

**TRƯ**

**Ờ**

**NG Đ**

**Ạ**

**I H**

**Ọ**

**C KHOA H**

**Ọ**

**C T**

**Ự**

**NHIÊN ĐHQG**

**-**

**HCM**

**\***

**-------**

**\***

**KHOA CÔNG NGHỆ THÔNG TIN**

**ĐỒ ÁN 2 – MA TRẬN NGHỊCH ĐẢO**

**Môn Học: Toán ứng dụng & thống kê**

**Giảng viên:**

Nguyễn Trọng Hiến

Nguyễn Văn Quang Huy

**Sinh viên thực hiện:**

Nguyễn Tấn Phát 20127588



Nguyễn Đình Thúc

Võ Nam Thục Đoan

## Nội dung đồ án:

Sinh viên viết hàm inverse(A), trong đó

* + Input: A là ma trận vuông.
  + Output: Ma trận nghịch đảo của ma trận A ban đầu nếu có, trường hợp không có ma trận nghịch đảo sẽ hiện thông báo " Ma trận không khả nghịch ". Lưu ý sinh viên phải sử dụng thuật toán đã được hướng dẫn trong phần bài tập để tìm nghịch đảo( dùng ma trận (A|I) ). Sinh viên không được dùng các hàm có sẵn của các thư viện để tìm định thức hoặc ma trận nghịch đảo.

## Môi trường làm việc:

* Ngôn ngữ lập trình: Python
* Text Editer: Visual Studio Code
* Thư viện hỗ trợ: numpy

## Cơ sở và ý tưởng:

### Xây dựng class và các đối tượng

* Tên class: **MyMatrix** (Là một lớp về ma trận)
* **self.root**: ma trận input và có giá trị không thay đổi (dạng numpy.array)
* **self.matrix**: ma trận input giúp đỡ trong quá trình biến đổi (dạng numpy.array)
* **self.inv**: ma trận đơn vị, biến đổi về ma trận nghịch đảo của input (dạng numpy.array)
* **self.numRow**: số hàng của ma trận (dạng int)
* **self.numCol**: số cột của ma trận (dạng int)

### Ý tưởng giải quyết bài toán

Về lý thuyết:

Ảnh có chứa đồng hồ, thiết bị đo

Mô tả được tạo tự động

* Gộp ma trận A và ma trận đơn vị I thành ma trận [A|I]
* Biến đổi ma trận bằng các phép biến đổi sơ cấp trên dòng nhằm đưa về ma trận [I|B]
* Nếu đưa về [I|B] thành công thì B chính là ma trận nghịch đảo của A

Về thực hiện (code):

* Thay vì phải gộp 2 ma trận lại thì ta chỉ cần biến đổi ma trận A thành ma trận đơn vị một cách bình thường. Với điều khiến, bất kì sự thay đổi ở ma trận A thì cũng xảy ra ở ma trận đơn vị I.
* Nếu A không phải ma trận vuông => thông báo không có nghịch đảo và return None.
* Nếu A đã là ma trận đơn vị => Thông báo hoàn thành và return self.inv [I]
* Thực hiện Gauss\_ Jordan để đưa A về ma trận đơn vị
* Nếu thành công thì trả về ma trận self.inv (đã được biến đổi cùng với A)

## Các hàm hỗ trợ:

### swapRow(matrix, row1, row2)

* *Chức năng: Đổi 2 dòng của ma trận cho nhau*
* VD:
  + Input: [[1, 2],[3, 4]], 2, 1
  + Output: [[3, 4], [1, 2]

### mulRow(matrix, k: float, row)

* *Chức năng: Nhân một hàng của ma trận với hệ số khác 0.*
* VD:
  + Input: [[1, 2],[3, 4]], 2, 1
  + Output: [[1, 2], [6, 8]

### plusRow(matrix, rowI, k: float, rowJ)

* Chức năng: hàng I = hàng I + k \* hàng J
* VD:
  + Input: [[1, 2],[3, 4]], 0, 1, 1
  + Output: [[4, 6], [6, 8]

\

## Giải thích thuật toán inverse():

* Nếu A không phải ma trận vuông => thông báo không có nghịch đảo và return None.
* Nếu A đã là ma trận đơn vị => thông báo hoàn thành và return ma trận đơn vị.
* Dùng vòng for để duyệt từng dòng của ma trận self.matrix với chỉ số i:
  + Nếu có dòng đang xét có giá trị matrix[i][i] == 0:
    - Tìm dòng khác thay thế để matrix[i][i] != 0
    - Nếu không có dòng nào cả thì thông báo không ma trận nghịch đảo và return None.
    - Thay vì kiểm tra matrix[i][i] == 0 ? Thì kiểm tra abs(matrix[i][i]) < 0,0001
    - Vì quá trình máy tính toán thì có thể làm tròn, nên đôi khi chỉ ra xấp xỉ 0 ~ 2e^–16 chứ không chính xác là 0.
  + Chia dòng I của ma trận matrix[i][i] để giá trị của matrix[i][i] = 1. Và làm tương tự với ma trận đơn vị self.inv
    - Tuy nhiên ta sẽ thực hiện với ma trận self.inv trước rồi mới đến self.matrix
    - Vì cả 2 đều chia hệ số của self.matrix[i][i] nên nếu thực hiện với self.matrix trước thì hệ số trên sẽ bị thay dổi.
  + Dòng vòng for để duyệt các dòng của ma trận self.matrix một lần nữa với chỉ số j:
    - Nếu j != i
      * Dòng j = dòng j – (matrix[i][i] \* dòng i) đối với cả 2 ma trận self.inv và self.matrix.
      * Khi thực hiện phép bién đổi này, ta đang làm cho các giá trị nào không nằm trên đường chéo (self.matrix) đều về 0 để đưa về ma trận đơn vị.
* Return self.inv

### manySolution(self, free, i, n, arr\_solve, arr\_input)

* Chức năng: Tính toán nghiệm khi trường hợp “vô số nghiệm xảy ra”
* Các tham số truyền vào có chức năng như hệt phía trên:
  + free : Đếm số các nghiệm tự do hiện chưa có.
* Cũng sử dụng công thức
* Do nghiệm là các chuỗi str nên phải viết các hàm nhỏ hỗ trợ:
  + subStr(number): trừ number
  + dev(a, b): chia a cho b (a/b)
  + mul(a, b): nhân a với b (a\*b)

### Algorithm(self)

* Chức năng: Giải hệ phương trình tuyến tính bằng Gauss.
* Đưa về ma trận bậc thang.
* Kiểm tra xem kết quả rơi vào TH nào:
  + 1 nghiệm duy nhất
  + Vô số nghiệm
  + Vô nghiệm
* Trả về nghiệm phù hợp

## Kết quả chạy thử:

### Ảnh có chứa văn bản Mô tả được tạo tự động Ảnh có chứa văn bản, thiết bị điện tử, ảnh chụp màn hình Mô tả được tạo tự động Ảnh có chứa văn bản Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HẾT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*