BÁO CÁO ĐỒ ÁN 1

1 Các kí hiệu được sử dụng

- A (in hoa, chữ thường hoặc hoa): ma trận với m dòng và n cột. Với phần tử $A[i][j]$ bất kì trong ma trận, ta có chỉ số dòng và chỉ số cột sẽ nằm trong khoảng sau: $i \in [0, m-1]$, $j \in [0, n-1]$
- $A[i]$: tất cả phần tử thuộc dòng i trong ma trận A
- $\mathbf{x}$ (in đậm, chữ thường): vector x

2 Phép khử Gauss − Gauss_elimination

2.1 Ý tưởng thực hiện

Cách làm này sẽ hơi khác so với phép khử Gauss thông thường, ta đưa ma trận mở rộng A về ma trận bậc thang bằng cách sử dụng 3 phép biến đổi dòng sao cho tại mỗi cột chỉ có một giá trị là 1, các giá trị còn lại đều bằng 0 (hay còn gọi với tên khác là phép khử Gauss−Jordan).

Các bước thực hiện

- Duyệt từng dòng từ trên xuống dưới trong ma trận mở rộng A. Ta gọi chỉ số dòng mà thuật toán đang xét là i.
- Bước 1: Tìm cột p nhỏ nhất $\geq i$ sao cho từ $A[i][p]$ trở xuống ta được một vector khác với vector $\mathbf{0}$. Nếu không tồn tại cột p trong ma trận thoả yêu cầu trên thì ta kết thúc thuật toán khử Gauss. Ví dụ, xét ma trận A như sau.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 10 \\ 0 & 0 & 6 & 12 \end{bmatrix}$$

  Xét dòng 2 ($i = 1$) và $p = 1$. Từ $A[i][p] = A[1][1]$ trở xuống ta có $\mathbf{x} = (A[1][1], A[2][1], A[3][1]) = (0, 0, 0) = \mathbf{0}$. Với $p = 2$, $\mathbf{x} = (A[1][2], A[2][2], A[3][2]) = (0, 5, 6) \neq \mathbf{0}$. Như vậy $p = 2$ là giá trị nhỏ nhất thoả mãn bước 1.

- Bước 2: Đổi dòng i nếu $A[i][p] = 0$ với dòng j có $A[j][p] \neq 0$ mà $j > i$.

  Ví dụ ma trận A như trên, với $i = 1$ và $p = 2$, do $A[i][p] = A[1][2] = 0$ và $A[j][p] = A[2][2] = 5 \neq 0$ nên ta tiến hành đổi 2 dòng $A[1]$ và $A[2]$. Ma trận sau khi hoán đổi dòng sẽ có các giá trị như sau.

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 5 & 10 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 12 \end{bmatrix}$$

- Bước 3: Chia $A[i]$ cho $A[i][p]$ để phần tử *leader* trong dòng đó là 1. *Leader* của một dòng được hiểu là phần tử đầu tiên trong dòng đó có giá trị khác 0.

Với ma trận A sau khi hoán đổi dòng, với $i = 1$ và $p = 2$, ta lấy từng phần tử trong dòng i chia cho giá trị $A[i][p] = A[1][2] = 5 \Rightarrow$ Sau khi thực hiện phép chia thì $A[i] = A[1] = (0, 0, 1, 2)$. Ma trận lúc này trở thành

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 12 \end{bmatrix}$$

- Bước 4: $\forall j \in [0, m-1], i \neq j, A[j] = A[j] - A[j][p] * A[i]$

  Khi thực hiện xong bước 4, ma trận A có dạng

$$\begin{bmatrix} 1 & 2 & 0 & -2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

  Ta nhận xét được từ dòng 3 ($i = 2$) trở đi tất cả các dòng đều là vector **0** nên thuật toán sẽ trả ra ma trận bậc thang như kết quả trên.

2.2 Mô tả hàm

```
def Gauss_elimination(A):
    m = len(A)
    n = len(A[0])
    for i in range(m):
        cont = True
        p = i
        while p < n:
            for j in range(i, m):
                if A[j][p] != 0:
                    cont = False
                    break
            if cont == False: break
            p += 1
        if p == n: break
        A[i], A[j] = A[j], A[i]
        temp = A[i][p]
        for k in range(n):
            A[i][k] /= temp
        for j in range(0, m):
            if j != i:
                temp = A[j][p]
                for k in range(n):
                    A[j][k] -= temp * A[i][k]
```

- Ta lấy ra $m = len(A)$ là số lượng dòng và $n = len(A[0])$ là số lượng cột trong ma trận A.
- Như ý tưởng thực hiện được nêu ở phần trên, ta duyệt theo từng dòng trong khoảng $[0, m-1]$.
  - Tạo 2 biến: biến p như mô tả ở ý tưởng và biến *cont* để xem có giá trị p phù hợp hay chưa.

- Sau khi tìm được p, nếu $p = n$ thì từ dòng i trở xuống tất cả các giá trị đều là $0 \Rightarrow$ Kết thúc thuật toán.
- j là dòng đầu tiên mà $A[j][p] \neq 0$, đổi 2 dòng $A[i]$ và $A[j]$ (dòng 15) với nhau (nếu i và j bằng nhau thì dòng code này vẫn giữ nguyên ma trận mà không thay đổi dòng nào cả).
- 3 dòng tiếp theo (từ 16 đến 18) thực hiện công việc chia từng phần tử trong dòng $A[i]$ cho $A[i][p]$.
- Ở vòng lặp cuối cùng ta thực hiện bước 4: $\forall j \in [0, m-1], i \neq j, A[j] = A[j] - A[j][p] * A[i]$ để các phần tử trong cột p khác $A[i][p]$ đều là 0.

3 Phép thế – Back_substitution

3.1 Ý tưởng thực hiện

Sau khi đưa về ma trận bậc thang, ta dùng phép thế ngược lên để tìm nghiệm cho hệ phương trình (vô nghiệm, có duy nhất 1 nghiệm hoặc vô số nghiệm).

- Với trường hợp *vô nghiệm*, ma trận sẽ tồn tại 1 dòng có dạng $(0, 0, ..., 0, c)$ với $c \neq 0$.

Ta cũng rút ra được nhận xét sau đây: Nếu đưa về ma trận bậc thang rút gọn bằng phép khử Gauss−Jordan, xét dòng i ($i \in [0, m-1]$), nếu cột j (j đi từ 0 đến $n-2$, ta không xét $n-1$ vì nó là hệ số bên phải của hệ phương trình chứ không phải là hệ số của ẩn x_k nào đó) là vectơ $\mathbf{0}$ thì x_j là nghiệm tự do và không được biểu diễn bởi các nghiệm khác, nếu khác vectơ $\mathbf{0}$ ta có $x_j = A[i][n-1]$.

- Với trường hợp *vô số nghiệm*, ta xét biến k chạy từ $j+1$ đến $n-2$, nếu giá trị $A[i][k] \neq 0$ thì ta lấy $x_j \mathrel{-}= A[i][k] * x_k$ với x_k là nghiệm tự do.

3.2 Mô tả hàm

```
1  def back_substitution(A):
2      m = len(A)
3      n = len(A[0])
4      p = []
5      for i in range(n - 1): p.append(False)
6
7      for i in range(m - 1, -1, -1):
8          noSol = False
9          for j in range(n - 1):
10             if a[i][j] != 0: break
11         if j == n - 2 and a[i][j + 1] != 0 and a[i][j] == 0:
12             noSol = True
13             break
14
15     if noSol == True: print('No solution')
16     else:
17         run = 1
18         for i in range(m):
19             while run <= n - 1 and a[i][run - 1] == 0:
20                 print('x' + str(run) + ' = x' + str(run))
21                 p[run - 1] = True
22                 run += 1
23             if run >= n: break
```

```
24              line = 'x' + str(run) + ' = ' + str(a[i][n - 1])
25              p[run - 1] = True
26              for j in range(run, n - 1):
27                  if a[i][j] != 0:
28                      if a[i][j] > 0: line += ' - ' + str(a[i][j]) + 'x' + str(j + 1)
29                      else: line += ' + ' + str(-a[i][j]) + 'x' + str(j + 1)
30              run += 1
31              print(line)
32          for i in range(len(p)):
33              if p[i] == False: print('x' + str(i + 1) + ' = x' + str(i + 1))
```

- Ta khởi tạo số dòng m và số cột n của ma trận, đồng thời dùng mảng p để đánh dấu xem nghiệm đã được in ra hay chưa. Nếu kết thúc thuật toán, mảng p vẫn còn vị trí có giá trị là $False$ thì in nghiệm đó ra (cũng là nghiệm tự do của hệ phương trình).
- Từ dòng 7 đến dòng 15 ta kiểm tra trường hợp *vô nghiệm*.
  - Ta tiến hành duyệt theo dòng i từ dưới lên để có thể tìm dòng làm hệ *vô nghiệm* nhanh hơn so với duyệt từ trên xuống trong đa số trường hợp.
  - Kiểm tra từ cột 0 đến cột $n-2$. Nếu sau vòng lặp $j = n-2$ thì có nghĩa các giá trị từ $A[i][0]$ đến $A[i][n-3]$ đều là 0. Ta xét 2 giá trị cuối cùng trong dòng đó, nếu $A[i][j+1] = A[i][n-1] \neq 0$ và $A[i][j] = A[i][n-2] = 0$ thì dòng đó có dạng $(0, 0, ..., 0, c)$ với $c \neq 0 \Rightarrow$ Kết luận hệ phương trình *vô nghiệm* và kết thúc thuật toán.
- Từ dòng 17 đến 31 ta xử lý về trường hợp *có 1 nghiệm duy nhất* và trường hợp *vô số nghiệm*.
  - Biến *run* thể hiện cho việc đã duyệt tới cột nào rồi (hay có thể hiểu là đã đi tới nghiệm nào rồi).
  - Ta tiến hành duyệt theo từng dòng i. Nếu $A[i][run] = 0 \Rightarrow x_{run}$ là nghiệm tự do $\Rightarrow$ In nghiệm tự do đó ra và tăng biến *run* lên 1 đơn vị.
  - Nếu $A[i][run] \neq 0$ thì $x_{run} = A[i][n-1]$. Ta tiếp tục duyệt cột j từ *run* đến $n-2$ ($n-1$ là hệ số bên phải của hệ chứ không phải hệ số ẩn nên không xét), nếu $A[i][j] \neq 0$ thì $x_{run} \mathrel{-}= A[i][j] * x_j$
    - Với trường hợp *có 1 nghiệm duy nhất*, ma trận hệ số ẩn sẽ trông giống ma trận đơn vị I_n.
- Với 2 dòng cuối (32 và 33), ta kiểm tra nếu có nghiệm chưa in ra thì tiến hành in ra màn hình.