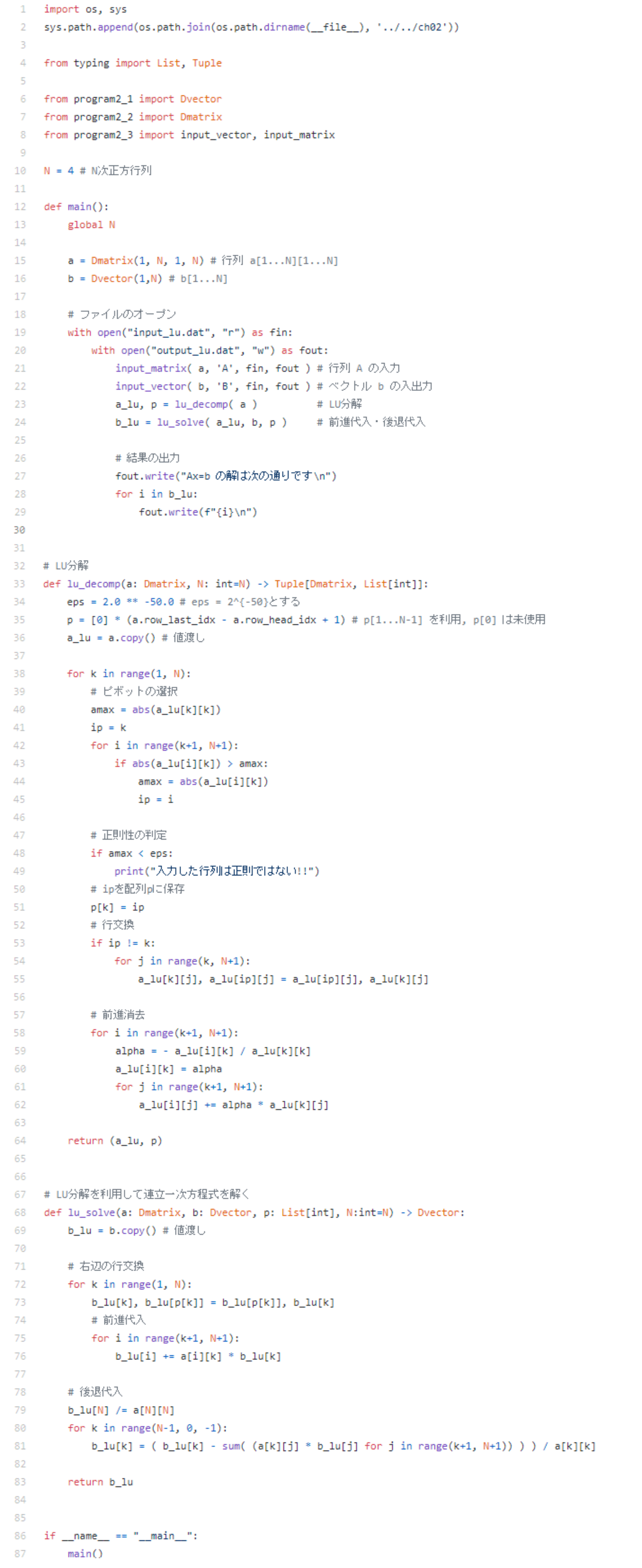

```python
import os, sys
sys.path.append(os.path.join(os.path.dirname(__file__), '../../ch02'))

from typing import List, Tuple

from program2_1 import Dvector
from program2_2 import Dmatrix
from program2_3 import input_vector, input_matrix

N = 4 # N次正方行列

def main():
    global N

    a = Dmatrix(1, N, 1, N) # 行列 a[1...N][1...N]
    b = Dvector(1,N) # b[1...N]

    # ファイルのオープン
    with open("input_lu.dat", "r") as fin:
        with open("output_lu.dat", "w") as fout:
            input_matrix( a, 'A', fin, fout ) # 行列 A の入力
            input_vector( b, 'B', fin, fout ) # ベクトル b の入出力
            a_lu, p = lu_decomp( a )          # LU分解
            b_lu = lu_solve( a_lu, b, p )     # 前進代入・後退代入

            # 結果の出力
            fout.write("Ax=b の解は次の通りです\n")
            for i in b_lu:
                fout.write(f"{i}\n")


# LU分解
def lu_decomp(a: Dmatrix, N: int=N) -> Tuple[Dmatrix, List[int]]:
    eps = 2.0 ** -50.0 # eps = 2^{-50}とする
    p = [0] * (a.row_last_idx - a.row_head_idx + 1) # p[1...N-1] を利用, p[0] は未使用
    a_lu = a.copy() # 値渡し

    for k in range(1, N):
        # ピボットの選択
        amax = abs(a_lu[k][k])
        ip = k
        for i in range(k+1, N+1):
            if abs(a_lu[i][k]) > amax:
                amax = abs(a_lu[i][k])
                ip = i

        # 正則性の判定
        if amax < eps:
            print("入力した行列は正則ではない!!")
        # ipを配列pに保存
        p[k] = ip
        # 行交換
        if ip != k:
            for j in range(k, N+1):
                a_lu[k][j], a_lu[ip][j] = a_lu[ip][j], a_lu[k][j]

        # 前進消去
        for i in range(k+1, N+1):
            alpha = - a_lu[i][k] / a_lu[k][k]
            a_lu[i][k] = alpha
            for j in range(k+1, N+1):
                a_lu[i][j] += alpha * a_lu[k][j]

    return (a_lu, p)


# LU分解を利用して連立一次方程式を解く
def lu_solve(a: Dmatrix, b: Dvector, p: List[int], N:int=N) -> Dvector:
    b_lu = b.copy() # 値渡し

    # 右辺の行交換
    for k in range(1, N):
        b_lu[k], b_lu[p[k]] = b_lu[p[k]], b_lu[k]
        # 前進代入
        for i in range(k+1, N+1):
            b_lu[i] += a[i][k] * b_lu[k]

    # 後退代入
    b_lu[N] /= a[N][N]
    for k in range(N-1, 0, -1):
        b_lu[k] = ( b_lu[k] - sum( (a[k][j] * b_lu[j] for j in range(k+1, N+1)) ) ) / a[k][k]

    return b_lu


if __name__ == "__main__":
    main()
```