Modul Praktikum Fisika Komputasi I

disusun Oleh : Yudha Arman

Program Studi Fisika

Fakultas Matematika dan Ilmu Pengetahuan Alam

Untiversitas Tanjungpura

Pontianak

2018

Modul I. Penjumlahan dan Pengurangan Matriks

Dasar teori

Matriks terdiri dari susunan angka-angka (elemen-elemen) berbentuk kotak yang dinyatakan oleh sebuah lambang tunggal.

$$[A]=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Himpunan elemen yang mendatar dinamakan baris dan himpunan tegak dinamakan kolom. Variabel pertama, i selalu menunjuk nomor baris tempat elemen itu terletak. Variabel kedua j selalu menunjuk pada nomor kolom. Misalnya elemen a_{23} berada di baris 2 dan kolom 3. Matriks A di atas mempunyai m baris dan n kolom dan dikatakan berukuran m kali n (atau m x n) dan biasa disebut sebagai matriks dengan ukuran m kali n. Matriks dengan ukuran m = 1 biasa disebut sebagai vektor baris, sedangkan matriks-matriks dengan ukuran kolom n =1 biasa disebut sebagai vektor kolom. Matriks-matriks dengan ukuran m = n disebut matriks bujur sangkar. Elemen diagonal pada matriks ditandai dengan variabel penunjuk kolom dan baris yang memiliki angka yang sam, misalnya a_{11}, a_{22}, a_{33}, a_{44} ..$a_{kk.}$ Dua matriks *m x n* adalah sama, jika dan hanya jika setiap elemen pada matriks pertama sama dengan setiap elemen pada matriks yang kedua; yaitu [A] = [B] jika $a_{ij} = b_{ij}$ untuk semua i dan j.

Matriks bujur sangkar secara umum ditemukan pada saat menyelesaikan permasalahan sistem persamaan linier. Untuk permasalahan tersebut, banyaknya persamaan (berhubungan dengan baris-baris) dan banyaknya bilangan tak diketahui (berhubungan dengan kolom-kolom) harus sama agar solusi penyelesaian bersifat unik.

Terdapat sejumlah bentuk khas matriks bujur sangkar yang biasa ditemukan, yaitu

1. *Matriks simetri, yaitu* matriks dengan $a_{ij} = a_{ji}$ untuk semua i dan j.
2. *Matriks diagonal* adalah matriks bujur sangkar dimana semua elemen bukan diagonal sama dengan nol.
3. *Matriks satuan* adalah matriks diagonal dimana semua elemen pada diagonal sama dengan satu.
4. *Matriks segitiga atas* adalah matriks dimana semua elemen di bawah diagonal utama adalah nol.
5. *Matriks segitiga bawah* adalah matriks dimana semua elemen di atas diagonal utama adalah nol.
6. *Matriks pita* adalah matriks dimana semua elemen sama dengan nol kecuali pada suatu pita yang berpusat pada diagonal utama. Contohnya adalah matriks berikut ini.

$$[A]=\begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix}$$

Matriks tersebut mempunyai lebar pita 3 dan biasa disebut dengan – *matriks tridiagonal*

Operasi Matriks

Matriks transpose adalah matriks yang memiliki elemen baris dan kolom yang berpadanan dengan elemen kolom dan baris matriks asal, atau jika matriks B adalah transpose dari matriks A, maka $B_{ji} = A_{ij}$ dimana i=1:m dan j=1:n. Dalam notasi Algoritma ditulis sebagai :

Algoritma Transpose Matriks

Masukan : A (m,n ← ukuran A)
Keluaran : B ← A^T
Langkah :

```
Untuk i = 1 : m
    Untuk j = 1 : n
        B(j,i) = A(i,j)
```

dalam bahasa Matlab dituliskan sebagai berikut :

```
clc;clear all;
A=[.......];[m,n]=size(A);
for i=1:m
    for j=1:n
        B(j,i)=A(i,j);
    end
end;B
```

Penambahan dan pengurangan dua matriks, misal matriks A dan B, dilakukan dengan menambah atau mengurangi elemen yang saling berpadanan pada masing-masing matriks. Elemen-elemen matriks yang dihasilkan C dihitung sebagai :

$C_{ij} = A_{ij} \pm B_{ij}$ untuk i = 1,2,3,…,m , dan j = 1,2,3,…,n, atau dengan aturan penulisan algoritma dituliskan sebagai

Algoritma Penjumlahan/Pengurangan Matriks

Masukan : A (m,n ← ukuran A), B(p,q ← ukuran B)
Keluaran : C ← A ± B
Langkah :

Jika m≠p atau n≠q maka 'Kedua Matriks tidak dapat dijumlahkan atau diselisihkan', selesai

```
Untuk i = 1 : m
    Untuk j = 1 : n
        C(i,j) = A(i,j) ± B(i,j)
```

dalam bahasa Matlab dituliskan sebagai :

```
clc;clear all;
A=[.......];B=[......];[m,n]=size(A);
for i=1:m
    for j=1:n
        C(i,j)=A(i,j)± B(i,j);
    end
end;C
```

Operasi perkalian matriks A dengan sebuah skalar g (dengan matriks hasil adalah matriks C) dilakukan dengan mengalikan setiap elemen A dengan g, yaitu $C_{ij} = g * A_{ij}$, dengan i=1:m dan j=1:n.

Perkalian antara dua buah matriks dituliskan sebagai C = A* B, dimana elemen – elemen matriks C dituliskan sebagai

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k}.B_{k,j}$$

dengan n adalah ukuran kolom matriks A yang sama dengan ukuran baris matriks B. Berikut adalah algoritma untuk perkalian matriks :

Algoritma Perkalian Matriks

Masukan : A ((m,n) ← ukuran A), B((p,q) ← ukuran B)
Keluaran : C ← A x B
Langkah :

Jika m≠p maka 'Kedua Matriks tidak dapat dikalikan', selesai
Untuk i = 1 : m
  Untuk j = 1 : q
    C(i,j) = 0
    Untuk k = 1 : n
      C(i,j) = C(i,j) + A(i,k)*B(k,j)

dalam bahasa Matlab dituliskan sebagai :

```
clc;clear all;
A=[.......];B=[......]; [m,n]=size(A); [p,q]=size(B);
for i=1:m
    for j=1:q
        C(i,j)=0;
        for k=1:n
            C(i,j)=C(i,j)+(A(i,k)* B(k,j);
        end
    end
end;C
```

Perkalian matriks dengan vektor kolom dilakukan dengan memasukkan nilai *q* dengan 1. Begitu pula dengan operasi perkalian matriks baris dengan matriks lengkap semula dengan memasukkan nilai m dengan 1.

Langkah Praktikum Modul I

1. Buka aplikasi Matlab
2. Buka M-File baru
3. Pelajari Algoritma yang tertulis di modul
4. Ketik Kode Program transpose, penjumlahan dan pengurangan matriks yang terdapat di modul ini
5. Masukkan matriks A dan B sesuai yang diinstruksikan oleh Asisten
6. Jalankan program dan analisis hasilnya

Modul II. Sistem Persamaan Linier

Dalam keseharian, dalam berbagai bidang sains maupun matematika, sering dijumpai sejumlah permasalahan yang merupakan sebuah set dari suatu sistem persamaan linier. Sebagai contoh adalah pada rangkaian elektronika dan pada bidang mekanika. Set persamaan tersebut harus diselesaikan secara simultan. Hal ini sekilas terlihat seperti aljabar, tapi dilengkapi dengan taksiran geometri. Taksiran geometri ini memungkinkan untuk dilakukannya analisis pada solusi lebih mendalam.

Dari hal tersebut di atas terlihat diperlukannya formulasi vektor untuk mempelajari set persamaan simultan tersebut. Keuntungan yang diperoleh dari formulasi vektor adalah permasalahan yang ditinjau tidak bergantung pada pemilihan sistem koordinat. Formulasi vektor juga ekivalen dengan jumlah dimensi permasalahan. Sebagai contoh adalah jumlah dari ukuran kolom vektor ekivalen dengan dimensi ruang permasalahan.

Pencarian solusi sebuah set sistem persamaan linier sederhana dapat dilakukan dengan metode substitusi maupun eliminasi. Namun, untuk permasalahan yang lebih kompleks diperlukan proses yang lebih sistematik. Dalam modul ini dibahas metode eliminasi yang lebih sistematik dan metode yang berbasiskan hubungan rekursif.

II.1. Eliminasi Gauss Penumpuan parsial

Eliminasi bilangan tak diketahui digunakan untuk menyelesaikan sebuah set persamaan. Prosedurnya terdiri dari dua langkah, yaitu :

1. Set persamaan dioperasikan untuk menghilangkan salah satu bilangan tak diketahui dari set persamaan tersebut. Hasil dari langkah eliminasi ini adalah diperolehnya satu persamaan dengan satu bilangan tak diketahui.
2. Akibatnya persamaan ini dapat langsung diselesaikan dan hasilnya disubstitusikan kembali ke salah satu persamaan semula untuk menyelesaikan bilangan tak diketahui yang tersisa.

Pendekatan dasar ini dapat diperluas ke himpunan persamaan yang lebih besar dengan cara mengembangkan skema bersistem untuk menghilangkan bilangan tak diketahui dan melakukan substitusi mundur. Eliminasi Gauss merupakan metode yang paling umum dari metode-metode lain yang menggunakan skema ini.

Pendekatannya dibangun untuk menyelesaikan suatu himpunan n persamaan yang umum berikut.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ \dots\ + a_{1n}x_n &= c_1 \\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ \dots\ + a_{2n}x_n &= c_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ \dots\ + a_{3n}x_n &= c_3 \\
\dots\quad \dots\quad \dots\quad \dots\quad \dots\quad \dots & \\
a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ \dots\ + a_{nn}x_n &= c_n
\end{aligned}
$$

Metode yang digunakan untuk n persamaan terdiri dari dua tahap, yaitu tahap eliminasi bilangan-bilangan tak diketahui dan penyelesaian melalui langkah substitusi mundur.

1. Tahap Eliminasi Maju

   Tahap ini dibuat untuk mereduksi sistem persamaan ke sistem dengan bentuk matriks segitiga atas. Langkah awal adalah dengan menghilangkan bilangan tak diketahui pertama x_1 dari persamaan ke dua sampai n. Untuk melakukan ini, kalikan persamaan 1 dengan a_{21} / a_{11} untuk memberikan :

$$a_{21}x_1 + \frac{a_{21}}{a_{11}}a_{12}x_2 + \ldots + \frac{a_{21}}{a_{11}}a_{1n}x_n = \frac{a_{21}}{a_{11}}c_1$$

   Sekarang persamaan ini dapat dikurangkan dari persamaan baris ke 2 untuk memberikan

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \ldots + \left(a_{2n} - \frac{a_{21}}{a_{11}}a_{n2}\right)x_n = c_2 - \left(\frac{a_{21}}{a_{11}}\right)c_1$$

   atau

$$a_{22}'x_2 + a_{23}'x_3 + \ \dots\ + a_{2n}'x_n = c_2'$$

Hasil lengkap dari proses ini dapat dituliskan sebagai *:*

$$
\begin{array}{lllllllll}
a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \ldots + & a_{1n}x_n & = & c_1 \\
 & & a_{22}'x_2 & + & a_{23}'x_3 & + & \ldots + & a_{2n}'x_n & = & c_2' \\
 & & a_{32}'x_2 & + & a_{33}'x_3 & + & \ldots + & a_{3n}'x_n & = & c_3' \\
 & & .. & & .. & & .. & .. & .. \\
 & & a_{n2}'x_2 & + & a_{n3}'x_3 & + & \ldots + & a_{nn}'x_n & = & c_n'
\end{array}
$$

Langkah berikutnya adalah mengulangi prosedur yang sama untuk persamaan berikutnya. Persamaan pada baris pertama ini merupakan persamaan *tumpuan* dan a_{11} disebut sebagai *koefisien tumpuan*. Operasi terakhir dalam proses eliminasi adalah dengan menggunakan persamaan ke (n-1) sebagai tumpuan untuk menghilangkan suku x_{n-1} dari persamaan ke *n*. Pada posisi ini, sistem akan bertransformasi ke bentuk matriks segitiga atas.

$$
\begin{array}{lllllllll}
a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & \ldots + & a_{1n}x_n & = & c_1 \\
 & & a_{22}'x_2 & + & a_{23}'x_3 & + & \ldots + & a_{2n}'x_n & = & c_2' \\
 & & & + & a_{33}''x_3 & + & \ldots + & a_{3n}''x_n & = & c_3'' \\
 & & & & .. & & .. & .. & & .. \\
 & & & & & & & a_{nn}^{(n-1)}x_n & = & c_n^{(n-1)}
\end{array}
$$

2. Substitusi mundur

   Sistem yang telah diperoleh kemudian digunakan untuk mendapatkan x_n, yaitu

$$x_n = \frac{c_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

   Hasil tersebut kemudian dapat disubstitusikan mundur ke persamaan ke (n-1) untuk mendapatkan x_{n-1}. Prosedur yang sama diulangi untuk mendapatkan solusi x lainnya yang tersisa, yang dapat dituliskan sebagai :

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

   Untuk *i = n-1, n-2, …, 1.*

Algoritma Eliminasi Gauss

Masukan : A (m,n ← ukuran A),C
Keluaran : x(i) dengan i = 1 : m
Langkah :

I. Pembentukan matriks Augmented A ← [A C]

II. Tahap Eliminasi

Untuk i = 1,2 ... (n-1)
- ℓ = i
- Untuk j = (i+1) : n
  - Jika | a(j,i) | > | a(ℓ,i) | maka ℓ = j
- Jika $\ell \neq$ i
- Untuk k = 1 : (n+1)
  - s ← a(i,k)
  - a(i,k) ← a(ℓ,k)
  - a(ℓ,k) ← s
- Jika a(i,i) = 0 maka ' SPL tidak dapat diselesaikan '
- Untuk t = i+1, i+2 ... n
  - $p = a_{ti} / a_{ii}$
  - Untuk q = 1: n+1
    - a (t,q) = a (t,q) – p*a(i,q)

III. Substitusi balik

$x_n = a_{n,n+1} / a_{nn}$
Untuk *i =n-1,n-2,…, 1*
- *D = 0*
- Untuk *j = i +1,i +2,…, n*
  - *D = D+(a_{ij} x_j)*
- $x_i = \dfrac{a_{i,n+1} - D}{a_{ii}}$

Dalam Bahasa Matlab dituliskan sebagai berikut:

```
clear all;clc;
a=[...];c=[...]; [m,n]=size(a);
a=[a c] % matriks Augmented
for k=1:(n-1)
    l=k;
    for i=k+1:n
        if abs(a(i,k))>abs(a(l,k));l=i;end;
    end
    if l>k
        for j=1:(n+1)
            s=a(l,j);a(l,j)=a(k,j);a(k,j)=s;
        end
    end
    for i=k+1:n
        p=a(i,k)/a(k,k);
        for j=k:(n+1)
            a(i,j)=a(i,j)-(p*a(k,j));
        end
    end
end

%  Substitusi balik
x(n)=a(n,n+1)/a(n,n);
for i=(n-1):-1:1
    D=0;
    for j=(i+1):1:n
        D=D+(a(i,j)*x(j));
    end
    x(i)=(a(i,n+1)-D)/a(i,i);
end
disp('Hasil proses eliminasi Gauss ini adalah'); x
```

Langkah Praktikum

1. Buka aplikasi Matlab
2. Buka M-File baru
3. Pelajari Algoritma yang tertulis di modul
4. Ketik Kode Program Eliminasi Gauss penumpuan parsial yang terdapat di modul ini
5. Analisis persoalan fisis sesuai yang diberikan oleh Asisten
6. Definisikan SPL yang dibuat
7. Jalankan program dan analisis hasilnya

II.2. Eliminasi Gauss Jordan

Metode ini merupakan variasi dari metode eliminasi Gauss. Perbedaan utama adalah terletak pada eliminasi variabel tak diketahui lainnya untuk seluruh persamaan. Sebagai tambahan, setiap barisnya dinormalisasi dengan membagi baris tersebut dengan elemen pivotnya. Hasil dari eliminasi ini adalah matriks identitas. Konsekuensinya, tahap substitusi mundur tidak diperlukan. Jika matriks yang akan dieliminasi diberikan tambahan matriks identitas di sisi kanan matriks tersebut dan proses eliminasi juga dilakukan pada matriks tambahan ini maka hasil eliminasi tidak saja berupa matriks identitas, namun juga menghasilkan matriks invers dari matriks [A].

$$\left[\begin{array}{ccc|c|ccc} a_{11} & a_{12} & a_{13} & c_1 & 1 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & c_2 & 0 & 1 & 0 \\ a_{31} & a_{32} & a_{33} & c_3 & 0 & 0 & 1 \end{array}\right] \xrightarrow{eliminasi} \left[\begin{array}{ccc|c|c} 1 & 0 & 0 & & \\ 0 & 1 & 0 & \text{solusi} & A^{-1} \\ 0 & 0 & 1 & & \end{array}\right]$$

Algoritma Gauss – Jordan

Masukan : A, (m,n ← ukuran A), C
Keluaran : A^{-1} dan x(i) dengan i = 1 : m
Langkah :
Pembentukan matriks Augmented A ← [A I C]

```
Untuk k = 1,2 ... (n)
    ℓ = k
    Untuk j = (k+1) : n
    └── Jika | a(j,i) | > | a(ℓ,i) | maka ℓ = j
    Jika ℓ ≠ k
    Untuk i = 1 : (2n+1)
        s      ← a(k,i)
        a(k,i) ← a(ℓ,i)
        a(ℓ,i) ← s
    Jika a(k,k) = 0 maka ' SPL tidak dapat diselesaikan '
    Untuk i = k+1, k+2 ... 2n+1
    └── aki = aki / akk
    akk = 1
    Untuk i = 1: n
        Jika k ≠ i maka p = a(i,k)
        Untuk j = (k+1) : 2n+1
        └── a (i,j) = a (i,j) – p*a(k,j)
    x    = a(:,n+1)
    A-1 = a(:,n+2 ... 2n+1)
```

dalam bahasa Matlab dituliskan sebagai berikut:

```
clear all;clc;
a=[...];c=[...]; [m,n]=size(a); Id=eye(m,n);
a=[a c Id]; % matriks Augmented
for k=1:(n-1)
    l=k;
    for i=k+1:n
        if abs(a(i,k))>abs(a(l,k));l=i;end;
    end
    if l~=k
        for j=1:(n+1)
            s=a(l,j);a(l,j)=a(k,j);a(k,j)=s;
        end
    end
    a(k,k)=1;
    for i=1:n
        if i~=k
            p=a(i,k);
            for j=1:((2*n)+1)
                  a(i,j)=a(i,j)-(p*a(k,j));
            end
        end
    end
end
a
B=a(:,(n+2):(2*n+1);
disp('Hasil invers dari matriks ini adalah');B
disp('serta solusi dari SPL adalah');a(:,n+1)
```

Langkah Praktikum

1. Buka aplikasi Matlab
2. Buka M-File baru
3. Pelajari Algoritma yang tertulis di modul
4. Ketik Kode Program Eliminasi Gauss-Jordan yang terdapat di modul ini
5. Analisis persoalan fisis sesuai yang diberikan oleh Asisten
6. Definisikan SPL yang dibuat
7. Jalankan program dan analisis hasilnya

II.3. Metode Gauss Seidel

Metode Gauss Seidel merupakan metode iterasi yang paling umum digunakan. Asumsikan bahwa himpunan n persamaan linier (SPL) berbentuk seperti berikut :

$$[A]\{X\} = \{C\}$$

dan jika elemen-elemen diagonal semuanya tidak nol, persamaan pertama dapat diselesaikan untuk x_1, yang kedua untuk x_2, dan seterusnya sehingga jika dilakukan secara iteratif maka akan menghasilkan:

$$x_1^{\,k} = \frac{c_1 - a_{12}x_2^{\,k-1} - a_{13}x_3^{\,k-1} - \ldots - a_{1n}x_n^{\,k-1}}{a_{11}} = \frac{\left[c_1 - \sum_{j=1, j\neq 1}^{n} a_{ij}x_j^{\,k-1}\right]}{a_{11}}$$

$$x_2^{\,k} = \frac{c_2 - a_{21}x_1^{\,k} - a_{23}x_3^{\,k-1} - \ldots - a_{2n}x_n^{\,k-1}}{a_{22}} = \frac{\left[c_2 - a_{21}x_1^{\,k} - \sum_{j=3}^{n} a_{ij}x_j^{\,k-1}\right]}{a_{22}}$$

$$x_3^{\,k} = \frac{c_3 - a_{32}x_2^{\,k} - a_{31}x_1^{\,k} - \ldots - a_{3n}x_n^{\,k-1}}{a_{33}} = \frac{\left[c_3 - \sum_{j=1}^{2} a_{3j}x_j^{\,k} - \sum_{j=4}^{n} a_{3j}x_j^{\,k-1}\right]}{a_{33}}$$

... ...

$$x_n^{\,k} = \frac{c_n - a_{n1}x_1^{\,k} - a_{n2}x_2^{\,k} - \ldots - a_{n,n-1}x_{n-1}^{\,k-1}}{a_{nn}} = \frac{\left[c_n - \sum_{j=1}^{n} a_{nj}x_j^{\,k}\right]}{a_{nn}}$$

k = iterasi = 1,2,3,…, maksimum iterasi

Secara umum dapat dituliskan

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

dengan i = 1, 2, 3, …, n

Alternatif termudah untuk memperoleh terkaan-terkaan awal (x^0) adalah dengan mengasumsikan bahwa semuanya nol. Berikut adalah algoritma metode Gauss-Seidel :

Algoritma Gauss – Seidel

Masukan : A, (m,n ← ukuran A),C *maks, epsilon*, $x_0(i)$ dengan i = 1 : n
Keluaran : x(i) dengan i = 1 : n
Langkah :

untuk iterasi = 1, 2, 3, … maks
  galat = 0
  untuk i = 1, 2, 3, …, n
    $x_b = c_i$
    untuk j = 1, 2, 3, …, n
      Jika $j \neq i$ $x_b = x_b - a_{ij} x_j$
    $x_b = x_b / a_{ii}$
    selisih= $\left| \frac{x_b - x_i}{x_b} \right|$
    jika *selisih* > *galat* maka *galat* = *selisih*
    $x_i = x_b$
  jika galat < epsilon maka selesai
Proses belum Konvergen atau divergen
Selesai

Dalam bahasa Matlab dituliskan sebagai berikut :

```
clear all;clc;
A=[...];C=[...]; [m,n]=size(A); [p,q]=size(C);
X=zeros(1,n);
m_iterasi = ..;eps=1e-5;
galat = 0;
for iterasi = 1:m_iterasi
    for ii=1:n
        Xb =C(ii);
        for j=1:m
            if j~= ii ; Xb=Xb-A(ii,j)*X(j);end;
        end
        Xb=Xb/A(ii,ii);
        selisih=abs((Xb-X(ii)/Xb);
        if selisih > galat; galat=selisih;end;
        X(ii)=Xb;
    end
    if  galat<eps
        disp('Hasil adalah');Xb
        break
    end
end
if (iterasi=m_iterasi)
    if (galat>eps);
        disp('Hasil adalah');X
        disp('Namun proses divergen atau belum konvergen');
    end
end
```

Langkah Praktikum

1. Buka aplikasi Matlab
2. Buka M-File baru
3. Pelajari algoritma yang tertulis di modul
4. Ketik kode program Gauss-Seidel yang terdapat di modul ini
5. Analisis persoalan fisis sesuai yang diberikan oleh Asisten
6. Definisikan SPL yang dibuat
7. Jalankan program dan analisis hasilnya

II.4. Metode Jacobi

Perbedaan metode iteratif Jacobi dengan metode Gauss-Seidel adalah sifat hampirannya. Untuk metode Jacobi, nilai-nilai terkaan awal x_o dihampiri secara serempak, yaitu nilai perhitungan x_2 untuk iterasi pertama tidak menggunakan nilai hampiran terkaan awal dari x_1. Nilai x_2 iterasi pertama diperoleh langsung dari terkaan awal, yaitu dengan menggunakan nilai x_1 dan x_3 dari terkaan awal, bukan hasil dari iterasi pertama.

$$x_1^{\,k} = \frac{c_1 - a_{12}x_2^{\,k-1} - a_{13}x_3^{\,k-1} - \ldots - a_{1n}x_n^{\,k-1}}{a_{11}} = \frac{\left[c_1 - \sum_{j=2}^{n} a_{ij}x_j^{\,k-1}\right]}{a_{11}};\ j \neq 1$$

$$x_2^{\,k} = \frac{c_2 - a_{21}x_1^{\,k-1} - a_{23}x_3^{\,k-1} - \ldots - a_{2n}x_n^{\,k-1}}{a_{22}} = \frac{\left[c_2 - \sum_{j=1}^{n} a_{ij}x_j^{\,k-1}\right]}{a_{22}};\ j \neq 2$$

$$x_3^{\,k} = \frac{c_3 - a_{32}x_2^{\,k-1} - a_{31}x_1^{\,k-1} - \ldots - a_{3n}x_n^{\,k-1}}{a_{33}} = \frac{\left[c_3 - \sum_{j=1}^{n} a_{ij}x_j^{\,k-1}\right]}{a_{33}};\ j \neq 3$$

…

$$x_n^{\,k} = \frac{c_n - a_{n1}x_1^{\,k-1} - a_{n2}x_2^{\,k-1} - \ldots - a_{n,n-1}x_{n-1}^{\,k-1}}{a_{nn}} = \frac{\left[c_n - \sum_{j=1}^{n} a_{ij}x_j^{\,k-1}\right]}{a_{nn}};\ j \neq n$$

k adalah variabel yang menandakan jumlah iterasi, yaitu *iterasi = 1,2,3,…, maksimum iterasi*
Secara umum dapat dituliskan :

$$x_i^{\,k} = \frac{c_i - \sum_{\substack{j=i \\ j\neq i}}^{n} a_{ij}x_j^{\,k}}{a_{ii}}$$ dengan i = 1, 2, 3, …, n

Berikut adalah algoritma Metode Jacobi.

Algoritma Jacobi

Masukan : A, (m,n ← ukuran A),C, *maks, epsilon*, $x_0(i)$ dengan i = 1 : n
Keluaran : $x_0(i)$ dengan i = 1 : n
Langkah :

untuk iterasi = 1, 2, 3, … maks
  galat = 0
  untuk i = 1, 2, 3, …, n
    $x_i = c_i$
    untuk j = 1, 2, 3, …, n
      Jika $j \neq i$ $x_i = x_i - a_{ij} x_{0j}$
    $x_i = x_i / a_{ii}$
    $\text{selisih} = \left| \frac{x_i - x_{0i}}{x_{0i}} \right|$
    jika *selisih* > *galat* maka *galat* = *selisih*
  jika galat < epsilon maka selesai
  $x_{0i} = x_i$
Proses belum Konvergen atau divergen
Selesai

Dalam bahasa Matlab dituliskan sebagai berikut :

```
clear all;clc;
%input
A=[_ _ _;_ _ _;_ _ _];
[m,n]=size(A);
C=[_;_;_];
[p,q]=size(C);
%X1=0;X2=0;X3=0
X=[0 0 0];
maks_iterasi=2;
Eps=1e-5; %1.10^-5=0,00001
%Langkah pengerjaan
for iterasi=1:maks_iterasi
    galat=0;
    for ii=1:m
        Xb=C(ii);
        for j=1:n
            if j~=ii
                Xb=Xb-A(ii,j)*X(j);
            end
        end
        Xb=Xb/A(ii,ii);
        selisih=abs((Xb-X(ii))/Xb);
        if selisih > galat;
        galat=selisih;
        end
        X(ii)=Xb;
    end
    if galat < Eps
        disp('Hasil adalah');
        Xb
        break
    end
end
if(iterasi==maks_iterasi);
    if(galat>Eps);
    disp('Hasil adalah');
    X
    disp('Proses Divergen atau belum Konvergen')
    end
end
```

Langkah Praktikum

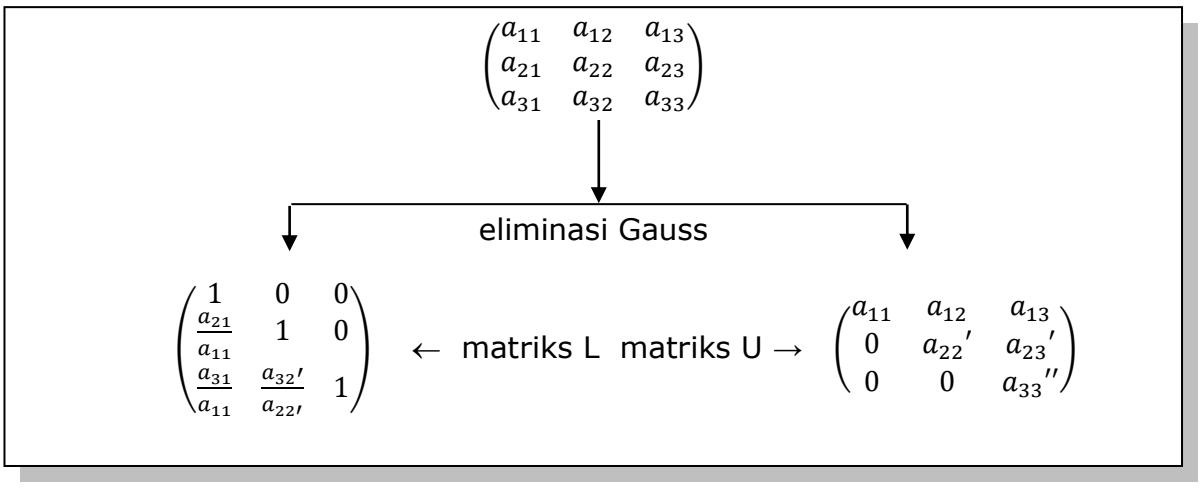
1. Buka aplikasi Matlab
2. Buka M-File baru
3. Pelajari Algoritma yang tertulis di modul
4. Ketik Kode Program Iteratif Jacobi yang terdapat di modul ini
5. Analisis persoalan fisis sesuai yang diberikan oleh Asisten
6. Definisikan SPL yang dibuat
7. Jalankan program dan analisis hasilnya

II.5. Metode Dekomposisi LU

Suatu matriks bujur sangkar A yang dapat didekomposisi menjadi matriks lain yang lebih sederhana, yaitu terdiri dari sebuah matriks segitiga bawah [L] dan matriks segitiga atas [U] dikatakan memiliki dekomposisi LU berupa matrik L dan matriks U tersebut. Matriks L dan U tidak unik karena terdapat beberapa metode yang dapat dilakukan untuk mendapatkan matriks L dan U tersebut, dan setiap metode tadi menghasilkan matriks L dan U yang berbeda. Metode-metode tersebut adalah :

1. Metode Crout
   Ciri dari metode Crout adalah elemen diagonal matriks U yang dihasilkan adalah 1 ($u_{ii} = 1$)
2. Metode Doolittle
   Metode Doolittle menghasilkan matriks L dengan nilai elemen diagonalnya adalah 1 ($l_{ii}=1$)
3. Metode Cholesky
   Metode ini menghasilkan matriks L dan matriks U yang memiliki nilai elemen diagonal yang sama ($u_{ii} = l_{ii}$)

Metode dekomposisi LU digunakan untuk menyelesaikan SPL. Metode ini memiliki efisiensi yang lebih baik dibandingkan dengan metode eliminasi Gauss maupun Gauss Jordan terutama pada saat menyelesaikan SPL dalam jumlah besar. Proses dekomposisi membuat penyelesaian SPL menjadi lebih sederhana karena pengulangan proses eliminasi Gauss yang tidak perlu dapat dihindarkan. Khusus untuk metode Doolittle, metode ini bisa dikatakan hanya melakukan setengah dari proses eliminasi Gauss, yaitu hanya pada tahap eliminasi. Hasil dari tahap eliminasi adalah matriks U, sedangkan matriks L adalah matriks dengan elemen diagonal 1 dan elemen segitiga bawah lainnya diisi oleh elemen pivot pada saat proses eliminasi dilakukan.



Apabila terdapat SPL seperti di bawah ini

$$Ax = C$$

Penyelesaian SPL dilakukan sebagai berikut :

1. dekomposisi matriks A menjadi matriks L dan U

$$LUx = C$$

2. dan misal

$$Ux = y$$

3. maka matriks SPL awal dapat dituliskan sebagai :

$$Ly = C$$

4. selesaikan matriks pada langkah 3 tersebut melalui proses substitusi yang sederhana (hal ini karena elemen diagonal matriks bernilai 1. Substitusi dilakukan secara maju)
5. lakukan substitusi mundur pada langkah 2 untuk mendapatkan solusi SPL.

Algoritma Dekomposisi LU Metode Doolittle

Masukan : A (m,n ← ukuran A),C
Keluaran : x(i) dengan i = 1,2,... m
Langkah :

```
% membentuk matriks L dan U
    Untuk i = 1,2 ... n
        Untuk j=1:n
            L(i,j)=0;
        jika i=j maka L(i,j)=1
    Untuk k = 1,2,... n-1
        Untuk i=(k+1):n
            p=a(i,k)/a(k,k)
            L(i,k)=p;
            Untuk j=k:n
                a(i,j)=a(i,j)-(p*a(k,j))
    U=a(:,1:n)

% subsitusi maju untuk mencari y dari hubungan Ly=C
    y1 = C(1)
    Untuk i =2,3,...n
        D = 0
        Untuk  j = i,i-1,...,1
            D = D+(Lij yj)
        yi = Ci − D

% subsitusi balik untuk mencari x dari hubungan Ux=y
    xn = yn / Unn
    Untuk i =n-1,n-2,…, 1
        D = 0
        Untuk  j = i +1,i +2,…, n
            D = D+(Uij xj)
        xi = (yi − D) / Uii
```

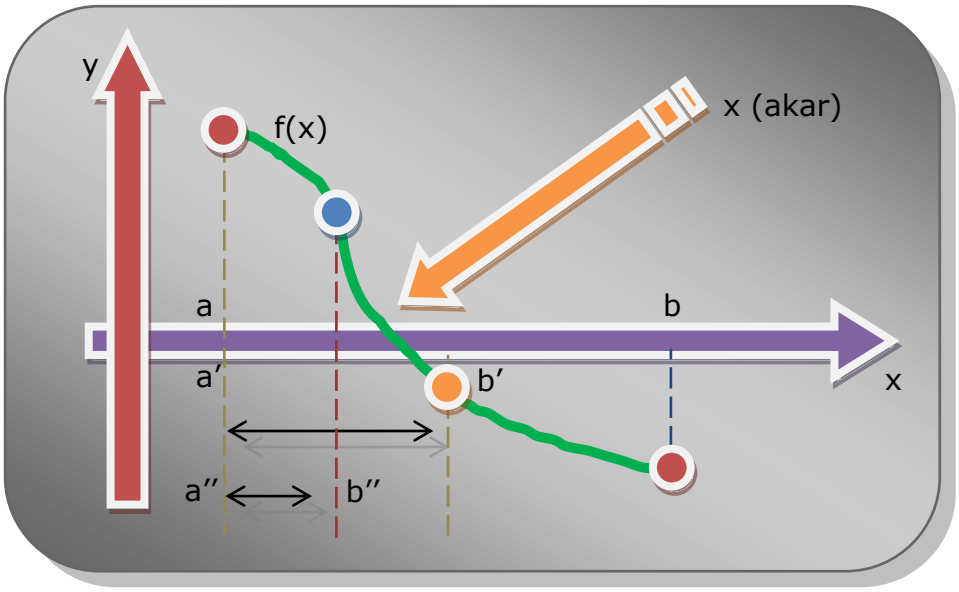
Dalam bahasa Matlab dituliskan sebagai berikut :

```
clear all;clc;
% input
a=[];c=[...];
[m,n]=size(a);
for i=1:n;
    for j=1:n
        L(i,j)=0;
        if i==j;
            L(i,j)=1;
        end
    end
end
for k=1:(n-1);
    for i=k+1:n
        p=a(i,k)/a(k,k);
        L(i,k)=p;
        for j=k:n
            a(i,j)=a(i,j)-(p*a(k,j));
        end
    end
end
U=a(:,1:n);
% subsitusi maju untuk mencari y dari hubungan Ly=C
y(1) = c(1);
for i=2:n
    D=0;
    for j=i:-1:1
        D=D+(L(i,j)*y(j));
    end
    y(i)=C(i)-D;
end
% subsitusi balik untuk mencari x dari hubungan Ux=y
x(n)=y(n)/U(n,n);
for i=n-1:-1:1
    D=0;
    for j=i+1:n
        D=D+(U(i,j)*x(j));
    end
    x(i)=(y(i)-D)/U(i,i);
end
disp('Matriks L dan U yang dihasilkan adalah');L
U
disp('Solusi SPL yang dihasilkan');x
```

Modul III. Pencarian Akar Persamaan Non Linier (*Root Finding*)

III.1. Metode Bagi Dua (Bisection Method)

Metode bagi dua yang juga dinamakan pemenggalan biner, pemaruhan selang atau metode bolzano merupakan salah satu jenis pencarian incremental dalam mana selang di bagi dua.. Jika suatu fungsi berubah tanda pada suatu selang, maka nilai fungsi dihitung pada titik tengah. Kemudian lokasi akar ditentukan sebagai terletak pada titik tengah selang bagian tempat terjadinya perubahan tanda. Prosesnya diulang untuk memperoleh taksiran yang diperhalus.



Berikut adalah algoritma metode bagi dua.

Algoritma Metode Bagi Dua

Masukan : *f(x)*

a,b dimana *f[a] .f[b] < 0*

Epsilon

Keluaran : Akar

Langkah-langkah :

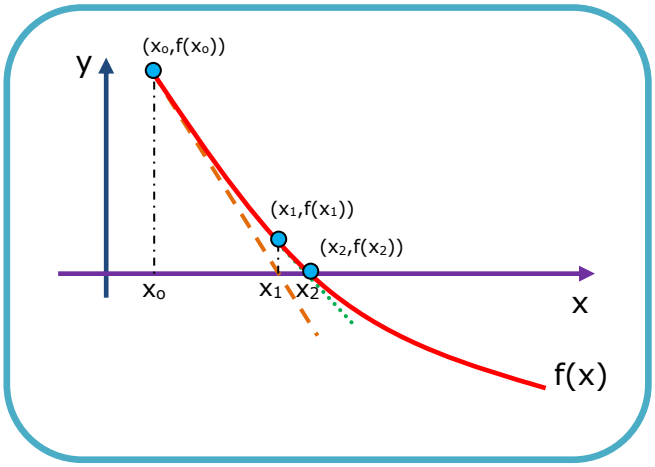
1. *T = [a + b] / 2*
2. jika *f[a].f[t] < 0* maka *b = T*; lainnya *a = T.*
3. Jika abs(*b-a) < epsilon* maka *akar = T* ; selesai.
4. kembali ke (1) jika belum selesai.

Tugas Praktikum : Lengkapi kode program pascal berikut untuk fungsi $f(x) = e^x - 4x$.

III.2. Metode Newton Raphson

Metode ini merupakan salah satu metode pencarian akar secara terbuka, yaitu tanpa selang yang menaungi lokasi akar. Kelemahan metode ini adalah pencarian akar tidak selalu konvergen.

Metode ini didasarkan pada sebuah fungsi f(x) non linier yang dihampiri oleh garis singgung yang menyinggung fungsi f(x). Titik potong garis singgung ini terhadap sumbu x merupakan hampiran akar yang baru. Evaluasi fungsi tersebut pada hampiran akar yang baru, kemudian dicari kembali persamaan garis singgung dari titik baru ini maka akan diperoleh hampiran akar yang baru. Proses ini diulang hingga sebuah nilai kesalahan tertentu diperoleh.



Persamaan garis singgung yang dimulai dari titik $(x_o,f(x_o))$ menyinggung fungsi f(x) adalah :

$$y - f(x_0) = m(x - x_0) \text{ dengan } m = f'(x_0)$$

Dalam bentuk lain dapat dituliskan sebagai :

$$y - f(x_0) = f'(x_0)\,(x - x_0)$$

Titik potong persamaan garis singgung ini terhadap sumbu x adalah $(x_1,0)$. Substitusi ke persamaan sebelumnya akan menghasilkan persamaan :

$$0 - f(x_0) = f'(x_0)\,(x_1 - x_0)$$

dan disusun ulang untuk mendapatkan :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

dengan cara yang sama diperoleh hampiran akar yang ke dua :

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

dan hampiran akar ke k

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Dilihat dari karakteristik persamaan yang dihasilkan, untuk suatu nilai *k* metode akan berhasil bila turunan fungsi pada x_k ($f'(x_k) \neq 0$.

Terdapat beberapa kriteria penghentian pada metode ini, namun untuk praktikum kali ini digunakan kriteria penghentian berikut :

1. $\left| \frac{x_{k+1} - x_k}{x_{k+1}} \right| < epsilon$
2. Pembatasan Interasi (hingga iterasi maksimum dimana akar konvergen)

Berikut adalah algoritma metode Newton Raphson

Algoritma Newton Raphson

Masukan : $f(x)$, $f'(x)$, x_o
epsilon, maks

Keluaran : Akar

Langkah-langkah :
I. Untuk iterasi = 1,2 3, … , maks
   1. Jika $f'(x_o) = 0$ maka "proses gagal", selesai.
   2. $x_{baru} = x_o - \frac{f(x_o)}{f'(x_o)}$
   3. Jika $\left| \frac{x_{baru1} - x_o}{x_{baru}} \right| < epsilon$ maka *akar* = x_{baru}, selesai.
   4. $x_o = x_{baru.}$

II. " Proses Divergen atau belum konvergen "
III. selesai.

III.3. Metode Newton Raphson Untuk Polinom

Pada Praktikum ini ditekankan pada penerapan metode Newton Raphson untuk menghitung akar dari suatu polinom berderajat n. Program yang digunakan mempunyai kriteria penghentian yang sama dengan program Newton Raphson sebelumnya, hanya penerapannya berbeda.

Bentuk umum hampiran Newton Raphson seperti yang telah dibahas sebelumnya adalah :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Sedangkan untuk hampiran polinom : $\boxed{x_{k+1} = x_k - \frac{p(x_k)}{p'(x_k)}} \approx \boxed{x_{k+1} = x_k - \frac{b_0}{c_1}}$

$P(x_k)$ dan $P'(x_k)$ dihitung menggunakan perkalian bersarang yang diilustrasikan seperti berikut ini :

z	a_n	a_{n-1}	a_{n-2}	…	a_1	a_o
	↓					
	b_n	b_{n-1}	b_{n-2}	…	b_1	b_o
	↓					
	c_n	c_{n-1}	c_{n-2}	…	c_1	

Algoritma NewtonRaphson untuk Polinom

Masukan : n, a_i, $i=0,1,2,3,\ldots,n$
x_o, epsilon, maks

Keluaran : Akar

Langkah-langkah : I. Untuk iterasi = 1,2 3, … , maks

- $b_n = a_n$
- $c_n = b_n$
- Untuk $i = n-1, n-2, \ldots, 1$
  - $b_i = a_i + x_o\, b_{i+1}$
  - $c_i = b_i + x_o\, c_{i+1}$
- $b_o = a_o + x_o\, b_1$
- Jika $c_1 = 0$ maka *" proses gagal ", selesai*
- $x_{baru} = x_o - \frac{b_0}{c_1}$
- jika $\left|\frac{x_{baru} - x_o}{x_{baru}}\right| < epsilon$ maka *akar* = x_{baru} ; selesai.
- $x_o = x_{baru}$

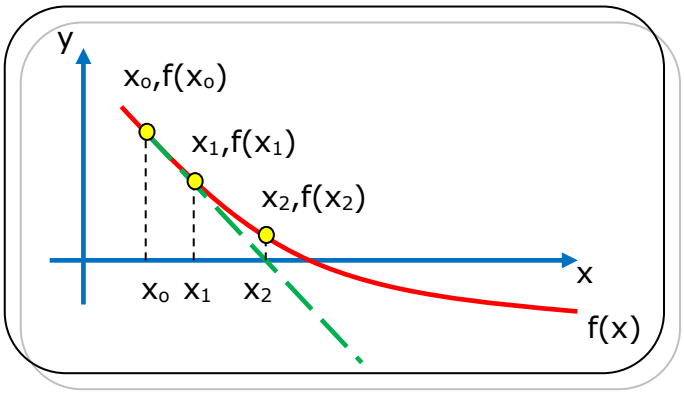
II. " Proses Divergen atau belum konvergen "

III. selesai

III.4. Metode Secant (Tali Busur)

Masalah potensial dalam menerapkan metode Newton-Raphson adalah evaluasi dari turunan pertama. Terdapat fungsi-fungsi yang mungkin turunannya sangat sulit untuk dievaluasi. Untuk fungsi seperti ini, turunan dapat dihampiri oleh metode beda hingga terbagi.

Hampiran ini dapat dijelaskan sebagai berikut. Tinjau sebuah titik x_{k+1} yang merupakan absis dari titik perpotongan terhadap sumbu *x* dari sebuah garis yang melalui titik $(x_{k-1},f(x_{k-1}))$ *dan* $(x_k,f(x_k))$. Taksiran geometri dari pernyataan tersebut digambarkan berikut ini.



Pada kasus dimana *f'(x)* sulit untuk dicari, digunakan hampiran untuk mencari turunan sebagai berikut :

$$f'(x) = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

dalam bentuk lain dapat dituliskan dalam bentuk hampiran berikut :

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Substitusi hampiran ke persamaan iteratif menghasilkan :

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{f(x_k)}{\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}}$$

k =1, 2, 3, …

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

Dari hampiran tersebut diketahui bahwa pogram Tali Busur yang akan dibuat memerlukan masukan x_o dan x_1 untuk menghitung x_2.

Berikut adalah algoritma metode secant (tali busur)

Algoritma Secant (Tali Busur)

Masukan : *f(x), x_o, x_1*
Epsilon, Maks

Keluaran : Akar

Langkah-langkah : I. Untuk iterasi = 1,2 3, … , maks

1. $x_{baru} = x_1 - f(x_1)\dfrac{x_1 - x_0}{f(x_1) - f(x_0)}$
2. Jika $\left|\dfrac{x_{baru} - x_o}{x_{baru}}\right| < epsilon$ maka *akar* = *xbaru*, selesai
3. $x_o = x_1$.
   $x_1 = x_{baru}$; selesai

IV. " Proses Divergen atau belum konvergen "
selesai.

IV.2. Integral Romberg.

Dasar Teori

Metode Ini didasarkan pada ekstrapolasi Richardson. Integral dilakukan secara iteratif dengan selang perhitungan yang berbeda-beda. Hasil dari setiap perhitungan kemudian digunakan untuk memperoleh hasil hampiran integral yang lebih baik.

Formulasi intgral Romber dimulai dengan menghampiri integral eksak I dengan metode Trapesium untuk selang h_1 dan h_2 yang berbeda.

$$I_e = I_e$$

$$I(h_1) + E(h_1) = I(h_2) + E(h_2)$$

Seperti diketahui bahwa Integral trapesium memiliki kesalahan hampiran sebesar :

$$E_T = -\frac{b-a}{12}h^2 f''(\xi)$$

dan dengan asumsi

$$f''(\xi_1) = f''(\xi_2)$$

maka

$$\frac{E(h_1)}{E(h_2)} \cong \frac{h_1^2}{h_2^2}$$

Substitusi ke persamaan sebelumnya menghasilkan :

$$I(h_1) + \frac{h_1^2}{h_2^2}E(h_2) = I(h_2) + E(h_2)$$

sehingga diperoleh

$$E(h_2) = \frac{I(h_2) - I(h_1)}{\left(\frac{h_1}{h_2}\right)^2 - 1}$$

Dari hasil hampiran kesalahan pada selang kedua ini diperoleh hampiran integral baru dalam bentuk :

$$I \cong I(h_2) + \frac{I(h_2) - I(h_1)}{\left(\frac{h_1}{h_2}\right)^2 - 1}$$

yang memiliki akurasi yang lebih baik. Hal ini terlihat dari kesalahan hampiran yang lebih kecil dari selang hampiran sebelumnya. Jika $h_2 = \frac{h_1}{2}$ maka

$$I = \frac{4I(h_2) - I(h_1)}{3}$$

Jika pendekatan ini dilanjutkan, maka orde kesalahan yang lebih kecil ($O(h^6)$) akan diperoleh, yaitu:

$$I = \frac{16I(h_2) - I(h_1)}{15}$$

kemudian akurasi $O(h^8)$ akan berbentuk :

$$I = \frac{64I(h_2) - I(h_1)}{63}$$

dan seterusnya hingga orde kesalahan akan mengecil dalam orde kesalahan berpangkat 2. Dalam bentuk umum hampiran intergral ini dapat dituliskan sebagai :

$$I_{j,k} = \frac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}$$

dengan orde kesalahan $O(h^{2k})$ dimana selang berikutnya adalah setengah dari selang sebelumnya.

Dalam bentuk tabel dapat diilustrasikan sebagai berikut :

j	k →			
↓	$O(h^2)$	$O(h^4)$	$O(h^6)$	$O(h^8)$
	I[1,1]	I[1,2]	I[1,3]	I[3,3]
	I[2,1]	I[2,2]	I[2,3]	Hampiran integral yang lebih baik
	I[3,1]	I[3,2]		
	I[4,1]			

Adapun Algoritma metode ini untuk menghitung integral :

$$I = \int_a^b f(x)dx$$

adalah sebagai berikut :

Masukan : a, b, f(x), k
Keluaran : I
Langkah :

n=1, h=b-a, x(0)=a, x(1)=b

$I(1,1) = \frac{h}{2}(f(x_0) + f(x_1))$

untuk s = 1 : k-1

```
    n=n*2
    h=h/2
    untuk r = 0 : n
    └  x(r) = x(0) + r*h
    sum = 0
    untuk q = 1 : n-1
    └  sum = sum + f(x(q))
└── I(s + 1,1) = h/2 (f(x_0) + 2 * sum + f(x_n))
untuk p = 2 : k
    untuk t = 1 : (k-p+1)
    │
└───  I(t,p) = (4^(p-1) I(t+1,p-1) − I(t,p-1)) / (4^(p-1) − 1)
```

Langkah praktikum :

1. Hitung Integral $\int_0^2 x^3 dx$ menggunakan metode Romberg
2. Hitung terlebih dahulu integral tersebut secara analitik
3. Salin Algoritma di atas ke dalam bahasa Matlab.
4. Analisa hasil dan lakukan perhitungan kesalahan hampiran.

DAFTAR PUSTAKA

Chapra. S., dan Canale,R.P., Numerical Methods For Engineers, McGraw Hill, 2009.

Jogiyanto, H.M., Pengenalan Komputer, Andi Offset, Yogyakarta. 1988

Jogiyanto, H.M., Turbo Pascal : Teori dan aplikasi Program Komputer Bahasa Turbo Pascal termasuk Databesa Toolbox, Jilid I, Andi, Yogyakarta, 2001

Munir, Rinaldi, Algoritma dan Pemrograman, edisi kedua,Informatika,Bandung. 1999.

Munir, Rinaldi, Metode Numerik, Informatika, Bandung, 2003.

Munadi, Suprajitno, Perhitungan Matriks dengan Fortran, Andi Offset, Yogyakarta, 1990.

Raltson, A. 1971 : Intoduction to Programming and Computer Science