VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

BÁO CÁO

CHỦ ĐỀ 8: PHƯƠNG PHÁP GAUSS

Sinh viên thực hành: Đặng Sỹ Tiến 20200537

Giáo viên hướng dẫn: TS. Hà Thị Ngọc Yến

Bộ môn: Giải tích số

Lời nói đầu

Cuộc cách mạng công nghiệp 4.0 xuất hiện, đánh dấu một chương mới: kỷ nguyên của công nghệ thông tin và dữ liệu lớn. Tại thời điểm này, những cỗ máy với những công cụ toán học cùng các ngành khoa học phụ trợ khác, đã có thể độc lập "tư duy", tự mình tìm kiếm những thông tin có giá trị từ một khối thông tin khổng lồ, hỗn độn. Những suy nghĩ về học máy, về trí tuệ nhân tạo giờ đây đã xuất hiện, chiếm sóng trong hầu hết hoạt động của cuộc sống. Từ những thứ đơn giản như máy bán nước, cây ATM, mua sắm online tới những quy trình phức tạp như phẫu thuật nội soi, quản lý xuất nhập cảnh, … tất cả đều nhờ sử dụng trí tuệ nhân tạo như là một cách thức để nâng cao tối đa tính hiệu quả và có phần nhẹ nhàng hơn trong cách sử dụng.

Và câu hỏi đặt ra là ẩn sau những khái niệm trừu tượng ấy là gì? Câu trả lời là những công thức toán học. Những mô hình học máy phức tạp kia, thực chất chỉ là những lần lặp với mục tiêu tối giản hàm mục tiêu, … Sau cùng, mọi thứ chúng ta đang có ở đây cũng chỉ là những kiến thức toán học đã được xây dựng từ trước. Toán học cũng có nhiều lĩnh vực. Dù vậy, có thể khẳng định rằng, Đại số tuyến tính đóng một vai trò rất quan trọng trong Khoa học máy tính. Từ việc xử lý ảnh, dựng hình đến việc dựng các mô hình tensors liên kết với nhau tạo thành mạng neuron, … tất cả chúng về bản chất đều là những thao tác trên ma trận số. Chính vì điều đó mà Đại số tuyến tính là một trong những học phần cơ bản đầu tiên sinh viên phải vượt qua nếu theo học các chương trình đào tạo liên quan đến tin học.

Mặt khác, khái quát việc nghiên cứu Đại số tuyến tính là gói gọn trong việc xem xét những hệ phương trình tuyến tính. Điều đó đã có từ rất lâu đời: Ở Trung Hoa đã nhắc đến nó trong tập The Nine Chapters on the Mathematical Art. Hay toán học cận đại cũng ghi nhận những nghiên cứu của Decartes hay Leibniz. Chính điều đó đã dẫn tôi đi tìm hiểu và trình bày việc giải hệ phương tình tuyến tính bằng việc sử dụng phương pháp khử Gauss nhằm giải ra nghiệm đúng của hệ phương trình. Xin cảm ơn TS Hà Thị Ngọc Yến đã giúp tôi chỉnh sửa và hoàn thiện báo cáo này! Đồng thời trong báo cáo ắt sẽ có nhiều chỗ sai sót, mong bạn đọc thông cảm và góp ý. Tôi xin chân thành cảm ơn!

Mục lục

Hệ phương trình tuyến tính và khái niệm cơ bản

3.1. Hệ phương trình tuyến tính (m phương trình n ẩn)**:**

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases} \qquad (1.1)$$

Trong đó, các hệ số aij, bi thuộc trường K. Trong phạm vi báo cáo ta coi K ≡ R. Từ đây ta có các định nghĩa sau:

a. Các hệ số của hệ phương trình lập thành một ma trận A, gọi là **ma trận của các hệ số**.

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

b. Ma trận $\overline{A}$ thu được từ ma trận A bằng cách ghép thêm cột hạng tự do gọi là **ma trận mở rộng của hệ**:

$$\overline{A} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m2} & \ldots & a_{mn} & b_m \end{pmatrix}$$

c. Ma trận B thu được từ cột hệ số tự do, gọi là **ma trận hệ số tự do của hệ**.

$$B = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{pmatrix}$$

Như vậy, nếu gọi X là ma trận cột biểu diễn các ẩn của hệ phương trình, thì hệ phương trình đấy có thể viết dưới dạng:

$$A.X = B \quad (1.2)$$

Từ đây về sau, ta sử dụng dạng trên để biểu diễn hệ phương trình là chủ yếu.

3.2. Nghiệm của hệ phương trình tuyến tính:

Mỗi nghiệm của hệ phương trình (1.2) là một bộ sắp thứ tự $(c_1,c_2,...c_n)$ thỏa mãn tất cả các phương trình của hệ (1.2) được thỏa mãn khi thay các ẩn x_i tương ứng bởi c_i

Một hệ phương trình có thể có duy nhất nghiệm, vô số nghiệm hoặc vô nghiệm.

3.3. Biến đổi tương đương:

Hai hệ phương trình được gọi là tương đương nếu hai hệ phương trình đó có cùng không gian nghiệm.

Các phép biến đổi sơ cấp của hệ phương trình là các phép biến đổi:

a. Nhân một phương trình nào đó của hệ với một số khác 0.

b. Cộng vào một phương trình nào đó của hệ với một phương trình khác đã nhân với một số khác 0.

c. Đổi chỗ hai phương trình của hệ.

Từ đây, ta có thể rút ra một số nhận xét sau:

- Các phép biến đổi sơ cấp của hệ phương trình tương ứng với các phép biến đổi trên hàng của ma trận mở rộng của hệ.

- Khi thực hiện các phép biến đổi sơ cấp, ta thu được một hệ phương trình mới tương đương với hệ phương trình ban đầu.

3.4. Các kết quả quan trọng:

a. Hệ phương trình Cramer - Định lý Cramer

Một hệ phương trình Cramer là một hệ phương trình mà số ẩn bằng số phương trình và định thức của ma trận hệ số khác không.

Từ đây, ta có thể phát biểu định lý Cramer.

Định lý 1.1 (Cramer): Hệ phương trình Cramer luôn có duy nhất nghiệm.

Chứng minh:

Xét hệ phương trình Cramer:

$$A.X=B$$

Do đây là hệ phương trình Cramer nên ta có:

$$\det(A) \neq 0$$

Như vậy tồn tại ma trận nghịch đảo của A là A^{-1}.Nhân cả hai vế của hệ phương trình ban đầu với A^{-1} ta được:

$$X = A^{-1}.B$$

Như vậy hệ phương trình đã cho có nghiệm duy nhất là:

$$X = A^{-1}.B$$

b. Định lý Kronecker – Capelli và các hệ quả

Trong định lý này, ta sẽ nêu ra điều kiện để một hệ phương trình tổng quát có nghiệm, cũng như các hệ quả xung quanh.

Định lý 1.2 (Kronecker – Capelli): Hệ phương trình tuyến tính:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

Có nghiệm khi và chỉ khi hạng của ma trận hệ số bằng hạng của ma trận mở rộng.

Từ đây, ta rút ra hai hệ quả quan trọng sau:

Hệ quả 1.2.1: Nếu hệ phương trình (1.1) với n ẩn có $r_A = r_{\bar{A}} = n$ thì hệ có nghiệm duy nhất.

Chứng minh:

Giả sử hệ có $r_A = r_{\bar{A}} = n$. Khi đó tồn tại định thức con cơ sở của của M (định thức con khác 0), cả A và $\bar{A}$. Bây giờ chúng ta sẽ chứng minh rằng mỗi dòng của $\bar{A}$ khác với n dòng đầu tiên là tổ hợp tuyến tính của n dòng này. Khi đó hệ phương trình đã cho tương đương với hệ phương trình:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ ..... \\ A_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \end{cases}$$

Do M ≠ 0 nên hệ này là một hệ Cramer, vì vậy nó có nghiệm duy nhất theo định lý Cramer.

Hệ quả 1.2.2: Nếu $r_A = r_{\bar{A}} < n$ thì hệ có vô số nghiệm phụ thuộc vào n - r_A tham số.

Chứng minh:

Giả sử $r_A = r_{\bar{A}} < n$. Kí hiệu M là định thức con cơ sở của A và $\bar{A}$. Hơn nữa giả thiết rằng định thức này nằm ở góc phía trên bên trái của A. Do đó:

$$M = \begin{vmatrix} a_{11}\ a_{12} \ldots a_{1r} \\ a_{21}\ a_{22} \ldots a_{2r} \\ \ldots\ldots\ldots \\ a_{r1}\ a_{r2} \ldots\ a_{r} \end{vmatrix}$$

Bởi vì mỗi dòng thứ i của $\bar{A}$, là tổ hợp tuyến tính của r dòng đầu tiên nên hệ đã cho tương đương với:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 - a_{1(r+1)}x_{(r+1)} - \cdots - a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 - a_{2(r+1)}x_{(r+1)} - \cdots - a_{2n}x_n \quad (1.3) \\ \qquad\qquad \ldots\ldots\ldots\ldots \\ a_{r1}x_1 + a_{r2}x_2 + \cdots + a_{rr}x_r = b_r - a_{r(r+1)}x_{(r+1)} - \cdots - a_{rn}x_n \end{cases}$$

Đây là một hệ phương trình Cramer r phương trình r ẩn $x_1, x_2, \ldots, x_r$. Với mỗi bộ giá trị $(c_{r+1}, \ldots, c_n)$ của các ẩn $(x_{r+1}, \ldots, x_n)$ ta được một nghiệm của (1.3) là $(c_1, \ldots, c_r)$. Khi đó $(c_1, c_2, \ldots, c_r, c_{r+1}, \ldots, c_n)$ là nghiệm của (1.1).

***Nhận xét:** Những hệ quả của định lý Kronecker Capelli rất quan trọng, nó là cơ sở giúp cho chúng ta thực hiện phương pháp khử Gauss khi cài đặt phương pháp giải quyết những trường hợp phương trình vô nghiệm hoặc có vô số nghiệm.

Phương pháp Gauss

4.1. Ý tưởng phương pháp:

Phương pháp Gauss (Gauss Elimination) là một phương pháp giải đúng hệ phương trình tuyến tính. Nó sử dụng những phép biến đổi cơ bản, biến ma trận mở rộng của hệ phương trình tuyến tính về dạng ma trận hình thang (Row echelon form), sau đó sẽ giải lần lượt từng phương trình của hệ theo thứ tự từ dưới lên, khi mà các phương trình ở dưới sẽ đơn giản hơn (ít ẩn hơn), và khi đó việc tìm từng giá trị của ẩn thỏa mãn hệ phương trình sẽ dễ dàng hơn.

4.2. Nội dung phương pháp:

Như đã nói ở trên, phương pháp Gauss là việc thực hiện những phép biến đổi tương đương trên ma trận mở rộng của hệ phương trình tuyến tính, hoặc phép thế để giải hệ phương trình này. Và ở đây, tôi xin được phép trình bày phương pháp Gauss cho ma trận cấp m x n

*Cho ma trận

$$\begin{cases} E_1: a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ E_2: a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ ..... \\ E_n: a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases} \quad (4.1)$$

- Ta tách hệ phương trình trên thành dạng ma trận:

$$A.X = B$$

Trong đó:

$$A = \begin{bmatrix} a11 & a12 & \dots & a1n \\ a21 & a22 & \dots & a2n \\ \vdots & \vdots & \ddots & \vdots \\ am1 & am2 & \dots & amn \end{bmatrix}; \quad X = \begin{bmatrix} x1 \\ x2 \\ \vdots \\ xn \end{bmatrix}; \quad B = \begin{bmatrix} b1 \\ b2 \\ \vdots \\ bm \end{bmatrix}$$

- Từ các ma trận trên, ta xây dựng ma trận bổ sung:

$$\bar{A} = \left[\begin{array}{cccc|c} a11 & a12 & \dots & a1n & b1 \\ a21 & a22 & \dots & a2n & b2 \\ \vdots & \vdots & \ddots & \dots & \vdots \\ am1 & am2 & \dots & amn & bm \end{array}\right] \quad (4.2)$$

- Tiếp theo, chúng ta sẽ thực hiện phương pháp Gauss theo 2 quá trình.

  +) Quy trình thuận:

  Giả sử $a_{11} \neq 0$, ta sẽ thực hiện thao tác sau:

  $$((Ej) - \frac{aji}{aii} \times (Ei)) \rightarrow (Ej) \text{ với mỗi j = i + 1, i + 2, ... , n}$$

  Các thao tác này sẽ khử hệ số của biến x_i trong mỗi hàng dưới hàng thứ i với mỗi i = 1,2, ..., m − 1. Ma trận cuối cùng sẽ có dạng như sau:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = a_{1.n+1}$$

$$a_{22}x_2 + \cdots + a_{2n}x_n = a_{2.n+1} \quad (4.3)$$

$$\dots\dots\dots\dots$$

$$a_{\mathrm{m}n}x_n = a_{\mathrm{m}.n+1}$$

Trong đó các phần tử ở cột n+1 tương ứng với $b_1, b_2, \dots, b_m$.

+) Quy trình nghịch:

Từ hệ (4.3) giải ra ta được x_n; x_{n-1}; ... x_1 bằng phép thế dần và kết quả đó cho ta nghiệm của hệ phương trình (4.1).

Một số ví dụ:

VD1: Giải hpt:

$$\begin{cases} x + 2y + z - t = 1 \\ 2x + 3y - 5z + t = 2 \\ 3x + 5y - 4z = 3 \end{cases}$$

Ta tạo ra ma trận bổ sung $\bar{A}$:

$$\left(\begin{array}{cccc|c} 1 & 2 & 1 & -1 & 1 \\ 2 & 3 & -5 & 1 & 2 \\ 3 & 5 & -4 & 0 & 3 \end{array}\right)$$

Biến đổi về ma trận bậc thang $\bar{A}'$

$$\left(\begin{array}{cccc|c} 1 & 2 & 1 & -1 & 1 \\ 0 & -1 & -7 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Ta lần lượt giải ra được:

$$\begin{cases} x = 1 + 13t_1 - 5t_2 \\ y = -7t_1 + 3t_2 \\ z = t_1 \\ t = t_2 \end{cases}$$

VD2: Giải hpt:

$$\begin{cases} x + y + z = 3 \\ 2x - y + 2z = -3 \\ x - 3y - 3z = -5 \end{cases}$$

Ta tạo ra ma trận bổ sung $\bar{A}$:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 3 \\ 2 & -1 & 2 & -3 \\ 1 & -3 & -3 & -5 \end{array}\right)$$

Biến đổi về ma trận bậc thang $\bar{A}'$

$$\left(\begin{array}{ccc|c} 1 & 1 & -1 & 1 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right)$$

Ta lần lượt giải ra được:

$$\begin{cases} x = 1 \\ y = 3 \\ z = -1 \end{cases}$$

4.3.Thuật toán và code:

+) **Quy trình thuận:** Biến đổi ma trận về dạng bậc thang:

Cho i = 1; j = 1

Bước 1: Tìm phần tử khác 0 đầu tiên

- Lặp liên tục đến khi $a_{[i][j]}$ khác 0 thì dừng:

+ Nếu $a_{[i][j]} = 0$ thì tăng i sau đó kiểm tra nếu i = m+1 thì cho i = 1 tăng j và lúc này nếu j = n+2 thì dừng vòng lặp (đã xét hết ma trận mà vẫn là số 0).

- Sau khi dừng vòng lặp thì có được vị trí đầu tiên của ma trận khác 0 ($a_{[i][j]} \neq 0$ với j bé nhất, i bé nhất)

- Nếu $i \neq 1$ thì đảo hàng i và 1.

```
// Biến đổi về ma trận bậc thang
void matranbacthang() {
    int i = 1, j = 1;
    // Đi tìm phần tử khác 0 đầu tiên (đi từ trên xuống dưới, trái sang phải)
    while (ISZERO(a[i][j])) {
        i++;
        if (i == m + 1) {
            i = 1; j++;
            if (j == n + 2) {
                countzeros();
                return;
            } // Duyệt đến phần tử cuối cùng mà vẫn chưa thấy thì ma trận A là ma trận 0
        }
    }
    cout << "Vi tri dau tien khac 0: " << i << " " << j << endl;
    if (i != 1) daohang(i, 1);
```

Bước 2: Bắt đầu lặp từ cột thứ j vừa tìm được ở trên

- Vòng lặp for từ j tìm được ở bước 1, tăng j đến n+1:

+ Vòng lặp với hàng i, i = 2 đến i = m, nếu $a_{[i][j]}$ khác 0 thì bắt đầu tìm hàng k để khử hàng hiện tại:

1. Khởi tạo một biến flag = false
2. vòng lặp for k = i – 1 đến k = 1:

Nếu $a_{[k][j]}$ khác 0 và $a_{[k][0]}$ = j – 1 thì gán flag = true và thoát vòng lặp.

3. Nếu flag = true thì:

a. gán ratio = $a_{[i][j]}$ / $a_{[k][j]}$

b. Chạy vòng lặp for z = j đến z = n+1, gán $a_{[i][z]} = a_{[i][z]} – ratio*a_{[k][z]}$

4. Sau khi khử xong cột, sắp xếp lại hàng thành dạng bậc thang.

// In ma trận sau mỗi lần khử cột và sau khi biến đổi.

```
// Bắt đầu lặp từ cột thứ j vừa tìm được ở trên đến n
   for (; j <= n + 1; j++) {
       for (i = 2; i <= m; i++) {
           // Nếu a[i][j] khác 0 ta bắt đầu khử
           if (!ISZERO(a[i][j])) {
               int k; // Tìm hàng k để a[k][j] khác 0 và a[k][l] = 0 với mọi l<j
               bool flag = false;
               for (k = i - 1; k >= 1; k--) { // Tìm từ hàng i-1 trở lên
                   if (!ISZERO(a[k][j]) && a[k][0] == j - 1) {
                       flag = true; // Đã có hàng k thỏa mãn
                       break;
                   }
               }
               if (!flag) continue; //Nếu không tìm thấy k ta tiếp tục tăng i

               // Lấy hàng i trừ đi aij/akj lần hàng k và gán cho hàng i
               double ratio = a[i][j] / a[k][j];
               for (int z = j; z <= n + 1; z++) a[i][z] = a[i][z] - ratio * a[k][z];
               // inmatran();
```

- **Chú ý:** Ở đây ta có sử dụng gói sắp xếp:

  Gói sắp xếp: (sắp xếp từ hàng i trở xuống)

  updatezeros(i): (cập nhật lại số số 0 ở đầu hang kể từ hàng i trở xuống)

  Sau đó bắt đầu sắp xếp

  Vòng lặp for từ z = i đến z < m

  Gán MIN = z,

vòng lặp for từ k = z + 1 đến k ≤ m, nếu $a_{[k][0]} < a_{[MIN][0]}$ thì cho MIN = k, sau khi tìm được hàng có số số 0 nhỏ nhất là hàng MIN đổi chỗ hàng $a_{[MIN]}$ với hàng $a_{[z]}$

```
// Sắp xếp lại thứ tự các hàng từ hàng i theo tăng dần số số 0 ở đầu bằng sắp xếp lựa
chọn
void sapxeplaihang(int i = 1) {
    int MIN;
    countzeros(i); // Cập nhật lại số số 0
    for (int z = i; z < m; z++) {
        MIN = z;
        for (int k = z + 1; k <= m; k++) if (a[k][0] < a[MIN][0]) MIN = k;
        daohang(MIN, z);
    }
```

+) **<u>Quy trình nghịch:</u>**

<u>Phần 1:</u> Xác định loại nghiệm

Cho rank A = rank $\bar{A}$ = m

Vòng lặp for từ i = m đến 1, nếu $a_{[i][0]}$ = n (hàng i full 0) thì:

Giảm rank A, nếu $a_{[i][n+1]}$ = 0 thì giảm rank $\bar{A}$

ngược lại thì hàng i có phần tử khác 0, dừng vòng lặp và ta có rank A.

Nếu rank A = rank $\bar{A}$ < n là Vô số nghiệm

Nếu rank A = rank $\bar{A}$ = n là Nghiệm duy nhất

Còn lại là Vô nghiệm. → In ra "vô nghiệm".

```
// Xác định hệ có nghiệm duy nhất, vô số nghiệm hay vô nghiệm
int xacdinhloainghiem() {
    int rankA = m, rankAn = m; // rank A va rank A ngang
    for (int i = m; i >= 1; i--) {
        if (a[i][0] == n) {    // Số số 0 ở hàng i == n tức là hàng full 0
```

```
            rankA--;
            if (ISZERO(a[i][n + 1])) rankAn--; // Giá trị ở cột bổ sung
        }
        else break;
    }

    if (rankA == rankAn && rankA < n)  return rankA; // Vô số nghiệm
    if (rankA == rankAn && rankA == n) return -1;    // Nghiệm duy nhất
    return -2; // Vô nghiệm
```

Phần 2: Tìm ra nghiệm (Hệ có nghiệm)

- Gói kiểm tra nghiệm trong TH hệ có nghiệm duy nhất:

  + Nghiệm duy nhất:

  Tạo mảng x lưu nghiệm

  Vòng lặp for từ i = n đến i = 1

  $x_{[i]} = a_{[i][n+1]}$ (hệ số ở cột bổ sung)

  Vòng lặp for từ j = n đến j > i

  Thì $x_{[i]} = x_{[i]} - x_{[j]} * a_{[i][j]}$

  Gán $x_{[i]} = x_{[i]} / a_{[i][i]}$

  In ra nghiệm duy nhất.

```
// Kiểm tra nghiệm trong trường hợp hệ có nghiệm duy nhất
bool kiemtra(double *x, int offset = 0, bool param = false) {
    ifstream fi(checkfile);
    fi >> m >> n;

    for (int i = 0; i < m; i++) {
        double tmp = 0.0, aij = 0.0;
        for (int j = 0; j < n; j++) {
            fi >> aij;
            tmp += aij * x[j + 1 + offset];
        }
        fi >> aij;
        if ((param && !ISZERO(tmp)) || (!param && fabs(tmp - aij) > 1e-6)) return false;
```

```
    }
    fi.close();
    return true;
}

// Giải nghiệm trong trường hợp có nghiệm duy nhất
void giainghiemduynhat() {
    double *x = new double[n + 1]();
    for (int i = n; i >= 1; i--) {
        x[i] = a[i][n + 1];
        for (int j = n; j > i; j--) x[i] -= x[j] * a[i][j];
        x[i] /= a[i][i];
    }

    cout << "He co nghiem duy nhat" << endl;
    for (int i = 1; i <= n; i++) cout << "x_" << i << " = " << x[i] << endl;
    if (kiemtra(x)) cout << endl << "---> True" << endl;
    else            cout << endl << "---> False" << endl;
};
```

+ Vô số nghiệm:

- Tạo một mảng 2 chiều lưu nghiệm x mỗi dòng là 1 vector nghiệm

Vòng lặp for từ i = 0 đến i < n thì $x_{[i+1][i]} = 1$

- Biểu diễn ẩn này qua ẩn khác:

Cho i = rank A, bắt đầu chạy từ hàng i đến hàng 1

cho j = $a_{[i][0]}$, k = 0, $a_{ij} = a_{[i][j+1]}$

Gán vector nghiệm x_j = vector 0

$x_{[0][j]} = a_{[i][n+1]} / a_{ij}$

Khi $k \leq n$, nếu k khác j+1 thì $x_{[k][j]} = x_{[j][k]} - a_{[i][k]} / a_{ij}$, tăng k.

– Thế ẩn này vào ẩn kia (x_i biểu diễn qua x_j với i < j)

Vòng lặp for từ i = 0 đến i < n,

Vòng lặp for từ j = i +2 đến j ≤ n,

nếu $x_{[j][i]}$ khác 0 thì cho tmp = $x_{[j][i]}$,

vòng lặp for từ z = 0 đến z = n (thế x_j vào x_i)

$$x_{[z][i]} = x_{[z][i]} + tmp*x_{[z][j-1]}$$

$$x_{[j][i]} = x_{[j][i]} - tmp$$

Cuối cùng thu được các vector nghiệm

Sau đó in ra nghiệm

Gói kiểm tra lại nghiệm.

```
// Giải nghiệm trong trường hợp vô số nghiệm
void giainghiemtongquat(int rankA) {
    double **x;

    // Tạo một ma trận lưu nghiệm
    x = new double *[n+1];
    for (int i = 0; i <= n; i++) x[i] = new double[n]();
    for (int i = 0; i < n; i++)  x[i + 1][i] = 1;
    cout << "Rank A: " << rankA << endl;

    // Biểu diễn ẩn này qua ẩn khác (x1 = 0 -2x2 - 3x3 - 3.5x4)
    int i = rankA;
    while (i >= 1) {
        int j = a[i][0], k = 0;
        double aij = a[i][j + 1];
        for (int z = 0; z <= n; z++) x[z][j] = 0;
        x[0][j] = a[i][n + 1] / aij;
        while (++k <= n) if (k != j + 1) x[k][j] = -a[i][k] / aij;
        i--;
    }

    // Thế ẩn này vào ẩn kia vì xi biểu diễn qua các xj với j > i nên sẽ mất dần các
tham số
    for (int i = 0; i < n; i++)
        for (int j = i + 2; j <= n; j++) // Ví dụ thế các xj khác vào x2 thì j > 2
            if (!ISZERO(x[j][i])) {
                double tmp = x[j][i]; // hệ số của xj trong biểu diễn xi từ bước trên
(xi = ... -2xj + ... thì tmp = -2)
```

```
                for (int z = 0; z <= n; z++) x[z][i] = x[z][i] + tmp * x[z][j - 1]; //
Thế xj vào xi
                x[j][i] = x[j][i] - tmp; // Phải trừ đi vì xi = bi + ai*xi + ...,\
                nếu ai = 1 (xi được chọn làm tham số) thì sau vòng lặp for trên xij = ai
+ tmp,\
                nếu ai = 0 (xi được biểu diễn qua ẩn khác) thì sau vòng for xij = tmp,\
                cả 2 trường hợp đều cần trừ đi tmp để đúng với hệ số
            }

    // In nghiệm
    for (int i = 0; i < n; i++) {
        cout << "x_" << i + 1 << " = " << x[0][i];
        for (int j = 1; j <= n; j++) if (!ISZERO(x[j][i]))
                cout << " " << showpos << x[j][i] << noshowpos << "x_" << j;
        cout << endl;
    }

    // Kiểm tra lại nghiệm
    for (int i = 0; i <= n; i++) {
        if (!kiemtra(x[i], -1, i)) {
            cout << endl << "---> False" << endl;
            break;
        }
        else if (i==n) cout << endl << "---> True" << endl;
    }
}
```

4.4. Hệ thống ví dụ:

Ví dụ 1: Hpt có duy nhất một nghiệm:

$$\begin{cases} x_1 + x_2 + x_3 = 3 \\ 2x_1 - x_2 + 2x_3 = -3 \\ x_1 - 3x_2 - 3x_3 = -5 \end{cases}$$

```
Vi tri dau tien khac 0: 1 1
0 : 1.0000     1.0000     1.0000     3.0000
1 : 0.0000     -3.0000    0.0000     -9.0000
2 : 0.0000     0.0000     -4.0000    4.0000
----------------------------------------------------------------
He co nghiem duy nhat
x_1 = 1.0000
x_2 = 3.0000
x_3 = -1.0000

---> True
```

Ví dụ 2: Với hệ cấp 10x9 và TH vô nghiệm.

12	19	18	5	17	3	10	10	20	1
14	1	2	12	13	19	15	12	9	19
14	14	7	19	1	2	3	10	7	14
15	2	1	6	4	7	19	2	19	15
7	18	1	14	11	3	5	1	1	4
18	4	5	15	5	12	2	6	19	1
14	19	2	12	16	1	12	3	9	16
6	5	12	20	7	15	17	7	7	14
17	2	4	20	12	5	6	4	17	3
14	4	15	9	10	17	20	5	14	14

```
Vi tri dau tien khac 0: 1 1
0 : 12.0000    19.0000    18.0000    5.0000    17.0000    3.0000    10.0000    10.0000    20.0000    1.0000
1 : 0.0000     -21.1667   -19.0000   6.1667    -6.8333    15.5000   3.3333     0.3333     -14.3333   17.8333
2 : 0.0000     0.0000     -6.6693    10.7874   -16.1969   -7.4803   -9.9528    -1.7953    -10.8031   5.9528
3 : 0.0000     0.0000     0.0000     -9.7834   -5.4286    -10.4604  6.0242     -10.3105   11.9298    -6.3388
4 : 0.0000     0.0000     0.0000     0.0000    43.8305    37.0958   16.1184    12.4773    -4.9156    3.1990
5 : 0.0000     0.0000     0.0000     0.0000    0.0000     -0.0019   -11.9370   -6.1181    4.5978     -20.4435
6 : 0.0000     0.0000     0.0000     0.0000    0.0000     0.0000    31792.6364 16302.0704 -12254.3578 54467.8240
7 : 0.0000     0.0000     0.0000     0.0000    0.0000     0.0000    0.0000     -34.6002   29.5861    -49.3013
8 : 0.0000     0.0000     0.0000     0.0000    0.0000     0.0000    0.0000     0.0000     6.9928     -25.3041
9 : 0.0000     0.0000     0.0000     0.0000    0.0000     0.0000    0.0000     0.0000     0.0000     -39.5862
------------------------------------------------------------
No Solution
```

Ví dụ 3: Ví dụ thể hiện thuật toán đã giải quyết được vấn đề hệ không full hạng.

$$\begin{cases} x_1 + 2x_2 + 3x_3 + 5x_4 = 0 \\ 2x_1 + 4x_2 + 6x_3 + 7x_4 = 0 \\ 3x_1 + 6x_2 + 5x_3 + 4x_4 = 0 \end{cases}$$

```
Vi tri dau tien khac 0: 1 1
0 : 1.0000     2.0000     3.0000     5.0000     0.0000
2 : 0.0000     0.0000     -4.0000    -11.0000   0.0000
3 : 0.0000     0.0000     0.0000     -3.0000    0.0000
------------------------------------------------------------
Rank A: 3
x_1 = 0.0000 -2.0000x_2
x_2 = 0.0000 +1.0000x_2
x_3 = 0.0000
x_4 = -0.0000

---> True
```

Ví dụ 4: Ví dụ thể hiện thuật toán đã sửa được lỗi sắp xếp lại ma trận về dạng bậc thang.

$$\begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 7x_6 = 7 \\ 2x_1 + 4x_2 + 6x_3 + x_4 - 2x_5 + 3x_6 = 3 \\ 3x_1 + 6x_2 + 7x_3 + 4x_4 - 8x_5 + 6x_6 = 6 \\ 4x_1 + 8x_2 + 10x_3 + x_4 - 15x_5 + 2x_6 = 2 \end{cases}$$

```
Vi tri dau tien khac 0: 1 1
0 : 1.0000     2.0000     3.0000     4.0000     5.0000     7.0000     7.0000
2 : 0.0000     0.0000     -2.0000    -8.0000    -23.0000   -15.0000   -15.0000
3 : 0.0000     0.0000     0.0000     -7.0000    -12.0000   -11.0000   -11.0000
6 : 0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
------------------------------------------------------------
Rank A: 3
x_1 = -2.9286 -2.0000x_2 +15.7857x_5 +2.9286x_6
x_2 = 0.0000 +1.0000x_2
x_3 = 1.2143 -4.6429x_5 -1.2143x_6
x_4 = 1.5714 -1.7143x_5 -1.5714x_6
x_5 = 0.0000 +1.0000x_5
x_6 = 0.0000 +1.0000x_6

---> True
```

Kết luận

+) Ở đây ta nêu ra một số đặc điểm của phương pháp Gauss:

- Là một phương pháp giải đúng nghiệm của hệ phương trình
- Chia ra làm hai bước: Bước thuận và bước nghịch
- Ý tưởng đơn giản, tuy nhiên việc cách xử lí để đưa ra nghiệm lại khá phức tạp khi có khá nhiều trường hợp đặc biệt.

+) Ưu điểm:

- Có thể sử dụng giải hệ với số phương trình và ẩn không cố định.
- Có thể giải cả phương trình có định thức = 0
- Khối lượng phép tính nhỏ với hệ n ẩn n phương trình:

$N = \frac{n}{3}(n^2+3n-1)$ số phép nhân (chia)

$C = \frac{n}{2}(2n^2+3n-5)$ số phép cộng (trừ)

+) Nhược điểm: khi giải nghiệm ta phải chia cho $a_{ii}^{(k)} \approx 0$ thì nghiệm sẽ gặp sai số lớn

+) So sánh với các phương pháp khác:

- Sai số tính toán khuếch đại hơn so với phương pháp Gauss-Jordan do phương pháp Gauss-Jordan có áp dụng kĩ thuật chọn phần tử trội.

Kết luận chung: Đây là một phương pháp đơn giản nhưng khá hữu hiệu để giải đúng những hệ phương trình tuyến tính có kích thước

nhỏ. Tuy vậy, nó cũng có một số hạn chế về tốc độ, khả năng thực hiện song song, … Do vậy, nó thường được sử dụng ở những hệ có hàng ngàn phương trình. Với những hệ có hàng triệu phương trình, những phương pháp lặp như lặp Jacobi, lặp Gauss Seidel… thường được lựa chọn vì những lợi ích về tốc độ mà nó mang lại.

Tài liệu tham khảo

[1] Phạm Kỳ Anh, Giải tích số, NXB ĐHQG Hà Nội 1996

[2] Lê Trọng Vinh, Giải tích số, NXB.

[3] Lê Trọng Vinh, Trần Minh Toàn, Giáo trình phương pháp tính và Matlab, NXB Bách khoa.