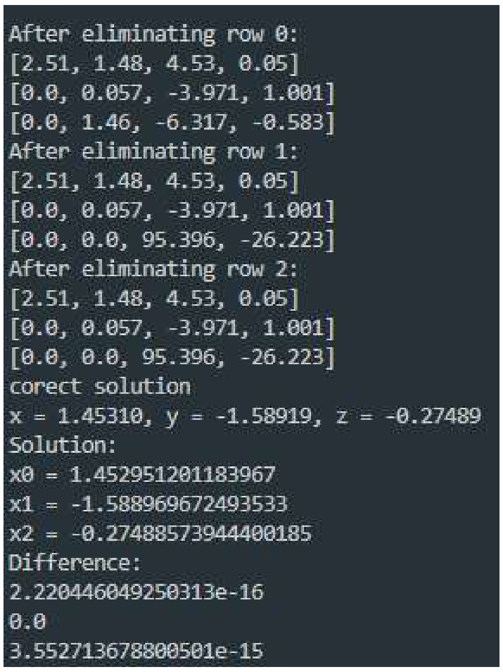
NM hw2 report

P1.

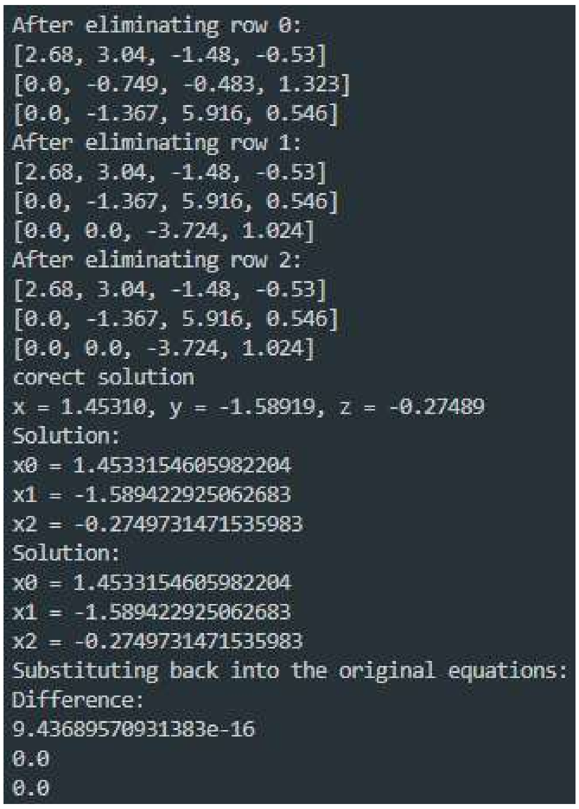
p1(a) gaussian with no interchanges



```
After eliminating row 0:
[2.51, 1.48, 4.53, 0.05]
[0.0, 0.057, -3.971, 1.001]
[0.0, 1.46, -6.317, -0.583]
After eliminating row 1:
[2.51, 1.48, 4.53, 0.05]
[0.0, 0.057, -3.971, 1.001]
[0.0, 0.0, 95.396, -26.223]
After eliminating row 2:
[2.51, 1.48, 4.53, 0.05]
[0.0, 0.057, -3.971, 1.001]
[0.0, 0.0, 95.396, -26.223]
corect solution
x = 1.45310, y = -1.58919, z = -0.27489
Solution:
x0 = 1.452951201183967
x1 = -1.588969672493533
x2 = -0.27488573944400185
Difference:
2.220446049250313e-16
0.0
3.552713678800501e-15
```

觀察到small divisor 0.057，導致浮點數誤差被放大。因根據題目要求四捨五入到小數點第三位，造成浮點誤差，而因除數很小，所以接下來乘上的倍數也會很大(1.46/0.057)，因此將誤差放大。

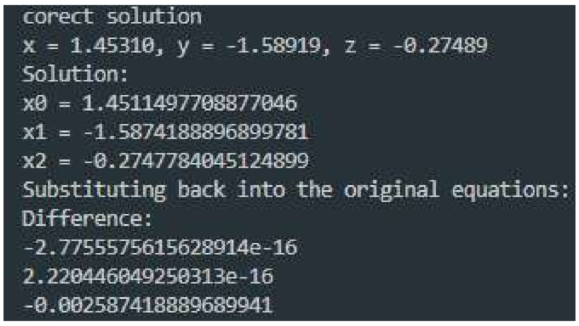
p1(b) gaussian with partial pivoting



```
After eliminating row 0:
[2.68, 3.04, -1.48, -0.53]
[0.0, -0.749, -0.483, 1.323]
[0.0, -1.367, 5.916, 0.546]
After eliminating row 1:
[2.68, 3.04, -1.48, -0.53]
[0.0, -1.367, 5.916, 0.546]
[0.0, 0.0, -3.724, 1.024]
After eliminating row 2:
[2.68, 3.04, -1.48, -0.53]
[0.0, -1.367, 5.916, 0.546]
[0.0, 0.0, -3.724, 1.024]
corect solution
x = 1.45310, y = -1.58919, z = -0.27489
Solution:
x0 = 1.4533154605982204
x1 = -1.589422925062683
x2 = -0.2749731471535983
Solution:
x0 = 1.4533154605982204
x1 = -1.589422925062683
x2 = -0.2749731471535983
Substituting back into the original equations:
Difference:
9.436895709313830e-16
0.0
0.0
```

透過partial pivoting每次選絕對值最大的當作pivot避免small divisor把誤差放大。因此可以在結果中看到相對p1(a)誤差小了很多。

p1(c) gaussian with chop to 3 significant digits



```
corect solution
x = 1.45310, y = -1.58919, z = -0.27489
Solution:
x0 = 1.4511497708877046
x1 = -1.5874188896899781
x2 = -0.2747784045124899
Substituting back into the original equations:
Difference:
-2.7755575615628914e-16
2.220446049250313e-16
-0.002587418889689941
```

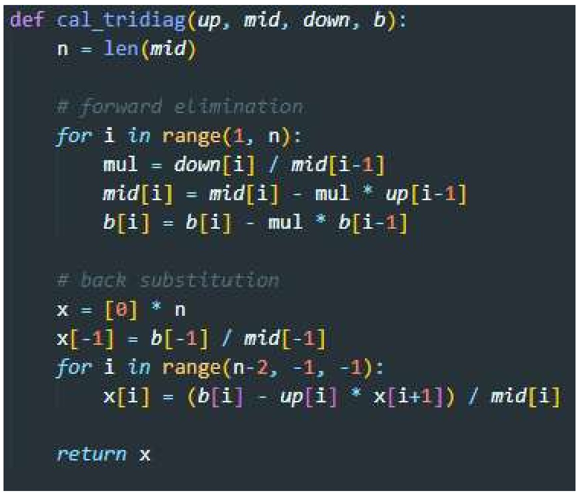
改成把第3位後的直接捨去，因此造成更大的誤差。

p1(d) substitute back to equations

可以在p1a-c的結果輸出最下方看到，差距最大的是chop (c)、然後是no interchanges(a)，差距最小的是做partial pivoting的(b)。

p2.

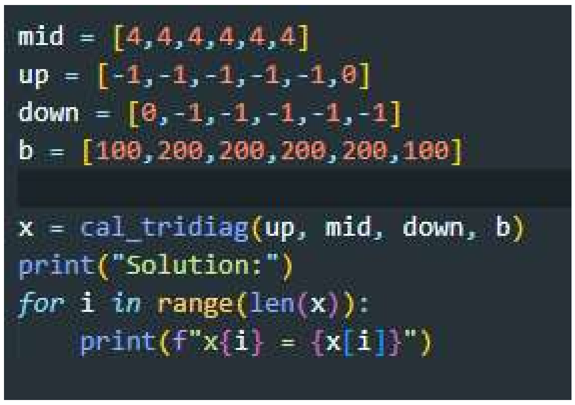
p2(a) function cal_tridiag



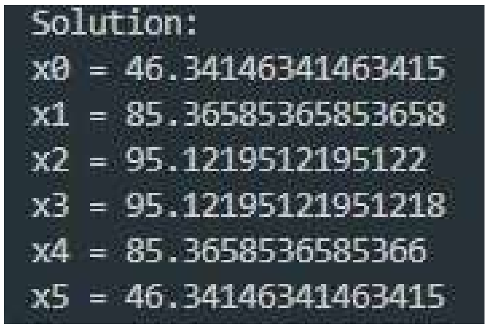
```
def cal_tridiag(up, mid, down, b):
    n = len(mid)

    # forward elimination
    for i in range(1, n):
        mul = down[i] / mid[i-1]
        mid[i] = mid[i] - mul * up[i-1]
        b[i] = b[i] - mul * b[i-1]

    # back substitution
    x = [0] * n
    x[-1] = b[-1] / mid[-1]
    for i in range(n-2, -1, -1):
        x[i] = (b[i] - up[i] * x[i+1]) / mid[i]

    return x
```

- 僅儲存三條對角線（up, mid, down）
- 使用 forward elimination 消去下方的對角線
- 使用 back substitution 求解

p2(b) 輸入矩陣求解



```
mid = [4,4,4,4,4,4]
up = [-1,-1,-1,-1,-1,0]
down = [0,-1,-1,-1,-1,-1]
b = [100,200,200,200,200,100]

x = cal_tridiag(up, mid, down, b)
print("Solution:")
for i in range(len(x)):
    print(f"x{i} = {x[i]}")
```



```
Solution:
x0 = 46.34146341463415
x1 = 85.36585365853658
x2 = 95.1219512195122
x3 = 95.12195121951218
x4 = 85.3658536585366
x5 = 46.34146341463415
```

p2(c)

每次迴圈（n-1 次）進行以下操作：

- 1 次除法：`mul = down[i] / mid[i-1]`
- 2 次乘法：`mul * up[i-1]`、`mul * b[i-1]`
- 2 次減法：更新 `mid[i]`、`b[i]`

⇒ 每回合共5次運算，共 (n−1) * 5 次

back substitution做了:

- 1次除法：`x[-1] = b[-1] / mid[-1]`

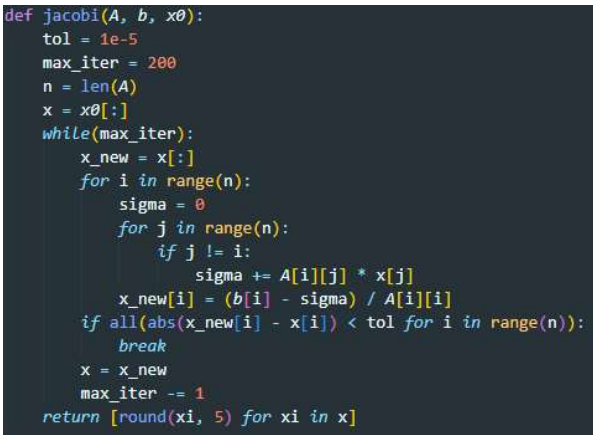
- 每次迴圈內：1 乘法 + 1 減法 + 1 除法

⇒ 1+3(n−1) 次

總計: 8n - 7次運算

p3.

p3 (a) jacobi

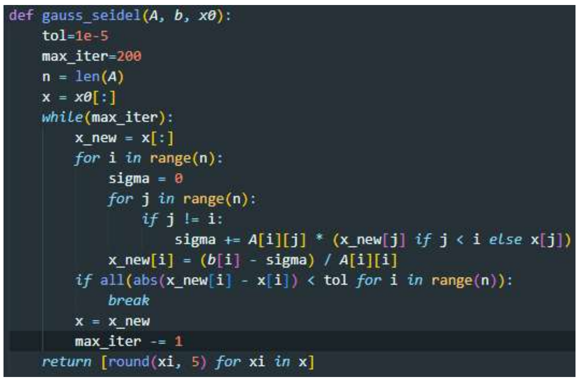


```
def jacobi(A, b, x0):
    tol = 1e-5
    max_iter = 200
    n = len(A)
    x = x0[:]
    while(max_iter):
        x_new = x[:]
        for i in range(n):
            sigma = 0
            for j in range(n):
                if j != i:
                    sigma += A[i][j] * x[j]
            x_new[i] = (b[i] - sigma) / A[i][i]
        if all(abs(x_new[i] - x[i]) < tol for i in range(n)):
            break
        x = x_new
        max_iter -= 1
    return [round(xi, 5) for xi in x]
```

對於第i列的Xk[j]，把其他的元素乘上矩陣元素A[i][j]全部丟到右邊用b[i]減掉。

更新解Xk[i] = b[i] - sigma / A[i]直到X更新的差值小於tol或是達到max_iter。

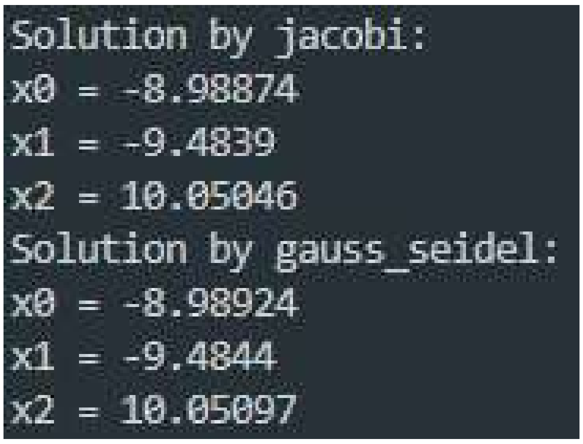
p3 (b) gauss-seidel



```
def gauss_seidel(A, b, x0):
    tol=1e-5
    max_iter=200
    n = len(A)
    x = x0[:]
    while(max_iter):
        x_new = x[:]
        for i in range(n):
            sigma = 0
            for j in range(n):
                if j != i:
                    sigma += A[i][j] * (x_new[j] if j < i else x[j])
            x_new[i] = (b[i] - sigma) / A[i][i]
        if all(abs(x_new[i] - x[i]) < tol for i in range(n)):
            break
        x = x_new
        max_iter -= 1
    return [round(xi, 5) for xi in x]
```

基本跟jacobi相似，只是對於更新完的X，也就是在j < i的時候用X_new，也就是更新完的X去計算新的值。

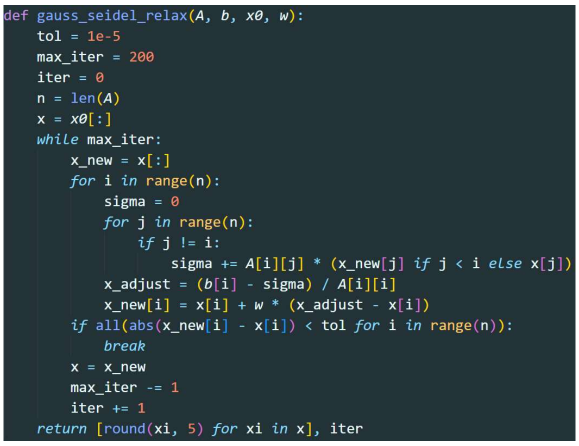
p3 result



```
Solution by jacobi:
x0 = -8.98874
x1 = -9.4839
x2 = 10.05046
Solution by gauss_seidel:
x0 = -8.98924
x1 = -9.4844
x2 = 10.05097
```

P4

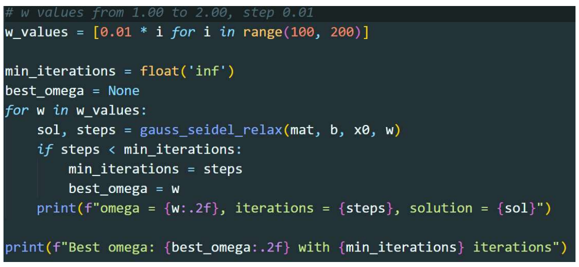
add relaxation factor in gauss-seidel



```python
def gauss_seidel_relax(A, b, x0, w):
    tol = 1e-5
    max_iter = 200
    iter = 0
    n = len(A)
    x = x0[:]
    while max_iter:
        x_new = x[:]
        for i in range(n):
            sigma = 0
            for j in range(n):
                if j != i:
                    sigma += A[i][j] * (x_new[j] if j < i else x[j])
            x_adjust = (b[i] - sigma) / A[i][i]
            x_new[i] = x[i] + w * (x_adjust - x[i])
        if all(abs(x_new[i] - x[i]) < tol for i in range(n)):
            break
        x = x_new
        max_iter -= 1
        iter += 1
    return [round(xi, 5) for xi in x], iter
```

加入w參數作為relaxation factor加速收斂，透過加大x趨近真正解的步長使迭代次數減少。

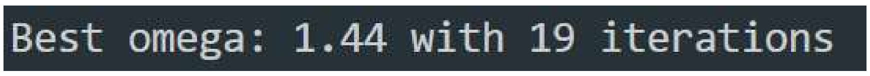
test w



```python
# w values from 1.00 to 2.00, step 0.01
w_values = [0.01 * i for i in range(100, 200)]

min_iterations = float('inf')
best_omega = None
for w in w_values:
    sol, steps = gauss_seidel_relax(mat, b, x0, w)
    if steps < min_iterations:
        min_iterations = steps
        best_omega = w
    print(f"omega = {w:.2f}, iterations = {steps}, solution = {sol}")

print(f"Best omega: {best_omega:.2f} with {min_iterations} iterations")
```

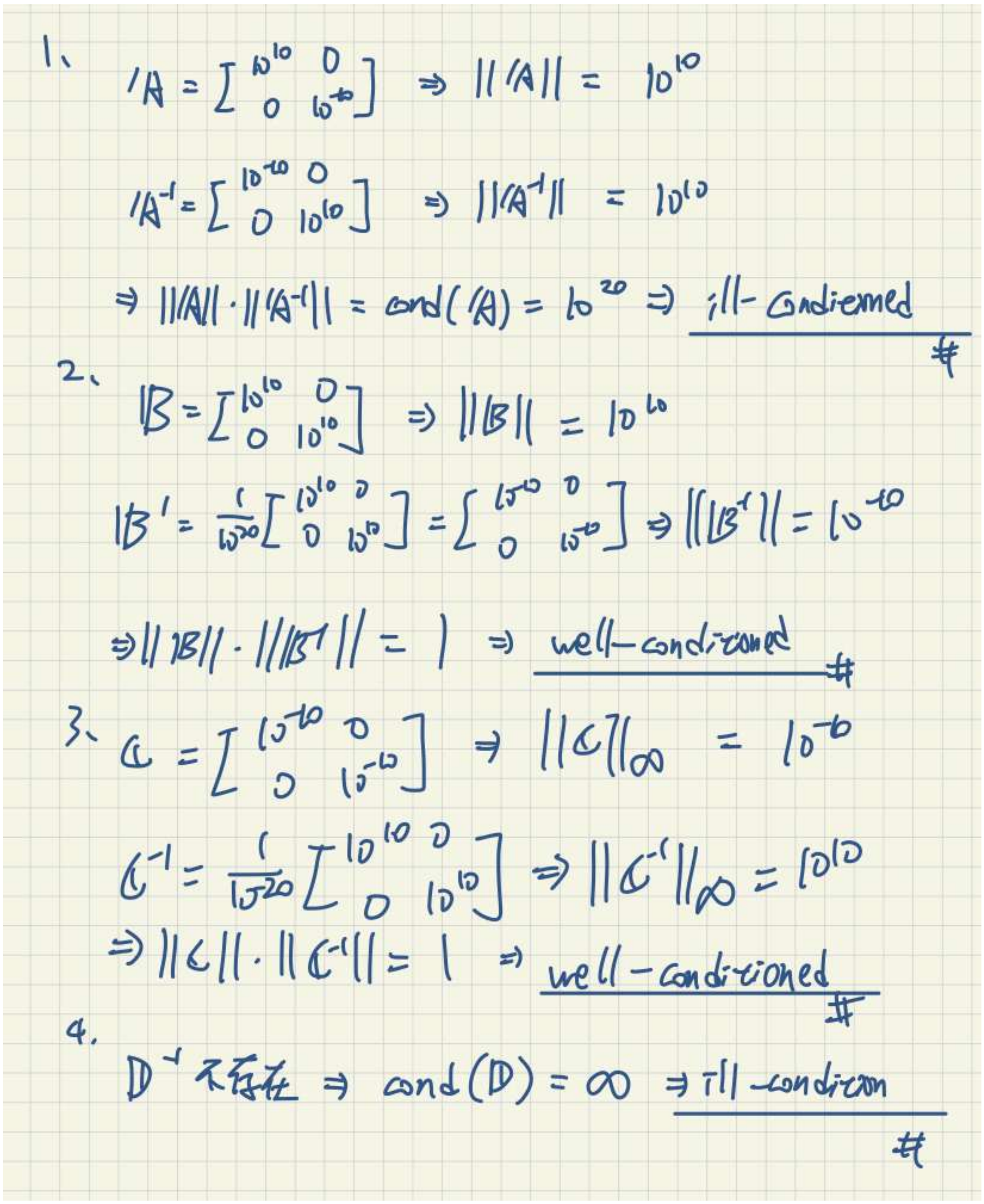
測試從1.00到2.00，間隔0.0.1的w值，並記錄每個w值的話iteration，紀錄最小值

result



```
Best omega: 1.44 with 19 iterations
```

P5



1、 $A = \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{-10} \end{bmatrix} \Rightarrow \|A\| = 10^{10}$

$A^{-1} = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{10} \end{bmatrix} \Rightarrow \|A^{-1}\| = 10^{10}$

$\Rightarrow \|A\| \cdot \|A^{-1}\| = \text{cond}(A) = 10^{20} \Rightarrow$ ill-Condiemed

#

2、 $B = \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix} \Rightarrow \|B\| = 10^{10}$

$B' = \frac{1}{10^{20}} \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix} = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix} \Rightarrow \|B^{-1}\| = 10^{-10}$

$\Rightarrow \|B\| \cdot \|B^{-1}\| = 1 \Rightarrow$ well-conditioned

#

3、 $C = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix} \Rightarrow \|C\|_\infty = 10^{-10}$

$C^{-1} = \frac{1}{10^{-20}} \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix} \Rightarrow \|C^{-1}\|_\infty = 10^{10}$

$\Rightarrow \|C\| \cdot \|C^{-1}\| = 1 \Rightarrow$ well-conditioned

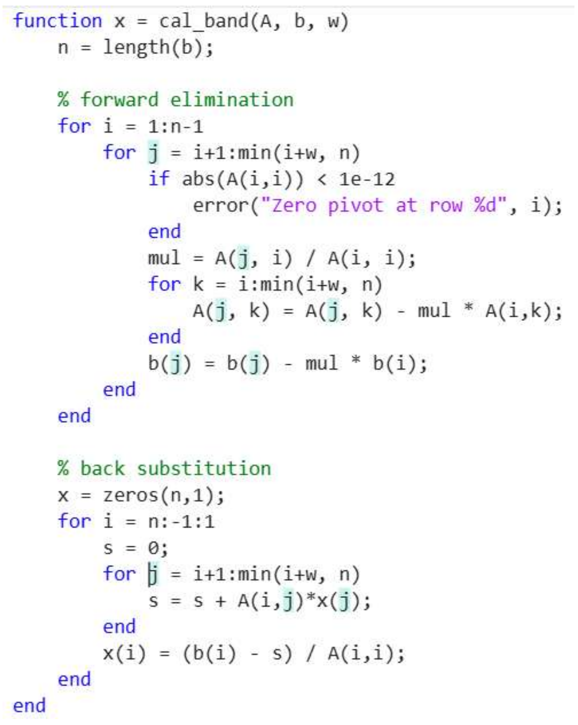
#

4. D^{-1} 不存在 $\Rightarrow \text{cond}(D) = \infty \Rightarrow$ ill-condition

#

P6

algorithm

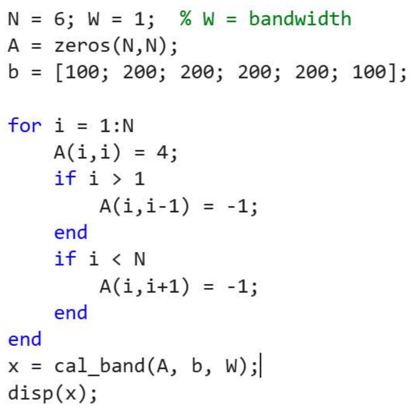


```
function x = cal_band(A, b, w)
    n = length(b);

    % forward elimination
    for i = 1:n-1
        for j = i+1:min(i+w, n)
            if abs(A(i,i)) < 1e-12
                error("Zero pivot at row %d", i);
            end
            mul = A(j, i) / A(i, i);
            for k = i:min(i+w, n)
                A(j, k) = A(j, k) - mul * A(i,k);
            end
            b(j) = b(j) - mul * b(i);
        end
    end

    % back substitution
    x = zeros(n,1);
    for i = n:-1:1
        s = 0;
        for j = i+1:min(i+w, n)
            s = s + A(i,j)*x(j);
        end
        x(i) = (b(i) - s) / A(i,i);
    end
end
```

利用band matrix的特殊結構來加快計算。因為這種矩陣的非零元素只集中在主對角線以及上下各 w 條對角線，也就是說，每一列只會影響到前後最多 w 個變數，其他都是 0，不會有影響。

這讓我們在實作gaussain elimination的時候，可以不用整列都處理，只需要處理從第 i 列到第 i + W 列之間的欄位，節省運算量。程式中會用 `min(i + w, n)` 來控制這個範圍。同樣的，在做 back substitution時，也只需要用到後面最多 w 個變數。

test



```
N = 6; W = 1;  % W = bandwidth
A = zeros(N,N);
b = [100; 200; 200; 200; 200; 100];

for i = 1:N
    A(i,i) = 4;
    if i > 1
        A(i,i-1) = -1;
    end
    if i < N
        A(i,i+1) = -1;
    end
end
x = cal_band(A, b, W);
disp(x);
```

使用第二題的矩陣 (bandwidth = 1)來做測試

result

```
46.3415
85.3659
95.1220
95.1220
85.3659
46.3415
```