THÔNG TIN SINH VIÊN

Mã số sinh viên: 23127447
Họ và tên: Nguyễn Thanh Owen
Lớp: 23CLC06
Email: ntowen23@clc.fitus.edu.vn

Thư viện sử dụng

'sympy' : dùng để lấy các kí hiệu Xi khi hệ phương trình có vô số nghiệm và in ma trận mở rộng và nghiệm

Hàm Gauss_elimination(A)

Input: A, là ma trận mở rộng của hệ phương trình Ax = b
Output: ma trận có dạng bậc thang có được từ ma trận A

Mô tả thuật toán hàm Gauss_elimination(A)

Bước 1. Xác định cột trái nhất không chứa toàn số 0.

Bước 2. Nếu phần tử ở dòng hiện tại và cột vừa tìm được bằng 0, thì đổi chỗ dòng hiện tại với dòng đầu tiên bên dưới mà có phần tử khác 0 trong cột đó để đưa phần tử khác 0 lên vị trí dòng hiện tại.

Bước 3. Với số hạng đầu cột nhận được từ Bước 2 là $a \neq 0$, nhân dòng chứa nó với $1/a$ để có số dẫn đầu 1 (leading 1).

Bước 4. Cộng một bội số thích hợp của dòng đầu cho từng dòng dưới để biến các số hạng bên dưới số dẫn đầu thành 0.

Bước 5. Che dòng đầu đã làm xong. Lặp lại các bước cho đến khi được ma trận bậc thang.

Cài đặt code

In [25]:
```
import sympy as sp
import time
def swap_rows(A, row1, row2):
    A[row1], A[row2] = A[row2], A[row1]

def Gauss_elimination(A):
    numRow = len(A)
    numCol = len(A[0])

    leftMost = 0 #cot trai nhat khong chua toan so 0
    for row in range(numRow):

        #B1: Xac dinh cot trai nhat khong chua toan so 0
        while leftMost < numCol:
            check = True
            for i in range(row, numRow):
                if A[i][leftMost] != 0:
                    check = False
                    break
            if check == True:
                leftMost += 1
            else:
                break

        if leftMost >= numCol:
            return A

        #B2: kiem tra phan tu dau tien cua dong hien tai co = 0 hay khong
            #neu co thi swap voi 1 dong khac
        if A[row][leftMost] == 0:
            for i in range(row + 1, numRow):
                if A[i][leftMost] != 0:
                    swap_rows(A, row, i)
                    break

        #B3: Nhan dong "row" nghich dao voi 1/A[row][leftMost] de chuyen phan tu dau tien thanh 1
        pivot = A[row][leftMost]
```

```
        for j in range(leftMost, numCol):
            A[row][j] /= pivot

        #B4: Cong boi so thich hop cua dong dau cho tung dong ben duoi de bien cac so hang ben duoi so dan dau thanh
        for i in range(row + 1, numRow):
            factor = A[i][leftMost] / A[row][leftMost]
            for j in range(leftMost, numCol):
                A[i][j] = A[i][j] - A[row][j] * factor

        #B5: tang chi so leftMost va lap lai qua trinh
        leftMost += 1
    return A
```

Hàm back_substitution

Input: A, là ma trận có dạng bậc thang thu được từ ma trận mở rộng của hệ phương trình Ax = b.
Output: nghiệm của hệ phương trình (trường hợp nghiệm duy nhất/ vô số nghiệm) hoặc thông báo hệ phương trình vô nghiệm.

Mô tả thuật toán hàm back_substitution()

Bước 1. Khởi tạo danh sách các biến tự do x1, x2, ..., xn bằng 'sympy'.

Bước 2. Tìm dòng dưới cùng khác 0.

Bước 3. Kiểm tra vô nghiệm: Nếu dòng cuối có số hạng tự do ([0, 0, ..., 0 | b] với b ≠ 0), thì hệ vô nghiệm.

Bước 4. Kiểm tra vô số nghiệm hoặc nghiệm duy nhất:

- Nếu số dòng khác 0 < số ẩn → vô số nghiệm (có biến tự do).
- Ngược lại nếu dòng khác 0 = số ẩn → có nghiệm duy nhất.

Bước 5. Giải nghiệm:

- giả sử A là ma trận $n \times m$
- Duyệt từ dòng dưới cùng khác 0 lên.

- Với mỗi dòng thứ i (tính từ 1) cột thứ j (tính từ 1) đầu tiên mà $A[i][j]$ khác 0,Khi đó nghiệm được tính bằng:
  $x_j = \frac{A[i][m-1] - \sum_{k=j+1}^{m-2} A[i][k] \cdot x_k}{A[i][j]}$

In [26]:
```
import sympy as sp
def back_substitution(A):
    numRow = len(A)
    numCol = len(A[0])

    symbols = sp.symbols(f'x1:{numCol}')  #tao bien tu do khi phuong trinh VSN
    X = list(symbols)

    downMost = numRow - 1 #vi tri dong dau tien khac 0 tu duoi len
    while downMost >= 0:
        check = True
        for j in range(numCol):
            if A[downMost][j] != 0:
                check = False
                break
        if check == False:
            break
        else:
            downMost -= 1

    #TH1: Vo nghiem
    check = True
    for j in range(numCol - 1):
        if A[downMost][j] != 0:
            check = False
            break
    if check == True and A[downMost][numCol - 1] != 0:
        #print("Phuong trinh vo nghiem!")
        return "Phuong trinh vo nghiem!"

    #TH2: Vo so nghiem va co nghiem duy nhat
    if (downMost + 1) < (numCol - 1):
        print("Phuong trinh co vo so nghiem:")
    else:
        print("Phuong trinh co nghiem duy nhat:")
    while downMost >= 0:
        for j in range(numCol - 1):
```

```
            if A[downMost][j] != 0:
                leftSum = 0
                for k in range(j + 1, numCol - 1):
                    leftSum += X[k]*A[downMost][k]
                X[j] = (A[downMost][numCol - 1] - leftSum) / A[downMost][j]
                break
        downMost -= 1
    return sp.Matrix(X)
```

Ví dụ cho các trường hợp:

Trường hợp 1: Vô nghiệm

In [27]:
```
A = [
    [2, -4, -1, 1],
    [1, -3, 1, 1],
    [3, -5, -3, 2]
]

start = time.time()
Gauss_elimination(A)
X = back_substitution(A)
end = time.time()
print('Ma tran mo rong sau khi rut gon:\n A = ')
sp.Matrix(A)
```

```
Ma tran mo rong sau khi rut gon:
 A =
```

Out[27]:
$$\begin{bmatrix} 1.0 & -2.0 & -0.5 & 0.5 \\ 0.0 & 1.0 & -1.5 & -0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

In [28]:
```
total_time = end-start
print(f"Thoi gian giai he phuong trinh: {total_time:.6f}")
X
```

```
Thoi gian giai he phuong trinh: 0.000000
```

Out[28]: 'Phuong trinh vo nghiem!'

Trường hợp 2: Vô số nghiệm

In [29]:
```
A = [
    [1, -1, 1, -3, 0],
    [2, -1, 4, -2, 0],
]

start = time.time()
Gauss_elimination(A)
X = back_substitution(A)
end = time.time()
print('Ma tran mo rong sau khi rut gon:\n A = ')
sp.Matrix(A)
```

```
Phuong trinh co vo so nghiem:
Ma tran mo rong sau khi rut gon:
 A =
```

Out[29]:
$$\begin{bmatrix} 1.0 & -1.0 & 1.0 & -3.0 & 0.0 \\ 0.0 & 1.0 & 2.0 & 4.0 & 0.0 \end{bmatrix}$$

In [30]:
```
total_time = end-start
print(f"Thoi gian giai he phuong trinh: {total_time:.6f}")
print('X = ')
X
```

```
Thoi gian giai he phuong trinh: 0.001441
X =
```

Out[30]:
$$\begin{bmatrix} -3.0x_3 - 1.0x_4 \\ -2.0x_3 - 4.0x_4 \\ x_3 \\ x_4 \end{bmatrix}$$

Trường hợp 3: Có nghiệm duy nhất

In [31]:
```
A = [
    [1, -2, 3, -3],
    [2, 2, 0, 0],
    [0, -3, 4, 1],
    [1, 0, 1, -1]
]

start = time.time()
Gauss_elimination(A)
X = back_substitution(A)
end = time.time()
print('Ma tran mo rong sau khi rut gon:\n A = ')
sp.Matrix(A)
```

```
Phuong trinh co nghiem duy nhat:
Ma tran mo rong sau khi rut gon:
 A =
```

Out[31]:
$$\begin{bmatrix} 1.0 & -2.0 & 3.0 & -3.0 \\ 0.0 & 1.0 & -1.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 4.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

In [32]:
```
total_time = end-start
print(f"Thoi gian giai he phuong trinh: {total_time:.6f}")
print('X = ')
X
```

```
Thoi gian giai he phuong trinh: 0.000506
X =
```

Out[32]:
$$\begin{bmatrix} -5.0 \\ 5.0 \\ 4.0 \end{bmatrix}$$

Sử dụng linsovler() của

`linsolve()` là hàm trong thư viện **SymPy** dùng để **giải hệ phương trình tuyến tính** dưới dạng ký hiệu (symbolic).

Input:

- Ma trận hệ số (A) và vector kết quả (b)
- Hoặc dạng tổ hợp ((A, b))

Output:

- **Tập nghiệm** (solution set) của hệ phương trình, có thể là nghiệm duy nhất, nghiệm vô số hoặc không có nghiệm.

In [33]:
```python
def solve_augmented_matrix(Aug):
    Aug = sp.Matrix(Aug)
    A = Aug[:, :-1]  # ma trận hệ số
    b = Aug[:, -1]   # véctơ kết quả
    n = A.shape[1]
    x = sp.symbols(f'x1:{n+1}')

    start = time.time()
    sol = sp.linsolve((A, b), x)
    end = time.time()
    print(f"Thoi gian giai he phuong trinh: {(end - start):6f}")

    if not sol:
        return "He phuong trinh vo nghiem."
    elif len(sol.free_symbols) == 0:
        sol_tuple = list(sol)[0]
        sol_list = list(sol_tuple)
        print("He phuong trinh co nghiem duy nhat:")
        return sp.Matrix(sol_list)
    else:
        sol_tuple = list(sol)[0]
        sol_list = list(sol_tuple)
        print("He phuong trinh co vo so nghiem:")
        return sp.Matrix(sol_list)
```

Trường hợp 1: Vô nghiệm

In [34]:
```python
A = [
    [2, -4, -1, 1],
    [1, -3, 1, 1],
    [3, -5, -3, 2]
```

```
]

X = solve_augmented_matrix(A)
X
```

```
Thoi gian giai he phuong trinh: 0.000000
```

Out[34]: 'He phuong trinh vo nghiem.'

Trường hợp 2: Vô số nghiệm

In [35]:
```
A = [
    [1, -1, 1, -3, 0],
    [2, -1, 4, -2, 0],
]
X = solve_augmented_matrix(A)
print('X = ')
X
```

```
Thoi gian giai he phuong trinh: 0.000999
He phuong trinh co vo so nghiem:
X =
```

Out[35]:
$$\begin{bmatrix} -3x_3 - x_4 \\ -2x_3 - 4x_4 \\ x_3 \\ x_4 \end{bmatrix}$$

Trường hợp 3: Có nghiệm duy nhất

In [36]:
```
A = [
    [1, -2, 3, -3],
    [2, 2, 0, 0],
    [0, -3, 4, 1],
    [1, 0, 1, -1]
]
X = solve_augmented_matrix(A)
print('X = ')
X
```

```
Thoi gian giai he phuong trinh: 0.000998
He phuong trinh co nghiem duy nhat:
X =
```

Out[36]:
$$\begin{bmatrix} -5 \\ 5 \\ 4 \end{bmatrix}$$

📊 Bảng So Sánh: `Gauss_elimination()` và `back_substitution()` vs `sympy.linsolve`

Tiêu chí	Gauss_elimination() và back_substitution() (Tự cài)	`sympy.linsolve` (SymPy)
Thời gian xử lý	Chậm, đặc biệt với hệ lớn; thuật toán thủ công không tối ưu.	Tối ưu tốt với hệ vừa và nhỏ; dùng giải thuật đại số hiệu quả.
Độ chính xác	Phụ thuộc vào kiểu dữ liệu (*float* dễ sai số); khó xử lý số vô tỉ, căn.	Chính xác tuyệt đối (*symbolic*); không bị trôi số hay sai số làm tròn.
Khả năng xử lý	Kém, do không dùng thư viện tính toán hiệu suất cao (*NumPy*, *BLAS*,...)	Cao hơn nhiều, phù hợp cho *symbolic computation* và hệ vừa phải.
Quy mô hoạt động	Phù hợp hệ nhỏ hoặc vừa; dễ bị lỗi và khó kiểm soát khi số phương trình hoặc ẩn tăng cao.	Tốt cho hệ vừa; nhưng không mạnh bằng *NumPy/SciPy* khi giải hệ số lượng rất lớn.
Mục tiêu phù hợp	Học tập, giảng dạy.	Ứng dụng thực tế, giải toán biểu thức, kiểm chứng nghiệm.