CHƯƠNG 2
CÁC PHƯƠNG PHÁP SỐ TRONG ĐẠI SỐ TUYẾN TÍNH

MỤC ĐÍCH, YÊU CẦU:

Sau khi nghiên cứu chương 1, yêu cầu sinh viên:

1. Hiểu và nắm được các phương pháp tìm nghiệm đúng, nghiệm xấp xỉ của hệ phương trình tuyến tính.

2. Biết cách ứng dụng các phương pháp trên vào việc tính định thức của ma trận, tìm ma trận nghịch đảo, giải quyết các bài toán thực tế.

3. Biết cách đánh giá sai số của từng phương pháp

2.1. MA TRẬN VÀ ĐỊNH THỨC

2.1.1. Ma trận

Cho ma trận chữ nhật A cấp m x n:

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

ở đây a_{ij} là các số thực. Ma trận này có m hàng và n cột. Khi m = n ta có ma trận cấp nxn và *được gọi tắt là ma trận vuông cấp n*.

Ma trận vuông cấp n mà mọi phần tử nằm ngoài đường chéo chính bằng 0, tức là $a_{ij} = a_{ji} = 0$ với $i \neq j$, được gọi là ***ma trận đường chéo***. Nếu ma trận đường chéo có $a_{ii} = 1$ thì ta gọi A là ma trận đơn vị và ta thường ký hiệu là E hoặc I.

Ma trận vuông A được gọi là ***ma trận tam giác trên***, nếu A có dạng

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ . & . & \dots & . \\ 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

Tương tự, ma trận vuông A được gọi là ***ma trận tam giác dưới***, nếu A có dạng:

$$A = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ . & . & \dots & . \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Ma trận chữ nhật A^T cấp n x m được gọi là ***ma trận chuyển vị*** của ma trận A cấp m x n nếu:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ . & . & \dots & . \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix}$$

2.1.2. Định thức của ma trận

Trước khi đưa ra định nghĩa định thức của ma trận, chúng tôi giới thiệu khái niệm hoán vị chẵn, hoán vị lẻ của một tập hợp n số nguyên {1, 2, ... , n}.

Cho $\alpha = (i_1, i_2,..., i_n)$ là một hoán vị của tập {1,2,...,n}. Ta xét tất cả các cặp (i_k, i_h), trong đó $k < h$. Nếu $i_k > i_h$ thì ta gọi cặp (i_k, i_h) là cặp ngược, tức là các giá trị i_k, i_h được sắp xếp ngược với k,h. Nếu trong α số cặp ngược là chẵn thì ta gọi α là *hoán vị chẵn*, ngược lại thì ta gọi α là *hoán vị lẻ*.

Với mỗi ma trận vuông A cấp n:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & \dots & . \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

tồn tại một số thực được gọi là ***định thức*** của ma trận A, ký hiệu là det A, được xác định bởi công thức:

$$\det A = \sum_{\alpha} s(i_1, i_2,..., i_n)\ a_{1i_1} a_{2i_2} ... a_{ni_n} \qquad (2.0)$$

với $\alpha = (i_1, i_2,..., i_n)$ chạy trong tập tất cả các hoán vị của tập {1,2,...,n}, và

$$s(i_1, i_2,..., i_n) = \begin{cases} 1 \text{ nếu } \alpha \text{ là hoán vị chẵn} \\ -1 \text{ nếu } \alpha \text{ là hoán vị lẻ} \end{cases}$$

Định thức của ma trận còn được ký hiệu là

$$A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ . & . & \dots & . \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}$$

Với mỗi ma trận chữ nhật A cấp m x n bất kỳ ta có thể tính định thức của tất cả các ma trận con vuông cấp k, với $k \leq \min(m, n)$. Nếu tồn tại một số r sao cho có một ma trận con cấp r có định thức khác 0, còn mọi ma trận con vuông cấp lớn hơn r đều bằng 0 thì ta nói rằng r là ***hạng của ma trận*** A.

Các phép biến đổi sơ cấp sau đây không làm biến đổi hạng của ma trận:

- Đổi chỗ 2 hàng hoặc 2 cột bất kỳ.
- Nhân một hàng hay một cột bất kỳ với một số khác không.
- Cộng các thành phần tương ứng của 2 hàng hoặc hai cột bất kỳ.

Các phép biến đổi sơ cấp sẽ được sử dụng để tính định thức của ma trận và tìm nghiệm của hệ phương trình tuyến tính.

Ma trận E được gọi là ma trận đơn vị cấp n nếu E là ma trận vuông cấp n và E có dạng

$$E = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ . & . & \dots & . \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

2.1.3. Các phương pháp tính định thức

a. Tính định thức dựa trực tiếp vào định nghĩa

Ta có thể dùng (2.0) để tính định thức của một ma trận trên máy tính. Tuy nhiên cách tính này đòi hỏi khoảng c*n! phép tính. Đây là con số khổng lồ với n không lớn lắm. Ví dụ với máy tính hiện đại nhất hiện nay cũng cần hàng triệu năm để tính định thức của ma trận cấp n = 25.

b. Tính định thức dựa vào công thức khai triển theo hàng

Cho A là ma trận vuông cấp n và a_{ij} là một phần tử bất kỳ của nó. Định thức của ma trận con cấp n-1 sau khi "xóa" hàng thứ i và cột thứ j đi và không thay đổi vị trí các thành phần còn lại, được gọi là *minor* của phần tử a_{ij} , và được ký hiệu là M_{ij}. Giá trị $A_{ij} = (-1)^{i+j} M_{ij}$ được gọi là phần bù đại số của phần tử a_{ij}. Ta có các công thức sau để tính định thức ma trận vuông cấp n thông qua việc tính định thức của các ma trận con cấp bé hơn:

Khai triển định thức theo hàng thứ i:

$$\det A = \sum_{j=1}^{n} a_{ij} A_{ij}$$

Khai triển định thức theo cột thứ j:

$$\det A = \sum_{i=1}^{n} a_{ij} A_{ij}$$

Áp dụng các công thức trên đây ta có thể dùng thuật toán đệ quy sau đây để tính định thức của ma trận vuông cấp n :

Nếu $n = 1$: $A_{11} = 1$; $\det A = a_{11} A_{11}$

$n > 1$: $\det A = \sum_{j=1}^{n} a_{1j} A_{1j}$

Tuy nhiên, cũng như cách tính trực tiếp, cách tính này cần khoảng c*n! phép tính, và như vậy không thể thực hiện được trên máy tính hiện đại nhất hiện nay dù chỉ với n không lớn lắm. Rõ ràng việc phân tính thuật toán giúp chúng ta đánh giá được thời gian tính toán trên máy tính và nếu thời gian đó là quá lớn thì chúng ta khỏi phải tốn công vô ích viết chương trình và chạy thử.

c. Tính định thức bằng cách chuyển ma trận về dạng tam giác trên

Ta sẽ biến đổi để đưa ma trận A về dạng ma trận tam giác trên

$$B = \begin{bmatrix} b_{11} & b_{12} & ... & b_{1n} \\ 0 & b_{22} & ... & B_{2n} \\ . & . & ... & . \\ 0 & 0 & ... & b_{mn} \end{bmatrix}$$

Vậy $\det A = \det B = b_{11} b_{22} ... b_{nn}$

2.1.4. Ma trận nghịch đảo

Ma trận nghịch đảo của một ma trận vuông A cấp n là ma trận được ký hiệu là A^{-1}, thoả mãn điều kiện

$$A^{-1}A = A A^{-1} = E$$

Trong đó E là ma trận đơn vị. Có thể chứng minh rằng để thỏa mãn điều kiện trên thì bắt buộc A^{-1} phải là ma trận vuông, và ma trận đảo nếu tồn tại là duy nhất.

Điều kiện tồn tại của ma trận nghịch đảo: Ma trận vuông A cấp n có ma trận nghịch đảo ***khi và chỉ khi*** $\det A \neq 0$.

Cách tính ma trận nghịch đảo:

Gọi A_{ij} là phần bù đại số của phần tử a_{ij} , khi đó ta có:

$$A^{-1} = \frac{1}{\det A} \begin{bmatrix} A_{11} & A_{21} & ... & A_{n1} \\ A_{12} & A_{22} & ... & A_{n2} \\ & & & \\ . & . & ... & . \\ A_{1n} & A_{2n} & ... & A_{nn} \end{bmatrix}$$

Tuy nhiên công thức này chỉ có ý nghĩa lý thuyết, không thể áp dụng để tính trực tiếp ma trận đảo trên máy tính được vì số phép tính đòi hỏi quá lớn.

Trong phần sau ta sẽ áp dụng phương pháp khử Gauss-Jordan để tính ma trận nghịch đảo với số phép tính nhỏ hơn nhiều (khoảng n^3)

2.2. HỆ PHƯƠNG TRÌNH ĐẠI SỐ TUYẾN TÍNH

Xét một hệ phương trình gồm n phương trình tuyến tính với n ẩn số $x_1, x_2,...,x_n$ như sau:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \qquad (2.1)$$

Hệ phương trình này có thể viết dưới dạng ma trận $Ax = b$, trong đó

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ . & . & \ldots & . \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ x_n \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ . \\ b_n \end{pmatrix}$$

Nếu $\det A \neq 0$ thì nghiệm của hệ (2.1) có thể tính theo công thức $x = A^{-1}b$. Áp dụng công thức tính ma trận đảo ta có thể biến đổi và dẫn đến lời giải được diễn tả bằng định lý Cramer như sau:

Định lý Cramer. Gọi A_j là ma trận nhận được từ ma trận A bằng cách thay cột thứ j bằng cột b, khi đó hệ (2.1) có nghiệm duy nhất và x_j được tính bởi công thức

$$x_j = \frac{\det A_j}{\det A}$$

Tuy nhiên trong thực hành người ta không dùng công thức này để tính nghiệm vì số phép tính quá lớn. Người ta dùng những phương pháp hữu hiệu hơn mà chúng tôi sẽ giới thiệu sau đây.

2.2.1. Phương pháp trực tiếp giải hệ phương trình tuyến tính

Giả sử ta giải hệ phương trình(2.1)

a. Phương pháp khử Gauss

Phương pháp khử Gauss dùng cách khử dần các ẩn để đưa hệ phương trình đã cho về một dạng tam giác trên rồi giải hệ tam giác này từ giới lên trên, không phải tính một định thức nào

Phương pháp này được thực hiện qua các bước sau:

Quá trình xuôi:

- Bước 0: Dùng phương trình đầu tiên để khử x_1 trong n-1 phương trình còn lại. Giả sử $a_{11}\neq 0$. (Để cho công thức đơn giản , trước khi khử ta có thể chia phương trình thứ nhất cho a_{11}).

  Cụ thể để khử x_1 ở hàng thứ k(k=2,3,…n) ta phải tính lại các hệ số a_{kj} ở hàng thứ k (j=1,2,..n+1) như sau: $a_{kj}=a_{kj}-a_{1j}*a_{k1}/a_{11}$

  . . .

- **Bước 1:** Dùng phương trình thứ 2 để khử x_2 trong n-2 phương trình còn lại phía sau. Giả sử $a_{22}\neq 0$. (Để cho công thức đơn giản, trước khi khử ta có thể chia phương trình thứ hai cho a_{22}).

  Cụ thể để khử x_2 ở hàng thứ k (k=3,4,…n) ta phải tính lại các hệ số a_{kj} ở hàng thứ k (j=2,..n+1) như sau: $a_{kj}=a_{kj}-a_{2j}*a_{k2}/a_{22}$

  …….

- **Bước i:** Dùng phương trình i để khử x_i trong các phương trình thứ i+1,i+2, ..., n. Giả sử $a_{ii}\neq 0$. Để cho công thức đơn giản, trước khi khử ta có thể chia phương trình thứ i cho a_{ii}).

  Cụ thể để khử x_i ở hàng thứ k (k=i+1,…n) ta phải tính lại các hệ số a_{kj} ở hàng thứ k (j=i,..n+1) như sau: $a_{kj}=a_{kj}-a_{ij}*a_{ki}/a_{ii}$

- **Bước n-1:** Dùng phương trình thứ n-1 để khử x_{n-1} trong phương trình thứ n.Giả sử $a_{n-1\ n-1}\neq 0$. (Để cho công thức đơn giản, trước khi khử ta có thể chia phương trình thứ n-1 cho $a_{n-1\ n-1}$)

  Cụ thể để khử x_{n-1} ở hàng thứ n ta phải tính lại các hệ số a_{nj} ở hàng thứ n (j=n-1,n,n+1) như sau: $a_{nj}=a_{nj}-a_{n-1j}*a_{n-1i}/a_{n-1n-1}$

Kết thúc quá trình khử.

Chú ý:

Trong quá trình giải xuôi ta giả thiết $a_{11}\neq 0$, $a_{22}\neq 0$,$a_{33}\neq 0$,...,$a_{n-1\ n-1}\neq 0$. Nếu 1 trong các hệ số đó bằng không thì quá trình không tiếp tục được. Lúc đó ta phải thay đổi cách tính.

Giả sử khi khử x_1 ta gặp $a_{11}=0$ thì ta nhìn các hệ số a_{21}, a_{31} ...a_{n1} của x_1 ở các phương trình phía dưới, nếu có hệ số nào khác không ta có thể lấy nó thay cho vai trò của a_{11} bằng cách hoán vị hai phương trình. Nếu tất cả các hệ số số a_{11}, a_{21}, a_{31} ...,a_{n1} đều bằng không thì hệ đã cho suy biến. Vậy tốt nhất là trước khi khử x_1 ta chọn trong các hệ số a_{11}, a_{21}, a_{31} ...,a_{n1} hệ số có giá trị tuyệt đối lớn nhất làm trụ thứ nhất**(gọi là trụ tối đại thứ nhất)** rồi hoán vị hàng thứ nhất cho hàng có giá trị tuyệt đối lớn nhất). Tức là ta chọn hàng r sao cho:

$|a_{r1}| = \max\{|a_{k1}| / k=1,2,...,n\}$ Sau đó ta đổi hàng r cho hàng 1.

Tương tự trong các bước khử x_2,... x_{n-1} , trước khi khử ta cũng **tìm trụ tối đại**:

$|a_{ri}| = \max\{|a_{ki}| / k=i,i+1,...,n\}$ (với i=2,3,…,n-1)

Sau đó ta đổi hàng r cho hàng i.

Sau khi thực hiện xong quá trình giải xuôi hệ phương trình (2.1) có dạng:

Dạng1: Tại các bước (bước i) ta không chia cho hệ số a_{ii}

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots \\ a_{nn}x_n = b_n \end{cases}$$

hoặc: Dạng 2: Tại các bước (bước i) ta chia cho hệ số a_{ii}:

$$\begin{cases} x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ \quad x_2 + \ldots + a_{2n}x_n = b_2 \\ \quad \ldots\ldots\ldots\ldots \\ \quad\quad\quad x_n = b_n \end{cases}$$

Xuất phát từ phương trình thứ n ta lần lượt tính được các giá trị x_i bằng các công thức của quá trình giải ngược sau:

Quá trình giải ngược

$x_n = b_n/a_{nn}$ hoặc ($x_n=b_n$)

. . .

$$x_i = (b_i - (\sum_{j=i+1}^{n} a_{ij}x_j))/a_{ii}) \text{ hoặc } (b_i - (\sum_{j=i+1}^{n} a_{ij}x_j)), \quad i = n-1, n-2, ..., 1$$

Để việc viết chương trình được đơn giản, khi cài đặt trên máy tính ta dùng một mảng a thay cho cả ma trận a và vec tơ b. Tức là

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} & a_{1,(n+1)} \\ a_{21} & a_{22} & \ldots & a_{2n} & a_{2,(n+1)} \\ . & . & \ldots & . & . \\ a_{n1} & a_{n2} & \ldots & a_{nn} & a_{n,(n+1)} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ . & . & \ldots & . & . \\ a_{n1} & a_{n2} & \ldots & a_{nn} & b_n \end{bmatrix}$$

Ta áp dụng các phép biến đổi sơ cấp như vừa trình bày để biến đổi ma trận chữ nhật cấp nx(n+1) trên đây về dạng

$$\begin{bmatrix} 1 & a'_{12} & \ldots & a'_{1n} & a'_{1,(n+1)} \\ 0 & 1 & \ldots & a'_{2n} & a'_{2,(n+1)} \\ . & . & \ldots & . & . \\ 0 & 0 & \ldots & 1 & a'_{n,(n+1)} \end{bmatrix}$$

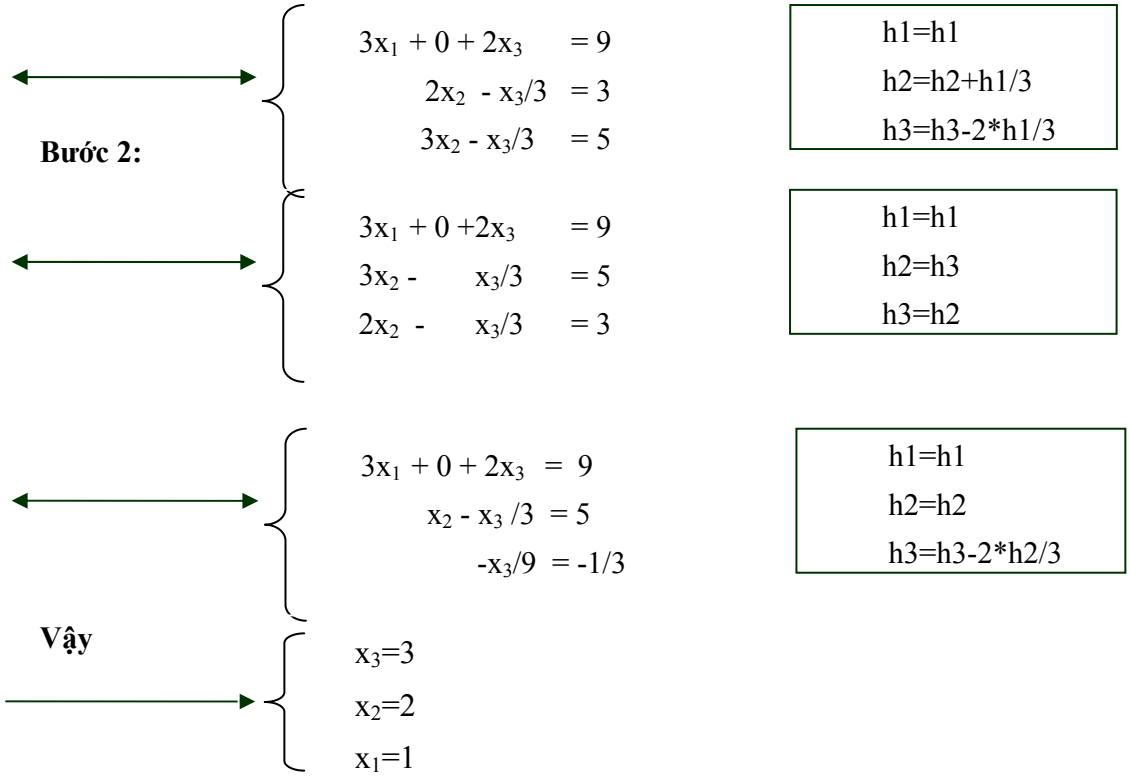
Ví dụ:Giải hệ phương trình sau bằng phương pháp khử Gauss:

$$\begin{cases} 2x_1 + 3x_2 + x_3 = 11 \\ -x_1 + 2x_2 - x_3 = 0 \\ 3x_1 + 2x_3 = 9 \end{cases}$$

Bước1: Hệ phương trình trên tương đương với:

$$\begin{cases} 3x_1 + 2x_3 = 9 \\ -x_1 + 2x_2 - x_3 = 0 \\ 2x_1 + 3x_2 + x_3 = 11 \end{cases}$$

h1=h3
h2=h2
h3=h1



Chương trình minh họa.

Sau đây là đoạn chương trình chính thể hiện (mô tả) thuật toán khử Gauss.

/*Giai he phuong trinh tuyen tinh dung khu Gauss, ma tran vuong n,
cac phan tu cot thu n+1 la vecto b*/
/*Dua ma tran a ve dang tam giac tren Giai he phuong trinh tuyen tinh.
Tra ve gia tri true neu co nghiem */

```
int khugauss(kmatran a,double *x,int n)
 {
  int i,j,k,h;double tmp,p;kmatran aa;
  int n1=n+1;
  for(i=1;i<=n;i++)
  for(j=1;j<=n1;j++) aa[i][j]=a[i][j];
  for(i=1;i<=n;i++) //Vong lap cac buoc khu
   {//Tim hang co phan tu dau lon nhat
    h=i;
    for(k=i+1;k<=n;k++)
     if(fabs(a[k][i])>fabs(a[h][i]) {h=k;}
    if(a[h][i]==0) {cout<<"Ma tran suy bien";delay(1000);return false;}
```

```
  if(h!=i) //Doi hang i va hang h vi a[h][i] > a[i][i]
   {int j;double tmp;
    for(j=i;j<=n1;j++)
     {tmp=a[i][j];a[i][j]=a[h][j];a[h][j]=tmp;}
   }
  //chuyen he so a[i][i] = 1
  tmp=a[i][i];
  for(j=i;j<=n1;j++) a[i][j] = a[i][j]/tmp;
  //Bat tinh lai cac hang
  for(k=i+1;k<=n;k++)
   {p=a[k][i];
    /*Vi ta biet a[k][i] =0 sau bien doi,
 chi tinh tu a[k][i+1]*/
    for(j=i+1;j<=n1;j++) a[k][j]=a[k][j] - p*a[i][j];
   }
 }
x[n]=a[n][n+1];
for(i=n-1;i>=1;i--)
 {double xx=0;
  for(j=i+1;j<=n;j++) xx=xx+a[i][j]*x[j];
  x[i]=a[i][n+1]-xx;//b[i]-xx
 }
//Dat cac gia tri phi duoi duong cheo chinh bang 0(phan nay khong can)
for(i=2;i<=n;i++)
for(j=1;j<i;j++) a[i][j]=0;

//Thu lai
kvecto bb;
for(i=1;i<=n;i++)
 {bb[i]=aa[i][1]*x[1];
  for(j=2;j<=n;j++) bb[i]+=aa[i][j]*x[j];
 }
//Dua ket qua vao tep ketqua
return true;
}
```

b. Phương pháp khử Gauss-Jordan

Phương pháp khử Gauss-Jordan dùng cách khử dần các ẩn để đưa hệ phương trình đã cho về một dạng ma trận đường chéo rồi giải hệ phương trình này, không phải tính một định thức nào

Phương pháp này được thực hiện qua các bước sau:

- **Bước 1:** Dùng phương trình đầu tiên để khử x_1 trong n-1 phương trình còn lại, cách làm tương tự như phương pháp khử để tính định thức... (Để cho công thức đơn giản, trước khi khử ta có thể chia phương trình thứ nhất cho a_{11}).

  Cụ thể để khử x_1 ở hàng thứ k(k=2,3,…n) ta phải tính lại các hệ số a_{kj} ở hàng thứ k (j=1,2,..n+1) như sau: $a_{kj}=a_{kj}-a_{1j}*a_{k1}/a_{11}$

  . . .

- **Bước i:** Dùng phương trình i để khử x_i trong các phương trình thứ 1,2, i-1,i+1,i+2,...,n.. (Để cho công thức đơn giản , trước khi khử ta có thể chia phương trình thứ i cho a_{ii})

  Cụ thể để khử x_i ở hàng thứ k (k=1,2, i-1,i+1,i+2,...,n.) ta phải tính lại các hệ số a_{kj} ở hàng thứ k (j=i,..n+1) như sau: $a_{kj}=a_{kj}-a_{ij}*a_{ki}/a_{ii}$

  . . .

- **Bước n:** Dùng phương trình thứ n để khử x_n trong phương trình thứ 1,2, ..., n-1.. (Để cho công thức đơn giản, trước khi khử ta có thể chia phương trình thứ n cho a_{nn})

  Cụ thể để khử x_n ở hàng thứ k(k=1,2, ..,n-1.) ta phải tính lại các hệ số a_{kj} ở hàng thứ k (j=n,n+1) như sau: $a_{kj}=a_{kj}-a_{nj}*a_{kn}/a_{nn}$

Tương tự phép khử Gauss tại mỗi bước, trước khi khử ta phải chọn trụ tối đại. Cụ thể tại bước i ta luôn chọn hàng có phần tử a_{ri} có giá trị tuyệt đối lớn nhất rồi đổi cho hàng thứ i cho hàng thứ r.

Hệ phương trình sau khi khử có dạng:

$$\begin{cases} a_{11}\,x_1 & & & = b_1 \\ & a_{22}\,x_2 & & = b_2 \\ & \cdots\cdots\cdots & & \\ & & a_{nn}\,x_n & = b_n \end{cases}$$

Hoặc (Nếu tại các bước (bước i) ta chia cho hệ số a_{ii}):

$$\begin{cases} x_1 & & & = b_1 \\ & x_2 & & = b_2 \\ & \cdots\cdots\cdots & & \\ & & x_n & = b_n \end{cases}$$

Tức là ta đã có các nghiệm mà không cần phải tính toán thêm.

Cũng như trong phương pháp khử Gauss, khi cài đặt trên máy tính ta dùng một mảng a thay cho cả ma trận A và vec tơ b. Tức là

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & a_{1,(n+1)} \\ a_{21} & a_{22} & \dots & a_{2n} & a_{2,(n+1)} \\ . & . & \dots & . & . \\ a_{n1} & a_{n2} & \dots & a_{nn} & a_{n,(n+1)} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ . & . & \dots & . & . \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

Ta áp dụng các phép biến đổi sơ cấp như vừa trình bày để biến đổi ma trận chữ nhật cấp n x (n+1) trên đây về dạng

$$\begin{bmatrix} 1 & 0 & \dots & 0 & a'_{1,(n+1)} \\ 0 & 1 & \dots & 0 & a'_{2,(n+1)} \\ . & . & \dots & . & . \\ 0 & 0 & \dots & 1 & a'_{n,(n+1)} \end{bmatrix}$$

Vậy ta có $x_i = a'_{i,(n+1)}$

Ví dụ:Giải hệ phương trình sau bằng phương pháp khử Gauss-Jordan:

$$\begin{cases} 2x_1 + 3x_2 + x_3 = 11 \\ -x_1 + 2x_2 - x_3 = 0 \\ 3x_1 + 2x_3 = 9 \end{cases}$$

Bước1: Hệ phương trình trên tương đương với:

$$\longleftrightarrow \quad \begin{cases} 3x_1 + 2x_3 = 9 \\ -x_1 + 2x_2 - x_3 = 0 \\ 2x_1 + 3x_2 + x_3 = 11 \end{cases}$$

h1=h3
h2=h2
h3=h1

Bước 1:

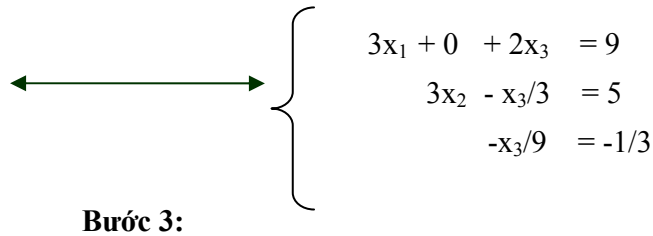
$$\longleftrightarrow \quad \begin{cases} 3x_1 + 0 + 2x_3 = 9 \\ 2x_2 - x_3/3 = 3 \\ 3x_2 - x_3//3 = 5 \end{cases}$$

h1=h1
h2=h2+h1/3
h3=h3-2*h1/3

Bước 2:

$$\longleftrightarrow \quad \begin{cases} 3x_1 + 0 + 2x_3 = 9 \\ 3x_2 - x_3/3 = 5 \\ 2x_2 - x_3/3 = 3 \end{cases}$$

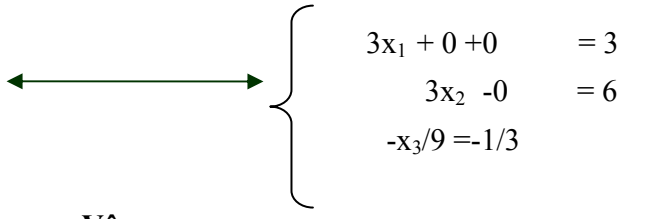
h1=h1
h2=h3
h3=h2



$$\begin{aligned} 3x_1 + 0 \quad + 2x_3 &= 9 \\ 3x_2 \; - x_3/3 &= 5 \\ -x_3/9 &= -1/3 \end{aligned}$$

h1=h1
h2=h2
h3=h3-2*h2/3

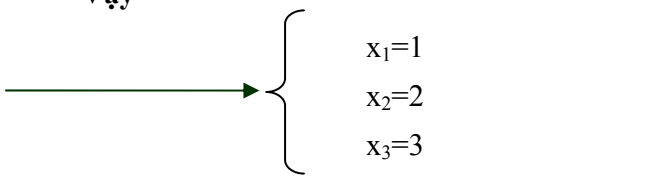
Bước 3:



$$\begin{aligned} 3x_1 + 0 + 0 &= 3 \\ 3x_2 \; - 0 &= 6 \\ -x_3/9 &= -1/3 \end{aligned}$$

h1=h1-2*h3/(-1/9)
h2=h2-(1/3)*h3/(-1/9)
h3=h3/(-1/9)

Vậy



$$\begin{aligned} x_1 &= 1 \\ x_2 &= 2 \\ x_3 &= 3 \end{aligned}$$

Chương trình minh họa.

Sau đây là đoạn chương trình chính thể hiện (mô tả) thuật toán khử Gauss-Jordan.

```
int gjordan(kmatran a,double *x,int n)
 {int i,j,k,h;double tmp,p;kmatran aa;
  int n1=n+1;
 for(i=1;i<=n;i++)
 for(j=1;j<=n1;j++) aa[i][j]=a[i][j];
 for(i=1;i<=n;i++) //Vong lap cac buoc khu
  {//Tim hang co phan tu dau lon nhat
   h=i;
   for(k=i+1;k<=n;k++)
    if(fabs(a[k][i])>fabs(a[h][i]) {h=k;}
   if(a[h][i]==0) {cout<<"Ma tran suy bien";delay(1000);return false;}
   if(h!=i) //Doi hang i va hang h vi a[h][i] > a[i][i]
    {int j;double tmp;
     for(j=i;j<=n1;j++)
      {tmp=a[i][j];a[i][j]=a[h][j];a[h][j]=tmp;}
    }
   //chuyen he so a[i][i] = 1
```

```
 tmp=a[i][i];
 for(j=i;j<=n1;j++) a[i][j] = a[i][j]/tmp;
 //Bat tinh lai cac hang
 for(k=1;k<=n;k++)
  {if(k==i) continue;
   p=a[k][i];
   /*Vi ta biet a[k][i] =0 sau bien doi,
 chi tinh tu a[k][i+1]*/
   for(j=i+1;j<=n1;j++) a[k][j]=a[k][j] - p*a[i][j];
  }
}
for(i=1;i<=n;i++) x[i]=a[i][n+1];

 /*Dat cac gia tri khong o tren duong cheo chinh bang 0
  (phan nay khong can)*/
 for(i=1;i<=n;i++)
 for(j=1;j<=n;j++) {if(i!=j) a[i][j]=0;}

//Thu lai
kvecto bb;
for(i=1;i<=n;i++)
 {bb[i]=aa[i][1]*x[1];
  for(j=2;j<=n;j++) bb[i]+=aa[i][j]*x[j];
 }
//Dua ket qua vao tep ketqua
return true;
```

2.2.2. Áp dụng phương pháp khử Gauss-Jordan để tính ma trận nghịch đảo

Để giải hệ n phương trình n ẩn Ax = b, trong phương pháp khử Gauss-Jordan ta đã dùng các phép biến đổi sơ cấp để đưa phương trình này về dạng

Ex = b'

Vì Ex = x, do đó ta có x=b'. Nếu B là một ma trận chữ nhật cấp n x k tùy ý, ta có thể áp dụng phương pháp khử Gauss-Jordan để giải đồng thời k hệ n phương trình n ẩn:

AX = B (2.2)

trong đó

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ x_{21} & x_{22} & \dots & x_{2k} \\ . & . & \dots & . \\ x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ . & . & \dots & . \\ b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix}$$

Ta viết ma trận B bên phải ma trận A như sau:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_{11} & b_{12} & \dots & b_{1k} \\ a_{21} & a_{21} & \dots & a_{2n} & b_{21} & b_{22} & \dots & b_{2k} \\ . & . & \dots & . & . & . & \dots & . \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_{n1} & b_{n2} & \dots & b_{nk} \end{bmatrix}$$

Nếu ma trận A không suy biến, ta có thể áp dụng các phép biến đổi sơ cấp để đưa ma trận này về dạng:

$$\begin{bmatrix} 1 & 0 & \dots & 0 & b'_{11} & b'_{12} & \dots & b'_{1k} \\ 0 & 1 & \dots & 0 & b'_{21} & b'_{22} & \dots & b'_{2k} \\ . & . & \dots & . & . & . & \dots & . \\ 0 & 0 & \dots & 1 & b'_{n1} & b'_{n2} & \dots & b'_{nk} \end{bmatrix}$$

Khi đó ta có

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ x_{21} & x_{22} & \dots & x_{2k} \\ . & . & \dots & . \\ x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix} = \begin{bmatrix} b'_{11} & b'_{12} & \dots & b'_{1k} \\ b'_{21} & b'_{22} & \dots & b'_{2k} \\ . & . & \dots & . \\ b'_{n1} & b'_{n2} & \dots & b'_{nk} \end{bmatrix};$$

Đặt $B' = \begin{bmatrix} b'_{11} & b'_{12} & \dots & b'_{1k} \\ b'_{21} & b'_{22} & \dots & b'_{2k} \\ . & . & \dots & . \\ b'_{n1} & b'_{n2} & \dots & b'_{nk} \end{bmatrix}$

Xét trường hợp đặc biệt B = E, ta có ma trận B' chính là ma trận nghịch đảo của ma trận A. Thật vậy, nếu X là nghiệm của (2.2) thì

$X = A^{-1}B$

Nếu B = E thì $X = A^{-1}$. Do đó việc tìm ma trận nghịch đảo của ma trận A tương đương với việc giải phương trình

$AX = E$

Ta có thể tóm tắt các bước cần thực hiện để tính ma trận đảo như sau:

- Viết thêm ma trận đơn vị E bên cạnh ma trận A

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & 1 & 0 & \dots & 0 \\ a_{21} & a_{21} & \dots & a_{2n} & 0 & 1 & \dots & 0 \\ . & . & \dots & . & . & . & \dots & . \\ a_{n1} & a_{n2} & \dots & a_{nn} & 0 & 0 & \dots & 1 \end{bmatrix} \qquad (2.3)$$

- Áp dụng phép biến đổi sơ cấp lên các hàng của ma trận (2.3) cho đến khi ma trận có dạng

$$\begin{bmatrix} 1 & 0 & \dots & 0 & c_{11} & c_{12} & \dots & c_{1n} \\ 0 & 1 & \dots & 0 & c_{21} & c_{22} & \dots & c_{2n} \\ . & . & \dots & . & . & . & \dots & . \\ 0 & 0 & \dots & 1 & c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix}$$

Khi đó ta có

$$A^{-1} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ . & . & \dots & . \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix}$$

Chú ý. Trong quá trình biến đổi ta *có thể đổi các hàng* của ma trận. Điều này không ảnh hưởng đến kết quả thu được: Ma trận C vẫn là ma trận nghịch đảo của ma trận A ban đầu. Lý do là vì để tìm ma trận nghịch đảo ta chỉ cần xác định ma trận nghiệm. Ma trận nghiệm không bị thay đổi nếu ta đổi chỗ các hàng.

Chương trình minh họa.

Sau đây là đoạn chương trình chính thể hiện (mô tả) thuật toán tìm ma trận nghịch đảo

```
int daomtran(kmatran a,kmatran &ad,int n)
{int i,j,k,h;double tmp,p;
 int n2=n*2;
 kmatran aa;
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++) aa[i][j]=a[i][j];
 //Them phan sau cua ma tran a de co dang ma tran don vi
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++) a[i][n+j]=0;//Cho phan ma tran vuong phia sau bang 0
 for(i=1;i<=n;i++) a[i][n+i]=1; //Duong cheo chinh phan phia sau bang 1
 //Vong lap cac buoc khu
 for(i=1;i<=n;i++)
  {//Tim hang co phan tu dau lon nhat
   h=i;
```

```
  for(k=i+1;k<=n;k++)
   if(fabs(a[k][i])> fabs(a[h][i])) h=k;
  if(a[h][i])==0) {cout<<"Ma tran suy bien";delay(1000);return false;}
  if(h!=i) //Doi hang i va hang h vi a[h][i] > a[i][i]
   {int j;double tmp;
    for(j=i;j<=n2;j++)
     {tmp=a[i][j];a[i][j]=a[h][j];a[h][j]=tmp;}
   }
  //chuyen he so a[i][i] = 1
  tmp=a[i][i];
  for(j=i;j<=n2;j++) a[i][j] = a[i][j]/tmp;
  //Bat tinh lai cac hang
  for(k=1;k<=n;k++)
   {if(k==i) continue;
    p=a[k][i];
    /*Vi ta biet a[k][i] =0 sau bien doi,
 chi tinh tu a[k][i+1]*/
    for(j=i+1;j<=n2;j++) a[k][j]=a[k][j] - p*a[i][j];
   }
 }
/*Dat cac gia tri khong o tren duong cheo chinh bang 0
  (phan nay khong can)*/
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++) {if(i!=j) a[i][j]=0;}

//Ma tran dao la phan phia sau cua mang a
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++) ad[i][j]=a[i][n+j];

 //Thu lai A x AD = C
kmatran c;
for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
  {c[i][j]=aa[i][1]*ad[1][j];
  for(k=2;k<=n;k++) c[i][j]+=aa[i][k]*ad[k][j];
  }
  return true;
}
```

2.2.3. Sự không ổn định của hệ phương trình đại số tuyến tính

a. Chuẩn của ma trận và vec tơ

Chuẩn của ma trận chữ nhật cấp m x n $A = (a_{ij})$ là một số thực không âm được ký hiệu là ||A|| thỏa mãn các điều kiện sau

(1) $||A|| \geq 0$ (với $||A|| = 0 \Leftrightarrow A = 0$)

(2) $||\alpha A|| = |\alpha| \, ||A||$, α là số thực bất kỳ.

(3) $||A + B|| \leq ||A|| + ||B||$

(4) $||A.B|| = ||A||.||B||$

Người ta thường dùng ba chuẩn sau:

Chuẩn cột: $$||A||_1 = \max_j \sum_{i=1}^{m} |a_{ij}|$$

Chuẩn Oclit: $$||A||_2 = (\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2)^{1/2}$$

Chuẩn hàng: $$||A||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

Ví dụ. Cho

$$A = \begin{bmatrix} 5 & -2 & 1 \\ 1 & 4 & 3 \\ 2 & -1 & 7 \end{bmatrix}$$

Ta tính được các chuẩn của A theo định nghĩa trên như sau:

$||A||_1 = \max(5+1+2, 2+4+1, 1+3+7) = \max(8, 7, 11) = 11$

$||A||_2 = (5^2 + 2^2 + 1 + 1 + 4^2 + 3^2 + 2^2 + 1 + 7^2)^{1/2} = 110^{1/2} = 10.5$

$||A||_\infty = \max(5+2+1, 1+4+3, 2+1+7) = \max(8, 8, 10) = 10$

Vec tơ là ma trận chỉ có một cột, do đó đối với vec tơ

$$x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ x_n \end{pmatrix}$$

ta có 3 chuẩn sau

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

$$||x||_2 = (\sum_{i=1}^{n} x_i^2)^{1/2}$$

$$||x||_\infty = \max_i |x_i|$$

Ví dụ. Cho

$$x = \begin{pmatrix} 2 \\ -3 \\ 4 \\ 1 \\ 4 \end{pmatrix}$$

Ta có

$||x||_1 = 2 + 3 + 4 + 1 + 4 = 14$

$||x||_2 = (2^2 + 3^2 + 4^2 + 1 + 4^2)^{1/2} = \sqrt{46}$

$||x||_\infty = \max(2,3,4,1,4) = 4$

Trong các phần tiếp theo chúng ta sẽ ký hiệu đơn giản là $||A||$ hoặc $||x||$ để chỉ chuẩn của ma trận và vec tơ. Nếu không có gì giải thích thêm thì cách ký hiệu này được hiểu là một trong ba chuẩn trên đây.

b. Sự không ổn định của hệ phương trình đại số tuyến tính

Trên đây ta đã tìm hiểu các phương pháp giải hệ phương trình đại số tuyến tính một cách trực tiếp. Nếu như mọi tính toán của ta là chính xác thì các phương pháp trên cho kết quả hoàn toàn chính xác. Tuy nhiên trong thực tế khi tính toán ta phải thường xuyên làm tròn các số, nghĩa là ta thường chỉ tính toán trên các số gần đúng mà thôi. Liệu cách làm tròn trong tính toán có làm ảnh hưởng nhiều đến kết quả cuối cùng không? Ví dụ sau đây cho thấy rằng có những hệ phương trình đại số tuyến tính rất "nhạy cảm" với sai số, nghĩa là sai số nhỏ khi tính toán có thể ảnh hưởng nghiêm trọng đến kết quả cuối cùng. Nói một cách hình tượng thì ta gặp tình huống "sai một li đi một dặm". Những hệ thống phương trình kiểu này được gọi là hệ phương trình không ổn định.

Ví dụ . Ta xét hệ phương trình sau:

$$2x_1 + x_2 = 2$$
$$2x_1 + 1.01x_2 = 2.01$$

Hệ này có nghiệm $x_1 = 0.5$, $x_2 = 1$.

Tuy nhiên hệ phương trình sau đây nhận được với chút ít thay đổi hệ số trong hệ trên

$$2x_1 + x_2 = 2$$
$$2.01x_1 + 1x_2 = 2.05$$

lại có nghiệm $x_1 = 5$, $x_2 = -8$, khác xa so với nghiệm trên đây.

2.2.4. Phương pháp lặp giải hệ phương trình tuyến tính

Các phương pháp trực tiếp giải hệ phương trình tuyến tính nói chung cần khoảng cn^3 phép tính, trong đó c là một hằng số và người ta ước lượng $c \approx 2/3$. Phương pháp khử Gauss như chúng ta vừa tìm hiểu chẳng hạn, là một phương pháp đúng, nghĩa là nếu các phép tính sơ cấp được thực hiện *đúng hoàn toàn* thì cuối cùng ta được *nghiệm đúng* của hệ. Tuy nhiên trong thực tế ta phải luôn luôn làm tròn khi thực hiện các phép tính, và như ta đã thấy ở trên, sai số tổng hợp đôi khi có thể sẽ khá lớn. Và chúng ta gặp một nghịch lý: về lý thuyết phương pháp cho kết quả chính xác

100%, nhưng khi thực hiện để áp dụng thực tế thì đôi khi kết quả lại khác xa so với kết quả lý thuyết. Vì những lý do trên đây, người ta đã tìm kiếm những phương pháp gần đúng để giải các bài toán, tức là ngay từ đầu người ta chấp nhận kết quả xấp xỉ, hay sự xấp xỉ đã nằm ngay trong mô hình. Khi thực hiện tính toán cụ thể chúng ta lại gặp sai số một lần nữa. Như vậy trong các phương pháp gần đúng thì sai số sẽ là tổng hợp của sai số mô hình và sai số tính toán. Một điều đáng ngạc nhiên là trong nhiều trường hợp phương pháp gần đúng lại cho kết quả tốt hơn phương pháp đúng. Thực ra điều này cũng không có gì khó hiểu, vì trong thực tế chúng ta cũng rất hay gặp những trường hợp một lần sai còn nặng nề trầm trọng hơn 2 lần hay thậm chí một số lần sai cộng lại.

a. Các bước chung trong phương pháp lặp

Giả sử ta cần giải phương trình $F(x) = 0$, trong đó $F(x)$ là một hàm trên không gian định chuẩn nào đó và 0 được hiểu là phần tử 0 của không gian này. Ví dụ nếu không gian định chuẩn là R^n thì 0 là vectơ $(0,0,...,0)^T$. Ta biến đổi phương trình này về dạng tương đương $x = G(x)$. Ta có thể phát biểu định lý sau:

Định lý. Giả sử $y = G(x)$ là một hàm liên tục trên không gian định chuẩn nào đó và phép lặp $x_n=G(x_{n-1})$ n=1,2,... hội tụ tới x^* với xuất phát ban đầu x_0. Khi đó x^* là nghiệm của phương trình $x = G(x)$, tức là ta có $x^* = G(x^*)$.

Chứng minh. Từ $x_n=G(x_{n-1})$, với lưu ý là hàm $G(x)$ liên tục, ta có

$$\lim_{n->+\infty} x_n = \lim_{n->+\infty} G(x_{n-1}) = G(\lim_{n->+\infty} x_{n-1}) \Rightarrow x^* = G(x^*)$$

b. Phương pháp lặp đơn

Trở lại bài toán giải hệ phương trình tuyến tính

$$Ax = b \qquad (2.4)$$

Ta đưa (2.4) về dạng

$$x = Cx + d \qquad (2.5)$$

Trong đó ma trận C và vec tơ d được xây dựng từ A và b.

Để thực hiện phép lặp ta chọn một vec tơ ban đầu $x^{(0)}$, sau đó tính các $x^{(i)}$, i =1,2,... theo công thức lặp sau:

$$\begin{aligned} x^{(1)} &= Cx^{(0)} + d \\ x^{(2)} &= Cx^{(1)} + d \\ &\dots \\ x^{(k)} &= Cx^{(k-1)} + d \end{aligned} \qquad (2.6)$$

. . .

Véc tơ $x^{(k)}$ được gọi là vec tơ lặp thứ k.

Ta có định lý sau:

Định lý. (Sự hội tụ của phương pháp)

a. Nếu phép lặp (2.6) hội tụ, tức là tồn tại x^* sao cho $x^* = \lim_{k->+\infty} x^{(k)}$

thì khi đó x^* là nghiệm của (2.5) (và như vậy cũng là nghiệm của (2.4))

b. Nếu ||C|| < 1 với một chuẩn nào đó, thì (2.6) hội tụ và sai số giữa nghiệm gần đúng $x^{(k)}$ (nghiệm gần đúng tại bước lặp thứ k) và nghiệm đúng x^* có thể đánh giá bằng các công thức sau:

$$||x^{(k)} - x^*|| \le \frac{||C||}{1-||C||} ||x^{(k)} - x^{(k-1)}|| \qquad (2.7)$$

hoặc

$$||x^{(k)} - x^*|| \le \frac{||C||^k}{1-||C||} ||x^{(1)} - x^{(0)}|| \qquad (2.8)$$

Nói chung theo phương pháp lặp đơn, điều kiện để phép lặp được hội tụ thì ||C|| < 1. Tuy nhiên trong thực tế thì ta chỉ có ma trận A. Một câu hỏi đặt ra là ma trận A phải thỏa mãn điều kiện gì để ta có thể đưa (2.4) về dạng (2.5) và áp dụng phương pháp lặp đơn?

Để phương pháp lặp hội tụ thì thường ma trận A phải thỏa mãn tính chéo trội của ma trận vuông.

Định nghĩa: (Tính chéo trội của một ma trận vuông): Ma trận A với các thành phần a_{ij} được gọi là ***có tính chéo trội***, nếu giá trị tuyệt đối của các phần tử nằm trên đường chéo chính lớn hơn tổng các giá trị tuyệt đối của các phần còn lại nằm cùng hàng, tức là

$$| a_{ii} | > \sum_{j=1, j\neq i}^{n} | a_{ij} |, \; i = 1, 2, \ldots, n. \qquad (2.9)$$

Sau đây sẽ giới thiệu 2 phương pháp lặp đơn Jacobi và Gaus-Seidel

c. Phương pháp lặp Jacobi

Với giả thiết ma trận A có tính chéo trội, khi đó các hệ số $a_{ii} \neq 0$, i = 1,2,...,n do đó ta có thể chia phương trình thứ i của hệ (2.1) cho a_{ii} và nhận được hệ tương tương

$$x_1 + \frac{a_{12}}{a_{11}} x_2 + \frac{a_{13}}{a_{11}} x_3 + \ldots + \frac{a_{1n}}{a_{11}} x_n = \frac{b_1}{a_{11}}$$

$$\frac{a_{21}}{a_{22}} x_1 + x_2 + \frac{a_{23}}{a_{22}} x_3 + \ldots + \frac{a_{2n}}{a_{22}} x_n = \frac{b_2}{a_{22}}$$

$$\ldots$$

$$\frac{a_{i1}}{a_{ii}} x_1 + \frac{a_{i2}}{a_{ii}} x_2 + \ldots + \frac{a_{i,i-1}}{a_{ii}} x_{i-1} + x_i + \frac{a_{i,i+1}}{a_{ii}} x_{i+1} \ldots + \frac{a_{in}}{a_{ii}} x_n = \frac{b_i}{a_{ii}}$$

$$\ldots$$

$$\frac{a_{n1}}{a_{nn}} x_1 + \frac{a_{n2}}{a_{nn}} x_2 + \ldots + \frac{a_{n,n-1}}{a_{nn}} x_{n-1} + x_n = \frac{b_n}{a_{nn}}$$

Từ đây ta có

$$x_1 = -\left(0.x_1 + \frac{a_{12}}{a_{11}} x_2 + \frac{a_{13}}{a_{11}} x_3 + \ldots + \frac{a_{1n}}{a_{11}} x_n\right) + \frac{b_1}{a_{11}}$$

$$x_2 = -\left(\frac{a_{21}}{a_{22}} x_1 + 0.x_2 + \frac{a_{23}}{a_{22}} x_3 + \ldots + \frac{a_{2n}}{a_{22}} x_n\right) + \frac{b_2}{a_{22}}$$

$$\ldots$$

$$x_i = -\left(\frac{a_{i1}}{a_{ii}}x_1 + \frac{a_{i2}}{a_{ii}}x_2 + \ldots + \frac{a_{i,i-1}}{a_{ii}}x_{i-1} + 0.x_i + \frac{a_{i,i+1}}{a_{ii}}x_{i+1}. \ldots + \frac{a_{in}}{a_{ii}}x_n\right) + \frac{b_i}{a_{ii}}$$

. . .

$$x_n = -\left(\frac{a_{n1}}{a_{nn}}x_1 + \frac{a_{n2}}{a_{nn}}x_2 + \ldots + \frac{a_{n,n-1}}{a_{nn}}x_{n-1} + 0.x_n\right) + \frac{b_n}{a_{nn}}$$

Khi đó ma trận C, vectơ d là:

$$C = -\begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} & \ldots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \ldots & \frac{a_{2n}}{a_{22}} \\ . & . & \ldots & . \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \ldots & 0 \end{bmatrix}, \quad d = \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ . \\ \frac{b_n}{a_{nn}} \end{pmatrix}$$

(Đến đây ta đã đưa hệ (2.4) về dạng (2.5) và dễ thấy rằng ma trận C thỏa mãn điều kiện lặp đơn, tức là $\|C\|_\infty < 1$.).

Vậy đến đây ta tiếp tục áp dụng phương pháp lặp (2.6) để tính nghiệm ở các bước lặp như sau:

Với vec tơ $x^{(0)}$ cho trước bất kỳ, ví dụ $x^{(0)} = \theta$ (vec tơ 0) ta có thể tính các vec tơ $x^{(k)}$ tại bước lặp k bằng công thức $x^{(k)} = C\,x^{(k-1)} + d$, k =1, 2, ...

Cụ thể hơn, nếu $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})$ thì ta có

$$\begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ . \\ x_n^{(k)} \end{pmatrix} = -\begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} & \ldots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \ldots & \frac{a_{2n}}{a_{22}} \\ . & . & \ldots & . \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \ldots & 0 \end{bmatrix} \begin{pmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \\ . \\ x_n^{(k-1)} \end{pmatrix} + \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ . \\ \frac{b_n}{a_{nn}} \end{pmatrix}$$

Với từng thành phần $x_i^{(k)}$ ta có

$$x_i^{(k)} = -\sum_{j-1, j\neq i}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k-1)} + \frac{b_i}{a_{ii}} = \frac{1}{a_{ii}}\left(b_i - \sum_{j-1, j\neq i}^{n} a_{ij}\, x_j^{(k-1)}\right) \qquad (2.10)$$

i = 1, 2, . . ., n, k = 1,2,...

Điều kiện hội tụ, đánh gái sai số của phương pháp lặp Jacobi cũng giống với phương pháp lặp đơn.

Ví dụ. Dùng phương pháp lặp Jacobi tìm nghiệm gần đúng của hệ phương trình:

$$\begin{cases} 4x_1 + 0.24x_2 - 0.08x_3 = 8 \\ 0.09x_1 + 3x_2 - 0.15x_3 = 9 \\ 0.04x_1 - 0.08x_2 + 4x_3 = 20 \end{cases}$$

Giải. (1).Có thể thấy rằng ma trận các hệ số của hệ phương trình trên đây thỏa mãn tính chéo trội, do đó ta có thể biến đổi hệ này để áp dụng phương pháp lặp Jacobi. Chia hai vế phương trình đầu tiên cho 4, hai vế phương trình thứ hai cho 3 và hai vế của phương trình thứ ba cho 4 rồi biến đổi thích hợp ta nhận được:

$$x_1 = 2 - 0.06x_2 + 0.02x_3$$
$$x_2 = 3 - 0.03x_1 + 0.05x_3$$
$$x_3 = 5 - 0.01x_1 + 0.02x_2$$

hay

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{bmatrix} 0 & -0.06 & 0.02 \\ -0.03 & 0 & 0.05 \\ -0.01 & 0.02 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} = Cx + d$$

$$\|C\|_\infty = \max(0 + 0.06 + 0.02,\ 0.03 + 0 + 0.05,\ 0.01 + 0.02 + 0) =$$
$$= \max(0.08, 0.08, 0.03) = 0.08 < 1$$

(2) Chọn $x^{(0)} = (2,3,5)^T$, rồi tính $x^{(1)}, x^{(2)}, \ldots$ theo công thức (2.10) với lưu ý $a_{ii} = 1$ ta được bảng kết quả sau:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	2	3	5
1	1.92	3.19	5.04
2	1.9094	3.1944	5.0446
3	1.909228	3.194948	5.044794

(3) Xem $x^{(3)}$ là nghiệm gần đúng cần tìm, ta có thể đánh giá sai $x^{(3)}$ với nghiệm đúng x^* theo (2.10) như sau: $\|x^{(3)} - x^*\| \le \dfrac{\|C\|}{1-\|C\|} \|x^{(3)} - x^{(2)}\|$

$$\|x^{(3)} - x^{(2)}\|_\infty = \max_i |x_i^{(3)} - x_i^{(2)}| = \max(0.000172, 0.000548, 0.000194) = 0.000548$$

Như vậy

$$\|x^{(3)} - x^*\|_\infty \le \frac{0.08}{1 - 0.08} 0.000548 = 0.0000476 \approx 0.00005$$

d. Phương pháp lặp Gauss - Seidel

Với giả thiết ma trận A có tính chéo trội. Từ công thức (2.10) ta thấy rằng phần tử thứ i của vec tơ nghiệm tại bước k được tính qua các phần tử ở các vị trí khác i trong bước k-1. Phương pháp Gauss-Seidel cải tiến phương pháp Jacobi bằng cách dùng ngay những kết quả vừa tính được cho các thành phần của nghiệm tại bước k để tính các thành phần khác của bước k, chỉ có những thành phần nào chưa được tính thì mới lấy ở bước k-1. Cụ thể hơn ta có tại các bước:

(1) Giá trị $x_1^{(1)}$ được tính qua các giá trị $x_2^{(0)}, x_3^{(0)}, \ldots x_n^{(0)}$

Giá trị $x_2^{(1)}$ được tính qua các giá trị $x_1^{(1)}, x_3^{(0)}, \ldots x_n^{(0)}$

Giá trị $x_3^{(1)}$ được tính qua các giá trị $x_1^{(1)}, x_2^{(1)}, x_4^{(0)}, \ldots x_n^{(0)}$

. . .

(h) Giá trị $x_1^{(h)}$ được tính qua các giá trị $x_2^{(h-1)}, x_3^{(h-1)}, \ldots x_n^{(h-1)}$

Giá trị $x_2^{(h)}$ được tính qua các giá trị $x_1^{(h)}, x_3^{(h-1)}, \ldots x_n^{(h-1)}$

Giá trị $x_3^{(h)}$ được tính qua các giá trị $x_1^{(h)}, x_2^{(h)}, x_4^{(h-1)}, \ldots x_n^{(h-1)}$

. . .

Với vec tơ $x^{(0)}$ cho trước bất kỳ, ví dụ $x^{(0)} = \theta$ (vec tơ 0) ta có thể tính các vec tơ $x^{(k)}$ tại bước lặp k bằng công thức

$$x_i^{(k)} = \frac{1}{a_{ii}}\left(b_i - \left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} + \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right)\right) \qquad (2.11)$$

$i = 1, 2, \ldots, n, k = 1,2,...$

Trong công thức (2.11) chúng ta có thể không dùng chỉ số trên để chỉ ra rằng chúng ta chỉ dùng một mảng là vec tơ có n thành phần để lưu trữ nghiệm. Giá trị nào vừa được tính toán thì được lưu trữ ngay vào vị trí cũ và được dùng ngay trong công thức tính các giá trị khác.

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}\, x_j\right)$$

$i = 1, 2, \ldots, n, k = 1,2,...$

Sự hội tụ của phương pháp Gause-Seidel

Điều kiện hội tụ của phương pháp lặp Gause- Seidel cũng giống với phương pháp lặp đơn. Như ta sẽ thấy trong ví dụ trong phần sau, phương pháp Gause- Seidel nói chung hội tụ nhanh hơn phương pháp lặp đơn.

Ta có thể sử dụng các công thức sau để đánh giá sai số của phương pháp lặp Gause-Seidel:

Gọi x^* là nghiệm đúng của hệ phương trình và gọi

$$p_i = \sum_{j=1}^{i-1} |c_{ij}|,\ q_i = \sum_{j=i}^{n} |c_{ij}|\ ,\ \mu = \max_i \frac{q_i}{1-p_i}$$

Khi đó ta có

$$\|x^{(k)} - x^*\| \le \frac{\mu}{1-\mu}\|x^{(k)} - x^{(k-1)}\| \qquad (2.12)$$

hoặc

$$\|x^{(k)} - x^*\| \le \frac{\mu^k}{1-\mu}\|x^{(1)} - x^{(0)}\| \qquad (2.13)$$

Ví dụ. Dùng phương pháp lặp ***Gause-Seidel*** tìm nghiệm gần đúng của hệ phương trình:

$$\begin{cases} 4x_1 + 0.24x_2 - 0.08x_3 = 8 \\ 0.09x_1 + 3x_2 - 0.15x_3 = 9 \\ 0.04x_1 - 0.08x_2 + 4x_3 = 20 \end{cases}$$

Giải. (1).Có thể thấy rằng ma trận các hệ số của hệ phương trình trên đây thỏa mãn tính chéo trội, do đó ta có thể biến đổi hệ này để áp dụng phương pháp lặp Jacobi. Chia hai vế phương trình đầu tiên cho 4, hai vế phương trình thứ hai cho 3 và hai vế của phương trình thứ ba cho 4 rồi biến đổi thích hợp ta nhận được:

$$\begin{cases} x_1 = 2 - 0.06x_2 + 0.02x_3 \\ x_2 = 3 - 0.03x_1 + 0.05x_3 \\ x_3 = 5 - 0.01x_1 + 0.02x_2 \end{cases}$$

hay

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{bmatrix} 0 & -0.06 & 0.02 \\ -0.03 & 0 & 0.05 \\ -0.01 & 0.02 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix} = Cx + d$$

$$\|C\|_\infty = \max(0 + 0.06 + 0.02, 0.03 + 0 + 0.05, 0.01 + 0.02 + 0) =$$

$$= \max(0.08, 0.08, 0.03) = 0.08 < 1$$

(2) Chọn $x^{(0)} = (2,3,5)^T$, rồi tính $x^{(1)}, x^{(2)}$,... theo công thức (2.11) với lưu ý $a_{ii} = 1$ ta được bảng kết quả sau:

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	2	3	5
1	1.92	3.1924	5.044648
2	1.9093489	3.194952	5.0448056
3	1.909199	3.1949643	5.0448073

Xem $x^{(3)}$ là nghiệm gần đúng cần tìm, ta có thể đánh giá sai số phạm phải của $x^{(3)}$ theo(2.12): $\|x^{(k)} - x^*\| \le \dfrac{\mu}{1-\mu}\|x^{(k)} - x^{(k-1)}\|$

Trong đó:

$$\|x^{(3)} - x^{(2)}\|_\infty = \max_i |x_i^{(3)} - x_i^{(2)}| = \max(0.0001499, 0.000123, 0.0000017) = 0.0001499$$

$$\mu = \max_i \frac{q_i}{1-p_i} = \max(0.08, 0.0515463, 0) = 0.08$$

Như vậy

$$\|x^{(3)} - x^*\|_\infty \le \frac{\mu}{1-\mu}\|x^{(k)} - x^{(k-1)}\| \le \frac{0.08}{1-0.08}\ 0.00001499 \approx 0.000013$$

Thuật toán Jacobi cũng tương tự như thuật toán Gauss-Seidel, nhưng thuật toán Gauss - Seidel có tốc độ hội tụ nhanh hơn.

Chương trình minh họa.

Sau đây là đoạn chương trình chính thể hiện (mô tả) thuật toán *lặp Gauss - Seidel*

```
/*Giai he phuong trinh tuyen tinh dung lap Gauss-Seidel, ma tran vuong n,
  cac phan tu cot thu n+1 la vecto b*/
//================================================
double kcach(double *x,double *y,int n)
 {double tmp=0;
  for(int i=1;i<=n;i++) tmp=tmp+fabs(x[i]-y[i]);
  return tmp;
 }
//================================================
int cheotroi(kmatran a, int n)
 {double tmp;int i,j;
  for(i=1;i<=n;i++)
   {tmp=0;
    for(j=1;j<=n;j++) {if(j!=i) tmp=tmp+fabs(a[i][j]);}
    if(fabs(a[i][i])<=tmp) return false;
   }
  return true;
 }
//================================================
/*Giai he phuong trinh tuyen tinh bang phep lap Gauss-Seidel.
  Tra ve true neu co nghiem */
int gseidel(kmatran aa,double *x,int n)
{int h,i,j,k;double tmp;kvecto z;kmatran a;
 int n1=n+1;
 for(i=1;i<=n;i++)
 for(j=1;j<=n1;j++) a[i][j]=aa[i][j];
 if(!cheotroi(a,n))
  {cout<<endl<<"Khong phai cheo troi";delay(1000);return false;}
 for(i=1;i<=n;i++)  //chuyen ve dang he so a[i][i] == 1
  {tmp=a[i][i];
   for(j=1;j<=n1;j++) a[i][j] = a[i][j]/tmp;
  }
 //Vong lap cac buoc khu
```

```
for(i=1;i<=n;i++) {x[i]=0;z[i]=0;}
k=1;
while(true)
 {for(i=1;i<=n;i++)
  {tmp=0;
   for(j=1;j<=n;j++) if(j!=i) tmp+=a[i][j]*x[j];
   x[i] = a[i][n+1]-tmp;
  }
  k++;
  if(kcach(x,z,n)<epsi) break;
  if(k>kmax)
   {cout<<endl<<"Phep lap chua hoi tu";delay(1000);return(false);}
  //Gan z = x va chuan bi sang vong lap tinh x
  for(i=1;i<=n;i++) z[i]=x[i];
 }
//Thu lai
kvecto bb;
for(i=1;i<=n;i++)
 {bb[i]=aa[i][1]*x[1];
  for(j=2;j<=n;j++) bb[i]+=aa[i][j]*x[j];
 }
//Dua ket qua vao tep ketqua
return true;
)
```

2.3. BÀI TẬP

Bài 1. Tính và kiểm tra bằng chương trình định thức của ma trận

$$A = \begin{bmatrix} 1 & 3 & 1 \\ 2 & 4 & 6 \\ 7 & 6 & 11 \end{bmatrix}$$

Bài 2. Tìm và kiểm tra bằng chương trình nghịch đảo của ma trận

$$A = \begin{bmatrix} 2 & 3 & 1 \\ -1 & 2 & -1 \\ 3 & 0 & 2 \end{bmatrix}$$

Bài 3. Tìm nghiệm hệ phương trình

$$\begin{cases} 2x_1 + 3x_2 + x_3 = 11 \\ -x_1 + 2x_2 - x_3 = 0 \\ 3x_1 + 2x_3 = 9 \end{cases}$$

Bằng phương pháp khử Gauss và Jordan. Kiểm tra bằng chương trình.

Bài 4. Giải bằng các phương pháp khử Gauss, khử Gauss-Jordan, phương lặp Jacobi và lặp Gauss-Seidel (nếu thỏa mãn điều kiện) hệ phương trình sau:

$$A = \begin{bmatrix} 17 & 65 & -13 & 50 \\ 12 & 16 & 37 & 28 \\ 56 & 23 & 11 & -19. \\ 3 & -5 & 47 & 10 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 84 \\ 25 \\ 36 \\ 18 \end{pmatrix}$$

Kiểm tra trên máy tính và thông báo về khả năng giải được hay không các phương pháp trên.

Bài 5. Giải bằng các phương pháp lặp hệ phương trình sau:

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 9 \\ 2x_1 + 20x_2 - 2x_3 = -44 \\ -2x_1 + 3x_2 + 10x_3 = 22 \end{cases}$$

TÓM TẮT NỘI DUNG CHƯƠNG 2

Trong chương này sinh viên cần nắm vững ít nhất là các vấn đề sau:

1. Phương pháp trực tiếp giải hệ phương trình tuyến tính

a.Phương pháp khử Gauss

Phương pháp khử Gauss dùng cách khử dần các ẩn để đưa hệ phương trình đã cho về một dạng tam giác trên rồi giải hệ tam giác này từ giới lên trên, không phải tính một định thức nào.

b. Phương pháp khử Gauss-Jordan

Phương pháp khử Gauss-Jordan dùng cách khử dần các ẩn để đưa hệ phương trình đã cho về một dạng ma trận đường chéo rồi giải hệ phương trình này, không phải tính một định thức nào.

2. Phương pháp lặp giải hệ phương trình tuyến tính

a. Phương pháp lặp đơn

- Giả sử phải tìm nghiệm gần đúng của hệ phương trình tuyến tính (2.1) có dạng Ax=b. Đối với phương pháp lặp đơn, nói chung chúng ta phải đưa hệ (2.1) về dạng x=Cx+d.Trong đó ma trận C và vec tơ d được xây dựng từ A và b. Ma trận phải thoả mãn điều kiện ||C||<1.

Để thực hiện phép lặp ta chọn một vec tơ ban đầu $x^{(0)}$, sau đó tính các $x^{(i)}$, i =1,2,... theo công thức lặp sau: $x^{(i)} = Cx^{(i-1)} + d$ cho tới khi nào thảo mãn điều kiện dừng.

- Sai số của phương pháp:

$$||x^{(k)} - x^*|| \le \frac{||C||}{1-||C||}||x^{(k)} - x^{(k-1)}||$$

hoặc

$$||x^{(k)} - x^*|| \le \frac{||C||^k}{1-||C||}||x^{(1)} - x^{(0)}||$$

b. Phương pháp lặp Jacobi

- Giả thiết ma trận A có tính chéo trội. Phương pháp lặp Jacobi sẽ có các bước lặp như :

Với vec tơ $x^{(0)}$ cho trước bất kỳ, ví dụ $x^{(0)} = \theta$ (vec tơ 0) ta có thể tính các vec tơ $x^{(k)}$ tại bước lặp k bằng công thức $x^{(k)} = C\,x^{(k-1)} + d$, k =1, 2, ...

Cụ thể hơn, nếu $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)})$ thì ta có

$$\begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ . \\ x_n^{(k)} \end{pmatrix} = - \begin{bmatrix} 0 & \frac{a_{12}}{a_{11}} & \ldots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 0 & \ldots & \frac{a_{2n}}{a_{22}} \\ . & . & \ldots & . \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \ldots & 0 \end{bmatrix} \begin{pmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \\ . \\ x_n^{(k-1)} \end{pmatrix} + \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ . \\ \frac{b_n}{a_{nn}} \end{pmatrix}$$

Với từng thành phần $x_i^{(k)}$ ta có

$$x_i^{(k)} = -\sum_{j-1, j\neq i}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k-1)} + \frac{b_i}{a_{ii}} = \frac{1}{a_{ii}}(b_i - \sum_{j-1, j\neq i}^{n} a_{ij}\, x_j^{(k-1)})$$

$i = 1, 2, \ldots, n, k = 1,2,...$

- Điều kiện hội tụ, đánh gái sai số của phương pháp lặp Jacobi cũng giống với phương pháp lặp đơn.

c.Phương pháp lặp Gauss – Seidel

- Giả thiết ma trận A có tính chéo trội. Phương pháp lặp ***Gauss - Seidel*** sẽ có các bước lặp như sau:

Với vec tơ $x^{(0)}$ cho trước bất kỳ, ví dụ $x^{(0)} = \theta$ (vec tơ 0) ta có thể tính các vec tơ $x^{(k)}$ tại bước lặp k bằng công thức :

$$x_i^{(k)} = \frac{1}{a_{ii}}(b_i - (\sum_{j-1}^{i-1} a_{ij}x_j^{(k)} + \sum_{j-i+1}^{n} a_{ij}x_j^{(k-1)}))$$

$i = 1, 2, \ldots, n, k = 1,2,...$

- Đánh giá sai số:

$$p_i = \sum_{j=1}^{i-1} |c_{ij}|,\ q_i = \sum_{j=i}^{n} |c_{ij}|\ ,\ \mu = \max_i \frac{q_i}{1-p_i}$$

Khi đó ta có:

$$||x^{(k)} - x^*|| \leq \frac{\mu}{1-\mu} ||x^{(k)} - x^{(k-1)}||$$

hoặc

$$||x^{(k)} - x^*|| \leq \frac{\mu^k}{1-\mu} ||x^{(1)} - x^{(0)}||$$