 0 & 0 & 0 & -1 & 4 & -1 & 200 \\ 0 & 0 & 0 & 0 & -1 & 4 & 100 \end{array}\right]$$

Matrix A stores all the entries I circle in the above picture. Since the matric is symmetric, the value in constructed matrix A[0][1] store both the value in red circle. A[1~n-1][1] store the value in the same way. A[0~n-1][0] store all the diagonal value. In this way, we can store all the matrix in this form into n*3 matrix without losing the information.

We can use the compact form matric to carefully process the gaussian elimination and use back substitution for all the answers. Below is the result.



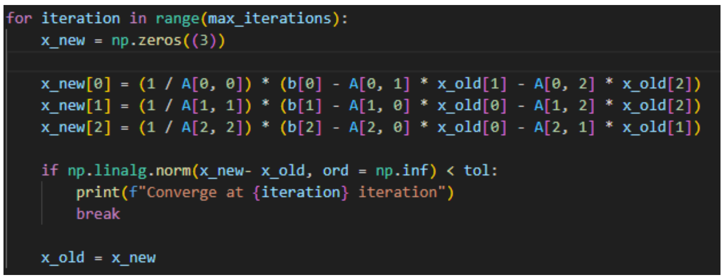
```
[46.34146, 85.36585, 95.12195, 95.121956, 85.36586, 46.341465]
```

(c)

For each row (0 ~ n-2), when implementing Gaussian elimination, we need to compute the ratio for next row, then one value in A and one value in B need to multiply that ratio and minus to the next row. Therefore, we need 5(n-1) for this step. For each row(0~n-1), when implementing back substitution, we need to multiply one entry in A and minus it to B and finally do a division. Therefore, we need another 3*n for this step, but the bottom row doesn't need multiplication and subtraction.

Therefore, total arithmetic operations = 5n-5+3*n-2 = 8n-7

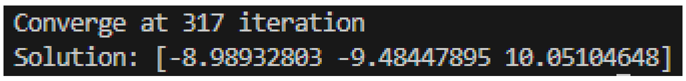
3(a)



```
for iteration in range(max_iterations):
    x_new = np.zeros((3))

    x_new[0] = (1 / A[0, 0]) * (b[0] - A[0, 1] * x_old[1] - A[0, 2] * x_old[2])
    x_new[1] = (1 / A[1, 1]) * (b[1] - A[1, 0] * x_old[0] - A[1, 2] * x_old[2])
    x_new[2] = (1 / A[2, 2]) * (b[2] - A[2, 0] * x_old[0] - A[2, 1] * x_old[1])

    if np.linalg.norm(x_new- x_old, ord = np.inf) < tol:
        print(f"Converge at {iteration} iteration")
        break

    x_old = x_new
```

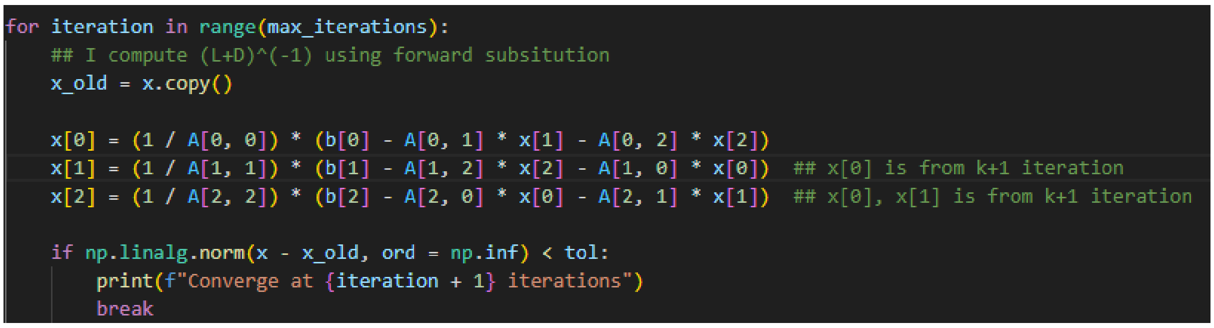
$$x_i^{(k+1)} = (b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)}) / a_{ii}, \quad i = 1, \ldots, n$$

Using the equation from lecture slide to obtain new value for x at each iteration. When the max value from x_new – x_old < tol, I view it as converge and get the result x.

Result: 

```
Converge at 317 iteration
Solution: [-8.98932803 -9.48447895 10.05104648]
```

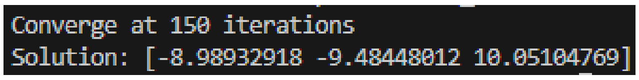
(b)



```
for iteration in range(max_iterations):
    ## I compute (L+D)^(-1) using forward subsitution
    x_old = x.copy()

    x[0] = (1 / A[0, 0]) * (b[0] - A[0, 1] * x[1] - A[0, 2] * x[2])
    x[1] = (1 / A[1, 1]) * (b[1] - A[1, 2] * x[2] - A[1, 0] * x[0])  ## x[0] is from k+1 iteration
    x[2] = (1 / A[2, 2]) * (b[2] - A[2, 0] * x[0] - A[2, 1] * x[1])  ## x[0], x[1] is from k+1 iteration

    if np.linalg.norm(x - x_old, ord = np.inf) < tol:
        print(f"Converge at {iteration + 1} iterations")
        break
```

$$x_i^{(k+1)} = \frac{1}{a_{ii}}(b_i - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)})$$

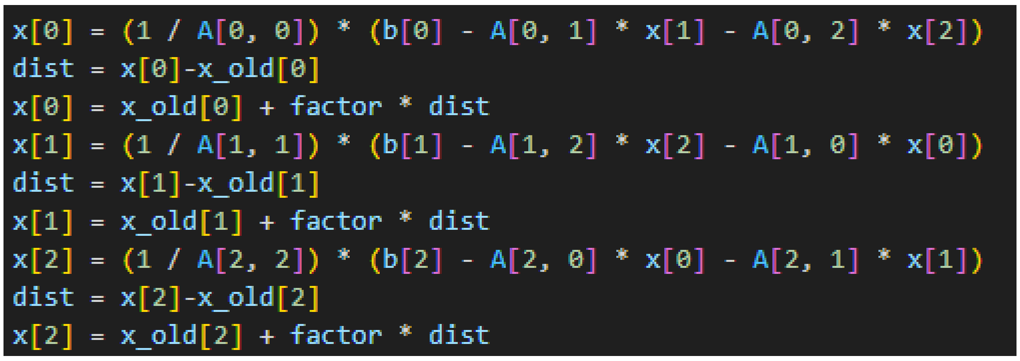
Gauss-Seidel is similar to Jacobi method, but we use the new x value as soon as possible. I use the above equation from lecture slide to compute new x value. This equation is obtained from the below equation. The first summation is b minus the result from Ux and the second summation is solving the forward substitution of (L+D). We can get the above equation. The converge condition is same as Jacobi method. The converge iteration times is less than Jacobi method.

Result: 

```
Converge at 150 iterations
Solution: [-8.98932918 -9.48448012 10.05104769]
```

$$(\mathbf{L}+\mathbf{D})\mathbf{x}^{(k+1)} = (\mathbf{b} - \mathbf{U}\mathbf{x}^{(k)})$$

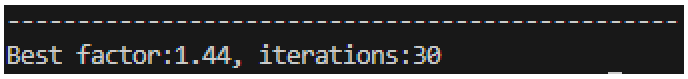
4



```
x[0] = (1 / A[0, 0]) * (b[0] - A[0, 1] * x[1] - A[0, 2] * x[2])
dist = x[0]-x_old[0]
x[0] = x_old[0] + factor * dist
x[1] = (1 / A[1, 1]) * (b[1] - A[1, 2] * x[2] - A[1, 0] * x[0])
dist = x[1]-x_old[1]
x[1] = x_old[1] + factor * dist
x[2] = (1 / A[2, 2]) * (b[2] - A[2, 0] * x[0] - A[2, 1] * x[1])
dist = x[2]-x_old[2]
x[2] = x_old[2] + factor * dist
```

$$x_i^{(k+1)} = x_i^{(k)} + \frac{w}{a_{ii}}(b_i - \sum_{j=i}^{n} a_{ij}x_j^{(k)} - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)})$$

Every time we update a value; we multiply a factor in order to get a faster convergence. I iterate all the possible factor from 1~2(gap : 0.01), the best result is

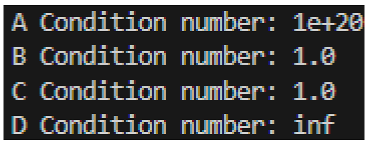


```
---------------------------------------------
Best factor:1.44, iterations:30
```

5.

I use infinity norm condition number to computer the 4 matrices' condition number, here's the result:

(I use function np.linalg.norm, np.linalg.inv since matlab have similar function)

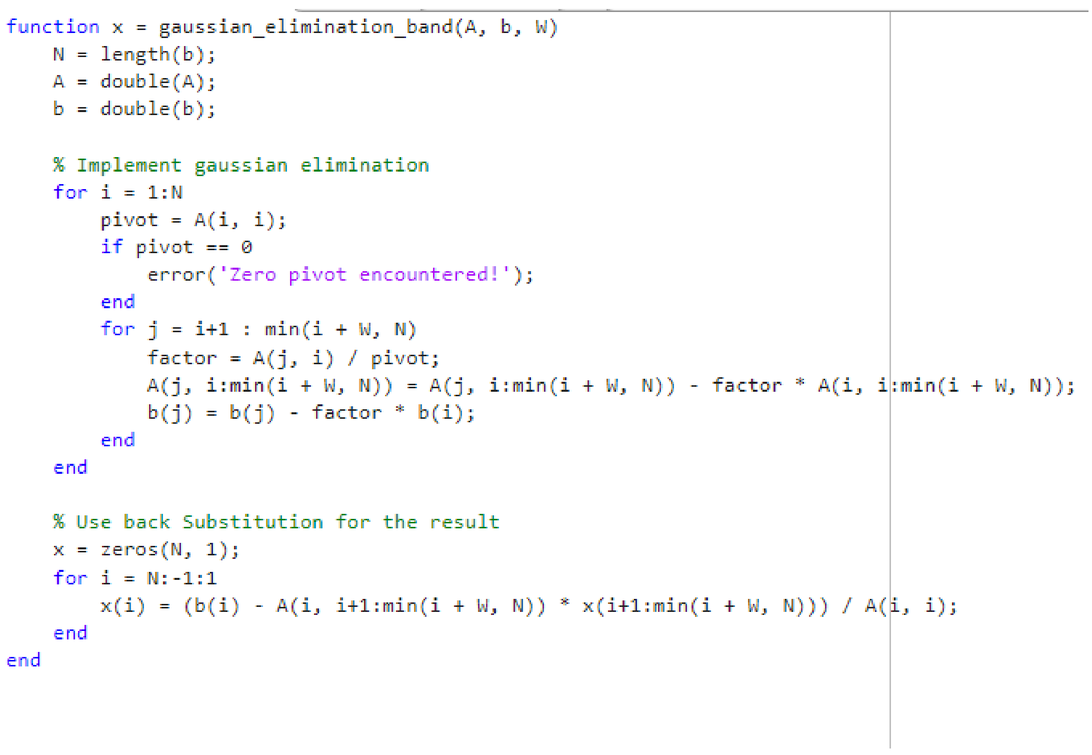


```
A Condition number: 1e+20
B Condition number: 1.0
C Condition number: 1.0
D Condition number: inf
```

Therefore, A, D is ill-conditioned and B, C is well-conditioned

Since D is singular matrix, the condition number of D is infinity.

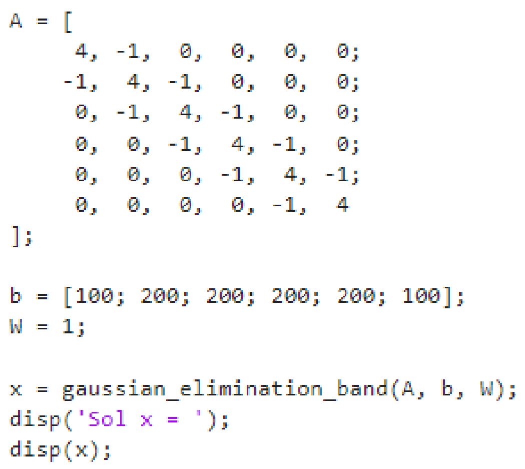
6



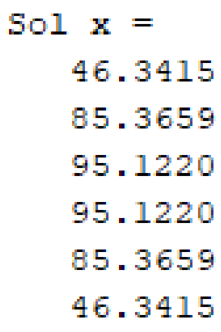
```
function x = gaussian_elimination_band(A, b, W)
    N = length(b);
    A = double(A);
    b = double(b);

    % Implement gaussian elimination
    for i = 1:N
        pivot = A(i, i);
        if pivot == 0
            error('Zero pivot encountered!');
        end
        for j = i+1 : min(i + W, N)
            factor = A(j, i) / pivot;
            A(j, i:min(i + W, N)) = A(j, i:min(i + W, N)) - factor * A(i, i:min(i + W, N));
            b(j) = b(j) - factor * b(i);
        end
    end

    % Use back Substitution for the result
    x = zeros(N, 1);
    for i = N:-1:1
        x(i) = (b(i) - A(i, i+1:min(i + W, N)) * x(i+1:min(i + W, N))) / A(i, i);
    end
end
```

I finished a python program and a MATLB program for this problem. The method I use is first use Gaussian elimination to let all the lower triangular value to be 0. Since it's a band matrix, we only need to do this operation limit times for each row, which the times are determined by the band width. After eliminating all the lower triangular values, I simply use back substitution for the result. Below is the testing.

This is the same matrix in problem 2

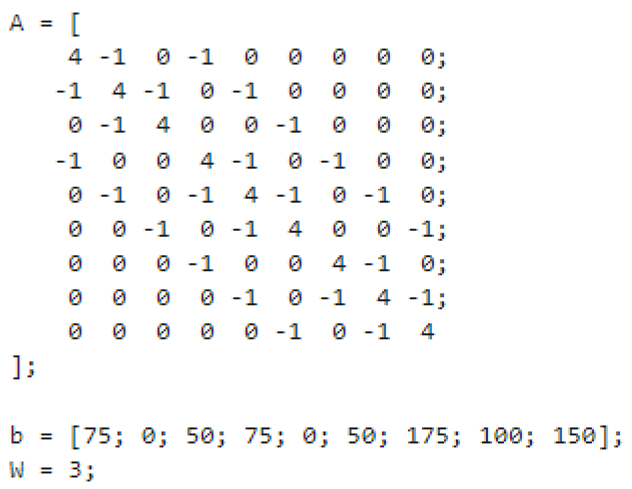


```
A = [
     4, -1,  0,  0,  0,  0;
    -1,  4, -1,  0,  0,  0;
     0, -1,  4, -1,  0,  0;
     0,  0, -1,  4, -1,  0;
     0,  0,  0, -1,  4, -1;
     0,  0,  0,  0, -1,  4
];

b = [100; 200; 200; 200; 200; 100];
W = 1;

x = gaussian_elimination_band(A, b, W);
disp('Sol x = ');
disp(x);
```

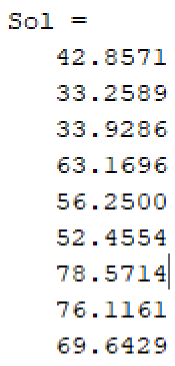


```
Sol x =
   46.3415
   85.3659
   95.1220
   95.1220
   85.3659
   46.3415
```

Another matrix with bandwidth 3 for testing



```
A = [
    4 -1  0 -1  0  0  0  0  0;
   -1  4 -1  0 -1  0  0  0  0;
    0 -1  4  0  0 -1  0  0  0;
   -1  0  0  4 -1  0 -1  0  0;
    0 -1  0 -1  4 -1  0 -1  0;
    0  0 -1  0 -1  4  0  0 -1;
    0  0  0 -1  0  0  4 -1  0;
    0  0  0  0 -1  0 -1  4 -1;
    0  0  0  0  0 -1  0 -1  4
];

b = [75; 0; 50; 75; 0; 50; 175; 100; 150];
W = 3;
```



```
Sol =
    42.8571
    33.2589
    33.9286
    63.1696
    56.2500
    52.4554
    78.5714
    76.1161
    69.6429
```