Môn học: Thực hành toán ứng dụng thống kê

Báo cáo lab 1

Họ và tên: Bùi Thị Thanh Ngân

MSSV: 21120505

Bài 1: chuyển ma trận mở rộng của hệ phương trình sang ma trận dạng bậc thang

Hàm Gauss_elimination(A):

Ý tưởng thực hiện:

- Dùng biến i lần lượt duyệt qua các hàng của ma trận A:
  - Kiểm tra nếu `A[i][i] = 0` thì tìm hàng `j > i` sao cho A[j][i] khác 0 và đổi chỗ 2 hàng i và j, nếu trong cùng cột i mà không tìm được hàng j thỏa A[j][i] != 0 thì chuyển qua cột tiếp theo `c = i+1`, nếu đã xét đến cột cuối cùng mà không tìm được hàng j thỏa A[j][i] != 0 thì `dừng vòng lặp`.
  - Nếu A[i][c] != 0 thì thực hiện `chia các phần tử cùng hàng cho A[i][c]` để đưa A[i][c] về 1
  - Duyệt qua các hàng `k != i` và thực hiện trừ các phần tử của hàng k cho A[k][c]*A[i][c] để `đưa các phần tử cùng cột về 0`

Mô tả hàm:

- Tham số đầu vào: ma trận mở rộng A
- Hàm trả về ma trận đã được chuyển về dạng bậc thang/ bậc thang rút gọn.
- Từ ý tưởng, xây dựng hàm Gauss_elimination(A) đã có comment giải thích khá chi tiết như sau:

```
def Gauss_elimination(A) :
    # số hàng của ma trận A
    nrow = len(A)
    # số cột của ma trận A
    ncol = len(A[0])

    # duyệt qua các hàng của ma trận A
    for r in range(nrow):
        # nếu phần tử A[r][c] trong hàng r là 0
        c = r
        while A[r][c] == 0:
            # tìm trong cột đó, phần tử khác 0, nếu có thì đổi hàng
            for i in range(r+1, nrow):
                if A[i][c] != 0:
                    A[r], A[i] = A[i], A[r]
                    break
            # nếu không có phần tử khác 0 trong cột đó thì xét qua cột tiếp theo
            if A[r][c] == 0:
```

```
                c += 1
            if c == ncol:
                break
        # nếu hàng đó không có phần tử khác 0 thì dừng
        if c == ncol:
            break
        # nếu phần tử đó khác 0 thì chia hết các phần tử cùng hàng cho nó
        A[r] = A[r] / A[r][c]
        #  đưa các phần tử cùng cột đó về 0
        for k in range(0, nrow):
            if k != r:
                A[k] = A[k] - A[k][c] * A[r]

    return A
```

Bài 2: giải hệ phương trình từ ma trận bậc thang tìm được ở bài 1

Hàm back_substitution(A):

Ý tưởng thực hiện:

- Xét các trường hợp:
- Nếu số hàng có hệ số A[i][i] khác 0 của ma trận bằng số ẩn của hệ phương trình thì hệ phương trình có nghiệm duy nhất.
- Nếu tồn tại phương trình dạng `0x = b` với b khác 0 thì hệ phương trình vô nghiệm.
- Ngược lại, hệ phương trình có vô số nghiệm.

Mô tả hàm:

- Tham số đầu vào: ma trận bậc thang A có được từ hàm Gauss_elimination(A) và `fout`
- Ở đây, em sử dụng thêm tham số fout để tiện ghi kết quả ra file `output.txt`
- Kết quả nghiệm được ghi ra file output phía dưới mỗi ma trận
- Nếu có nghiệm duy nhất thì in ra file : `Phuong trinh co nghiem duy nhat: (x ; y ; z ; t)`
- Trong đó x, y, z, t là các nghiệm của hệ phương trình
- Nếu vô nghiệm thì in ra file: `Phuong trinh vo nghiem !`
- Trường hợp vô số nghiệm sẽ in ra file: `Phuong trinh co vo so nghiem dang: (ma + nb ; a - kb ; b ; a)`
- Trong đó các nghiệm được biểu diễn thông qua ẩn tham số a, b, c, d, ... và các ẩn khác được biểu diễn qua các tham số đó
- Các giá trị ghi vào file sẽ được làm tròn 1 chữ số phần thập phân nếu là số thực và ghi ra số nguyên nếu là số nguyên để dễ nhìn hơn.
- Trong trường hợp vô số nghiệm:

- Đầu tiên sẽ đếm số ẩn của hệ phương trình
- Tính số ẩn cần đặt tham số tự do
- Tạo 1 list chứa các ẩn tham số tự do bắt đầu từ kí tự `a`

```
    listSolutions = []    # danh sách các nghiệm
    for i in range(count):
        if A[i][i] != 0:
            # đưa vào xâu rỗng để xử lý sau
            listSolutions.append("")
        else:
            # nếu A[i][i] = 0 thì đặt tham số cho ẩn đó và đưa vào danh sách
nghiệm
            listSolutions.append(listPara.pop(0))
            # tìm phần tử khác 0 đầu tiên trong cùng hàng nếu có
            for j in range(i+1, nc - 1):
                if A[i][j] != 0:
                    # đưa vào xâu rỗng để xử lý sau
                    listSolutions.append("")
                    break
    # nếu còn ẩn tham số thì đưa vào danh sách nghiệm
    for i in range(len(listPara)):
        listSolutions.append(listPara.pop(0))
```

- Danh sách nghiệm sẽ được lưu dạng chuỗi trong listSolutions
- Lần lượt duyệt qua các hàng khác 0 của ma trận và kiểm tra:
    - Nếu A[i][i] = 0 thì đặt tham số cho ẩn đó và đưa vào danh sách nghiệm, sau đó tìm phần tử khác 0 đầu tiên trong cùng hàng, nếu có thì đưa vào xâu rỗng để xử lý sau đó và chuyển sang hàng tiếp theo
    - Ngược lại thì đẩy vào chuỗi rỗng để xử lý sau vì nó phụ thuộc vào các ẩn khác
- Tiến hành duyệt từ cuối lên đầu của danh sách nghiệm
- Nếu là chuỗi rỗng thì thực hiện xử lý
    - `Ax = b`, nếu giá trị bên phía b khác 0 thì không cần chèn vào chuỗi nghiệm, ngược lại thì chèn vào chuỗi nghiệm giá trị đã làm tròn
    - Phương trình một hàng có dạng `x + ay + bz = d` : Duyệt qua các phần tử cùng hàng phía sau để trừ đi hệ số nhân với giá trị nghiệm của nó
- Một vài xử lý nhỏ:
    - Nếu `1a` thì chỉ ghi ra `a`, nếu `-1a` thì chỉ ghi ra `-a`.
    - Nếu `1(ax + by)` thì chỉ ghi ra `(ax + by)`, nếu `-1(ax + by)` thì chỉ ghi ra `-(ax + by)`.
    - Tránh trường hợp ghi ra + -x hoặc - +x.

Input, Output:

- `Input`: file input.txt

- Dòng đầu tiên chứa số lượng ma trận cần đọc
- Dòng tiếp theo chứa size của ma trận m x n: số hàng m và số cột n
- m dòng tiếp theo mỗi dòng chứa n phần tử của ma trận
- Tương tự cho các ma trận tiếp theo

- `Output`: file output.txt

- Dòng đầu tiên chứa số lượng ma trận đã giải được
- Dòng tiếp theo chứa size của ma trận m x n: số hàng m và số cột n
- m dòng tiếp theo mỗi dòng chứa n phần tử của ma trận
- Dòng tiếp theo sẽ là nghiệm của hệ phương trình từ ma trận trên đó
- Tương tự cho các ma trận tiếp theo