# Student Information

**Môn học: Toán Ứng dụng và Thống kê**

**Họ tên: Bế Lã Anh Thư**

**Lớp: 22CLC02**

**MSSV: 22127402**

# Thuật toán Gauss-Jordan

**Thuật toán Gausian-Jordan** là một cách biển đổi ma trận với mục đích tìm ma trận khả nghịch của một ma trận $A \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ cho trước.

Giải thuật gồm các bước như sau:

- **Bước 1:** Lập ma trận mở rộng $(A|I_n)$, với $I_n$ là ma trận đơn vị cùng kích thước $A$.
- **Bước 2:** Đổi chỗ hai dòng, nếu cần thiết, để đưa số hạng khác $0$ về cột đầu, có thể sử dụng cách đưa dòng đầu tiên có số hạng khác $0$ về dòng đầu.
- **Bước 3:** Ta nhân dòng nhận được từ bước 2 với $\frac{1}{pivot}$, với $pivot$ là số hạng đầu cột nhận được ở bước 2 ($pivot \neq 0$).
- **Bước 4:** Biến đổi các số hạng cùng cột với $pivot$ về $0$, đồng thời áp dụng phép tính tương đương với các số còn lại nằm chung dòng với cột đang xét.
- **Bước 5:** Che dòng đầu đã làm xong, lặp lại cho tới khi nhận được ma trận $(I_n|A^{-1})$

# Cài đặt thuật toán

In [9]:
```python
from fractions import Fraction

def swap_col(A, pre_index, new_index):
    A[pre_index], A[new_index] = A[new_index], A[pre_index]

def create_identity_matrix(n):
    #create the Identity matrix size nxn
    return [[1 if row == col else 0 for col in range(n)] for row in range(n)]

def print_matrix(A):
    # print out matrix A
    for row in range(len(A)):
        for col in range(len(A[0])):
            print(Fraction(A[row][col]).limit_denominator(1000), end=' ')
        print()

def gauss_jordan(A, I):
```

```
    """
        Generate the inversion matrix of A.
        Input: A - square matrix, size nxn, I - identity matrix, size nxn.
        Output: Inversion matrix of A, size nxn
    """
    n = len(A)
    m = len(A[0])
    pivot = {
        "row": 0,
        "col": 0
    }

    if (m!=n):
        # if matrix is not square
        return []

    while pivot["row"] < n and pivot["col"] < m:

        if A[pivot["row"]][pivot["col"]] == 0:
            change_pivot = False

            for row in range(pivot["row"] + 1, n):
                # if there is cell with value != 0 in same col with pivot, swap tha
                if A[row][pivot["col"]] != 0:
                    swap_col(A, pivot["row"], row)
                    swap_col(I, pivot["row"], row)
                    change_pivot = True
                    break

            if not change_pivot:
                # no non-zero element found in same column w pivot, move to the nex
                return []

        # eliminate to get leading 1
        leading = Fraction(1/A[pivot["row"]][pivot["col"]])
        for ele in range(m):
            A[pivot["row"]][ele] *= leading
            I[pivot["row"]][ele] *= leading

        # eliminate other col to zero
        for ele_row in range(n):
            if ele_row == pivot["row"]:
                continue
            coefficient = A[ele_row][pivot["col"]]
            for ele_col in range(m):
                if ele_col >= pivot["row"]:
                    A[ele_row][ele_col] -= coefficient*A[pivot["row"]][ele_col]
                I[ele_row][ele_col] -= coefficient*I[pivot["row"]][ele_col]

        pivot["row"] += 1
        pivot["col"] += 1

    return I

def inversion(A):
    I = create_identity_matrix(len(A))
```

```
    invertible_matrix = gauss_jordan(A, I)

    return invertible_matrix

#=======================================

# USING LIBRARY
import numpy as np
def calculate_using_numpy(A):
    A = np.array(A)

    # Compute the determinant
    determinant = np.linalg.det(A)
    if determinant == 0:
        return None

    # Compute the inverse using numpy
    A_inv = np.linalg.inv(A)
    return A_inv
```

In [10]:

```
# OUTPUT MẪU (BÀI 3 - BT TUẦN 3)
bt3a = [
    [1,2,1],
    [3,7,3],
    [2,3,4]
]

bt3b = [
    [1,-1,2],
    [1,1,-2],
    [1,1,4]
]

bt3c = [
    [1,2,3],
    [2,5,3],
    [1,0,8]
]

bt3d = [
    [-1,3,-4],
    [2,4,1],
    [-4,2,-9]
]

bt3 = [bt3a, bt3b, bt3c, bt3d]

from copy import deepcopy
import time
for bt in bt3:

    # print out the matrix A
    print(f'Ma trận vuông A:')
    print_matrix(bt)
    print()
```

```
    # compute the inverse manually
    start_time = time.time()
    inversion_matrix = inversion(deepcopy(bt))
    gauss_jordan_time = time.time() - start_time

    if not inversion_matrix:
        print("Ma trận không khả nghịch.")
    else:
        print("Ma trận khả nghịch:")
        print_matrix(inversion_matrix)
    print(f'Running times: {gauss_jordan_time:.6f}')
    print()

    # Compute the inverse using NumPy
    start_time = time.time()
    np_inv = calculate_using_numpy(deepcopy(bt))
    numpy_time = time.time() - start_time

    if np_inv is None:
        print("Ma trận không khả nghịch (NumPy).")
    else:
        print("Ma trận khả nghịch (NumPy):")
        print_matrix(np_inv)
    print(f'Running times: {numpy_time:.6f}')
    print('-----------------------------------')
```

```
Ma trận vuông A:
1 2 1
3 7 3
2 3 4

Ma trận khả nghịch:
19/2 -5/2 -1/2
-3 1 0
-5/2 1/2 1/2
Running times: 0.000998

Ma trận khả nghịch (NumPy):
19/2 -5/2 -1/2
-3 1 0
-5/2 1/2 1/2
Running times: 0.000000
-----------------------------------
Ma trận vuông A:
1 -1 2
1 1 -2
1 1 4

Ma trận khả nghịch:
1/2 1/2 0
-1/2 1/6 1/3
0 -1/6 1/6
Running times: 0.000997

Ma trận khả nghịch (NumPy):
1/2 1/2 0
-1/2 1/6 1/3
0 -1/6 1/6
Running times: 0.000000
-----------------------------------
Ma trận vuông A:
1 2 3
2 5 3
1 0 8

Ma trận khả nghịch:
-40 16 9
13 -5 -3
5 -2 -1
Running times: 0.000000

Ma trận khả nghịch (NumPy):
-40 16 9
13 -5 -3
5 -2 -1
Running times: 0.000000
-----------------------------------
Ma trận vuông A:
-1 3 -4
2 4 1
-4 2 -9
```

```
Ma trận không khả nghịch.
Running times: 0.000000

Ma trận không khả nghịch (NumPy).
Running times: 0.000000
-----------------------------------
```

# Giải thích hàm

**1. Hàm `gauss_jordan(A, I)`:**

Hàm này nhận một ma trận mở rộng $(A|I_n)$, trong đó $A \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ là ma trận vuông cho trước và $I_n$ là ma trận đơn vị cùng kích thước với $A$. Từ đó, trả về kết quả là ma trận khả nghịch $A^{-1}$ của ma trận $A$. Hàm được thực hiện với ý tưởng như sau:

- Kiểm tra nếu ma trận không vuông, trả về ma trận rỗng (ma trận không khả nghịch).
- Lặp qua ma trận, kiểm tra các phần tử khác 0 dẫn đầu ($pivots$) trong mỗi cột.
- Nếu một $pivots$ là 0, nó sẽ tìm kiếm một hàng bên dưới nó có phần tử khác 0 trong cùng một cột và hoán đổi các hàng đó. Nếu phép hoán đổi không được xảy ra, ma trận không thể biến đổi về dạng ma trận đơn vị, trả về ma trận rỗng (ma trận không khả nghịch).
- Khi tìm thấy một $pivots$, nó sẽ loại bỏ các phần tử khác trong cùng một cột với nó bằng cách trừ một bội số của hàng pivot.
- Quá trình tiếp tục cho đến khi toàn bộ ma trận A trở thành một ma trận đơn vị, kết quả cuối cùng ta có được $I$ từ ma trận đơn vị biến đổi thành ma trận khả nghịch $A^{-1}$.

**2. Hàm `inversion(A)`:**

Hàm nhận vào $A$ là ma trận vuông kích thước $n \times n$ cho trước, trả về $A^{-1}$ là ma trận khả nghịch của A. Ý tưởng hàm như sau:

- Nhận vào ma trận $A$ đã cho, tạo ra ma trận đơn vị $I_n$ có cùng kích thước với ma trận $A$.
- Sử dụng hai ma trận đã có, dùng hàm `gauss_jordan(A,I)` biến đổi, nhận về *invertible_matrix* là ma trận khả nghịch của $A$ (trong trường hợp không có ma trận khả nghịch, *invertible_matrix* rỗng).
- Trả về *invertible_matrix*.

**3. Sử dụng thư viện:**

Hàm `calculate_using_numpy(A)` sử dụng thư viện để tính toán ma trận khả nghịch.

- Tính toán định thức của ma trận $A$ bằng hàm *linalg.det(A)*, nếu $det(A) = 0$ tức ma trận không khả nghịch, trả về *None*.
- Sử dụng hàm *linalg.inv(A)* để tìm ma trận khả nghịch và trả về nó.

Hàm *linalg.inv(A)* được thực hiện với ý tưởng như sau:

- Bước 1: Phân rã ma trận $A$ về dạng $A = L \times U$, với:
  - $L$: ma trận tam giác dưới.
  - $U$: ma trận tam giác trên.
- Bước 2: Giải ma trận khả nghịch:
  - Giải ma trận $L \times Y = I$, với $Y$ sử dụng forward substitution.
  - Giải ma trận $U \times A^{-1} = Y$ với $A^{-1}$ sử dụng backward substitution.

# Đánh giá kết quả

Sau khi so sánh kết quả khi sử dụng hàm `inversion(A)` được cài đặt thủ công sử dụng thuật toán Gauss_Jordan và hàm `calculate_using_numpy(A)` sử dụng hàm có sẵn từ thư viện Numpy, ta rút ra được một số kết luận như sau:

1. Phương pháp thủ công:

Cách implement trên có độ phức tạp về thời gian là $O(n^3)$ cho việc nghịch đảo ma trận.

Với các ma trận có kích thước nhỏ, thời gian chạy là khá nhanh và không chênh lệch nhiều với thư viện (nhanh hơn thư viện). Tuy nhiên với các ma trận có kích thước lớn, phương pháp này tiêu tốn nhiều thời gian hơn và có thể không khả thi cho các ma trận với kích thước đặc biệt lớn.

2. Sử dụng thư viện NumPy

Hàm `inv()` sử dụng LU decomposition để tính toán ma trận nghịch đảo, từ đó tăng hiệu năng cũng như độ chính xác của kết quả đầu ra so với phương pháp thủ công đối với ma trận lớn hơn.