Komputasi untuk Sains dan Teknik
-Menggunakan Matlab-

Supriyanto Suparno
(Website: http://supriyanto.fisika.ui.edu)
(Email: supri@fisika.ui.ac.id atau supri92@gmail.com)

Edisi III
Revisi terakhir tgl: 29 Juni 2010

Departemen Fisika-FMIPA, Univeristas Indonesia
Dipublikasikan pertama kali pada September 2007

Untuk
Nina Marliyani
Muflih Syamil
dan
Hasan Azmi

Usia bukan ukuran kedewasaan

(*Supriyanto, 2006*)

Ketekunan adalah jalan yang terpercaya untuk mengantarkan kita menuju kesuksesan

(*Supriyanto, 2007*)

Kata Pengantar

Alhamdulillah, buku ini memasuki edisi ke-3. Penomoran edisi ini sebenarnya hanya untuk menandakan perubahan isi buku yang semakin kaya metode numerik dibandingkan dengan edisi-edisi sebelumnya. Pengayaan isi buku ini, sejujurnya, berasal dari sejumlah pertanyaan yang sampai ke *mailbox* saya, entah itu dalam bentuk konsultasi Tugas Akhir mahasiswa S1 sebagaimana yang saya terima dari mahasiswa UNPAD, UDAYANA, UNESA dan UNSRI serta UI sendiri, ataupun sekedar pertanyaan seputar tugas kuliah seperti yang biasa ditanyakan oleh para mahasiswa dari Univ. Pakuan, Bogor.

Pertanyaan-pertanyaan itu menjadikan saya sadar bahwa buku edisi ke-II yang berjumlah 187 halaman, ternyata belum bisa memenuhi kebutuhan banyak mahasiswa yang memerlukan teknik pengolahan data secara numerik. Karenanya, *insya Allah*, pada edisi ke-III ini, saya mencoba menyempurnakan buku ini secara bertahap.

Buku ini mulai ditulis pada tahun 2005 dengan isi yang seadanya, pokoknya asal tercatat. Kemudian di tahun 2006-akhir buku ini menjadi catatan perkuliahan Komputasi Fisika. Pengayaan isi buku terus berlangsung hingga akhir 2007. Lalu di awal tahun 2008, isi buku ini ditambah dengan materi perkuliahan Analisis Numerik. Jadi materi Komputasi Fisika tahun 2007 dan materi Analisis Numerik 2008, telah digabung jadi satu dalam buku ini.

Secara garis besar, ilmu fisika dapat dipelajari lewat 3 jalan, yaitu pertama, dengan menggunakan konsep atau teori fisika yang akhirnya melahirkan fisika teori. Kedua, dengan cara eksperimen yang menghasilkan aliran fisika eksperimental, dan ketiga, fisika bisa dipelajari lewat simulasi fenomena alam yang sangat mengandalkan komputer serta algoritma numerik. Tujuan penyusunan buku ini adalah untuk meletakkan pondasi dasar dari bangunan pemahaman akan metode-metode komputasi yang banyak digunakan untuk mensimulasikan fenomena fisika.

Rujukan utama buku ini bersumber pada buku teks standar yang sangat populer di dunia komputasi, yaitu buku yang ditulis oleh Richard L. Burden dan J. Douglas Faires dengan judul *Numerical Analysis* edisi ke-7, diterbitkan oleh Penerbit Brooks/Cole, Thomson Learning Academic Resource Center. Disamping itu, buku ini dilengkapi oleh sejumlah contoh aplikasi komputasi pada upaya penyelesaian problem-problem fisika.

Pada edisi ke-3 ini saya mulai menfokuskan menulis *script* dalam lingkungan **Matlab**. Padahal, dalam edisi ke-2 yang lalu, *script* numerik disalin ke dalam 2 bahasa pemrograman, yaitu **Fortran77** dan **Matlab**. Namun mayoritas ditulis dalam **Matlab**.

Akhirnya saya ingin mengucapkan rasa terima kasih yang tak terhingga kepada Dede Djuhana yang telah berkenan memberikan format LaTeX-nya sehingga tampilan tulisan pada buku ini benar-benar layaknya sebuah buku yang siap dicetak. Tak lupa, saya pun berterima kasih kepada seluruh mahasiswa yang telah mengambil mata kuliah Komputasi Fisika PTA 2006/2007 di Departemen Fisika, FMIPA, Universitas Indonesia atas diskusi yang berlangsung

selama kuliah. Kepada seluruh mahasiswa dari berbagai universitas di Timur dan di Barat Indonesia juga saya ungkapkan terima kasih atas pertanyaan-pertanyaan yang turut memperkaya isi buku ini.

Walaupun buku ini masih jauh dari sempurna, namun semoga ia dapat menyumbangkan kontribusi yang berarti untuk kebangkitan ilmu pengetahuan pada diri anak bangsa Indonesia yang saat ini sedang terpuruk. Saya wariskan buku ini untuk siswa dan mahasiswa Indonesia dimanapun mereka berada. Anda berhak memanfaatkan buku ini. Saya izinkan anda untuk meng-*copy* dan menggunakan buku ini selama itu ditujukan untuk belajar dan bukan untuk tujuan komersial, kecuali kalau saya dapat bagian komisi-nya :) . Bagi yang ingin berdiskusi, memberikan masukan, kritikan dan saran, silakan dikirimkan ke email: **supri92@gmail.com**

Depok, 8 Juni 2008
Supriyanto Suparno

Daftar Isi

Daftar Gambar

Daftar Tabel

Bab 1

Matrik dan Komputasi

Objektif :

- ▷ Mengenalkan matrik, vektor dan jenis-jenis matrik.
- ▷ Mendeklarasikan elemen-elemen matrik ke dalam memori komputer.
- ▷ Mengenalkan operasi penjumlahan dan perkalian matrik.
- ▷ Membuat *script* operasi matrik.

1.1 Mengenal matrik

Notasi suatu matrik berukuran n x m ditulis dengan huruf besar dan dicetak tebal, misalnya $\mathbf{A}_{n\times m}$. Huruf n menyatakan jumlah baris, dan huruf m jumlah kolom. Suatu matrik tersusun atas elemen-elemen yang dinyatakan dengan huruf kecil lalu diikuti oleh angka-angka indeks, misalnya a_{ij}. Indeks *i* menunjukkan posisi baris ke-i dan indeks *j* menentukan posisi kolom ke-j.

$$\mathbf{A} = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \tag{1.1}$$

Pada matrik ini, a_{11}, a_{12}, ..., a_{1m} adalah elemen-elemen yang menempati baris pertama. Sementara a_{12}, a_{22}, ..., a_{n2} adalah elemen-elemen yang menempati kolom kedua.

Contoh 1: Matrik $\mathbf{A}_{2\times 3}$

$$\mathbf{A} = \begin{bmatrix} 3 & 8 & 5 \\ 6 & 4 & 7 \end{bmatrix}$$

dimana masing-masing elemennya adalah $a_{11} = 3$, $a_{12} = 8$, $a_{13} = 5$, $a_{21} = 6$, $a_{22} = 4$, dan $a_{23} = 7$.

Contoh 2: Matrik $\mathbf{B}_{3\times 2}$

$$\mathbf{B} = \begin{bmatrix} 1 & 3 \\ 5 & 9 \\ 2 & 4 \end{bmatrix}$$

dimana masing-masing elemennya adalah $b_{11} = 1$, $b_{12} = 3$, $b_{21} = 5$, $b_{22} = 9$, $b_{31} = 2$, dan $b_{32} = 4$.

1.2 Vektor-baris dan vektor-kolom

Notasi vektor biasanya dinyatakan dengan huruf kecil dan dicetak tebal. Suatu matrik dinamakan vektor-baris berukuran m, bila hanya memiliki satu baris dan m kolom, yang dinyatakan sebagai berikut

$$\mathbf{a} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \dots & a_m \end{bmatrix} \tag{1.2}$$

Sedangkan suatu matrik dinamakan vektor-kolom berukuran n, bila hanya memiliki satu kolom dan n baris, yang dinyatakan sebagai berikut

$$\mathbf{a} = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{1.3}$$

1.3 Inisialisasi matrik dalam memori komputer

Sebelum dilanjutkan, saya sarankan agar anda mencari tahu sendiri bagaimana cara membuat *m-file* di Matlab dan bagaimana cara menjalankannya. Karena semua *source code* yang terdapat dalam buku ini ditulis dalam *m-file*. Walaupun sangat mudah untuk melakukan *copy-paste*, namun dalam upaya membiasakan diri menulis *source code* di *m-file*, saya anjurkan anda menulis ulang semuanya.

Dalam Matlab terdapat 3 cara inisialisasi matrik. Cara pertama[1], sesuai dengan Contoh 1, adalah

```
clear all
clc

A(1,1) = 3;
A(1,2) = 8;
A(1,3) = 5;
A(2,1) = 6;
A(2,2) = 4;
A(2,3) = 7;
A
```

[1]Cara ini bisa diterapkan pada bahasa C, Fortran, Pascal, Delphi, Java, Basic, dll. Sementara cara kedua dan cara ketiga hanya akan dimengerti oleh Matlab

Sedangkan untuk matrik $\mathbf{B}_{3\times 2}$, sesuai Contoh 2 adalah

```
clear all
clc

B(1,1) = 1;
B(1,2) = 3;
B(2,1) = 5;
B(2,2) = 9;
B(3,1) = 2;
B(3,2) = 4;
B
```

Cara kedua relatif lebih mudah dan benar-benar merepresentasikan dimensi matriknya, dimana jumlah baris dan jumlah kolom terlihat dengan jelas.

```
clear all
clc

A=[ 3 8 5
    6 4 7 ];

B=[ 1 3
    5 9
    2 4 ];
```

Cara ketiga jauh lebih singkat, namun tidak menunjukkan dimensi matrik lantaran ditulis hanya dalam satu baris.

```
clear all
clc

A=[ 3 8 5 ; 6 4 7 ];
B=[ 1 3 ; 5 9 ; 2 4];
```

1.4 Macam-macam matrik

1.4.1 Matrik transpose

Operasi transpose terhadap suatu matrik akan menukar elemen-elemen kolom menjadi elemen-elemen baris. Notasi matrik tranpose adalah $\mathbf{A}^T$ atau $\mathbf{A}^t$.

Contoh 3: Operasi transpose terhadap matrik $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} 3 & 8 & 5 \\ 6 & 4 & 7 \end{bmatrix} \qquad \mathbf{A}^T = \begin{bmatrix} 3 & 6 \\ 8 & 4 \\ 5 & 7 \end{bmatrix}$$

Dengan Matlab, operasi transpose cukup dilakukan dengan menambahkan tanda petik tunggal di depan nama matriknya

```
clear all
clc

A=[ 3 8 5
    6 4 7 ];

AT = A';
```

1.4.2 Matrik bujursangkar

Matrik bujursangkar adalah matrik yang jumlah baris dan jumlah kolomnya sama.

Contoh 4: Matrik bujursangkar berukuran 3x3 atau sering juga disebut matrik bujursangkar orde 3

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 8 \\ 5 & 9 & 7 \\ 2 & 4 & 6 \end{bmatrix}$$

1.4.3 Matrik simetrik

Matrik simetrik adalah matrik bujursangkar yang elemen-elemen matrik transpose-nya bernilai sama dengan matrik asli-nya.

Contoh 5: Matrik simetrik

$$\mathbf{A} = \begin{bmatrix} 2 & -3 & 7 & 1 \\ -3 & 5 & 6 & -2 \\ 7 & 6 & 9 & 8 \\ 1 & -2 & 8 & 10 \end{bmatrix} \qquad \mathbf{A}^T = \begin{bmatrix} 2 & -3 & 7 & 1 \\ -3 & 5 & 6 & -2 \\ 7 & 6 & 9 & 8 \\ 1 & -2 & 8 & 10 \end{bmatrix}$$

1.4.4 Matrik diagonal

Matrik diagonal adalah matrik bujursangkar yang seluruh elemen-nya bernilai 0 (nol), kecuali elemen-elemen diagonalnya.

Contoh 6: Matrik diagonal orde 3

$$\mathbf{A} = \begin{bmatrix} 11 & 0 & 0 \\ 0 & 29 & 0 \\ 0 & 0 & 61 \end{bmatrix}$$

1.4.5 Matrik identitas

Matrik identitas adalah matrik bujursangkar yang semua elemen-nya bernilai 0 (nol), kecuali elemen-elemen diagonal yang seluruhnya bernilai 1.

Contoh 7: Matrik identitas orde 3

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

1.4.6 Matrik upper-triangular

Matrik upper-tringular adalah matrik bujursangkar yang seluruh elemen dibawah elemen diagonal bernilai 0 (nol).

Contoh 8: Matrik upper-triangular

$$\mathbf{A} = \begin{bmatrix} 3 & 6 & 2 & 1 \\ 0 & 4 & 1 & 5 \\ 0 & 0 & 8 & 7 \\ 0 & 0 & 0 & 9 \end{bmatrix}$$

1.4.7 Matrik lower-triangular

Matrik lower-tringular adalah matrik bujursangkar yang seluruh elemen diatas elemen diagonal bernilai 0 (nol).

Contoh 9: Matrik lower-triangular

$$\mathbf{A} = \begin{bmatrix} 12 & 0 & 0 & 0 \\ 32 & -2 & 0 & 0 \\ 8 & 7 & 11 & 0 \\ -5 & 10 & 6 & 9 \end{bmatrix}$$

1.4.8 Matrik tridiagonal

Matrik tridiagonal adalah matrik bujursangkar yang seluruh elemen bukan 0 (nol) berada disekitar elemen diagonal, sementara elemen lainnya bernilai 0 (nol).

Contoh 10: Matrik tridiagonal

$$\mathbf{A} = \begin{bmatrix} 3 & 6 & 0 & 0 \\ 2 & -4 & 1 & 0 \\ 0 & 5 & 8 & -7 \\ 0 & 0 & 3 & 9 \end{bmatrix}$$

1.4.9 Matrik diagonal dominan

Matrik diagonal dominan adalah matrik bujursangkar yang memenuhi

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \tag{1.4}$$

dimana i=1,2,3,..n. Coba perhatikan matrik-matrik berikut ini

$$\mathbf{A} = \begin{bmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & 5 & -6 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 6 & 4 & -3 \\ 4 & -2 & 0 \\ -3 & 0 & 1 \end{bmatrix}$$

Pada elemen diagonal a_{ii} matrik **A**, $|7| > |2|+|0|$, lalu $|5| > |3|+|-1|$, dan $|-6| > |5|+|0|$. Maka matrik **A** disebut matrik diagonal dominan. Sekarang perhatikan elemen diagonal matrik **B**, $|6| < |4|+|-3|$, $|-2| < |4|+|0|$, dan $|1| < |-3|+|0|$. Dengan demikian, matrik **B** bukan matrik diagonal dominan.

1.4.10 Matrik *positive-definite*

Suatu matrik dikatakan *positive-definite* bila matrik tersebut simetrik dan memenuhi

$$\mathbf{x}^T\mathbf{A}\mathbf{x} > 0 \tag{1.5}$$

Contoh 11: Diketahui matrik simetrik berikut

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

untuk menguji apakah matrik **A** bersifat *positive-definite*, maka

$$\begin{aligned}
\mathbf{x}^T\mathbf{A}\mathbf{x} &= \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \\
&= \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} 2x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 \end{bmatrix} \\
&= 2x_1^2 - 2x_1x_2 + 2x_2^2 - 2x_2x_3 + 2x_3^2 \\
&= x_1^2 + (x_1^2 - 2x_1x_2 + x_2^2) + (x_2^2 - 2x_2x_3 + x_3^2) + x_3^2 \\
&= x_1^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + x_3^2
\end{aligned}$$

Dari sini dapat disimpulkan bahwa matrik **A** bersifat *positive-definite*, karena memenuhi

$$x_1^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + x_3^2 > 0$$

kecuali jika x_1=x_2=x_3=0.

1.5 Operasi matematika

1.5.1 Penjumlahan matrik

Operasi penjumlahan pada dua buah matrik hanya bisa dilakukan bila kedua matrik tersebut berukuran sama. Misalnya matrik $\mathbf{C}_{2\times3}$

$$\mathbf{C} = \begin{bmatrix} 9 & 5 & 3 \\ 7 & 2 & 1 \end{bmatrix}$$

dijumlahkan dengan matrik $\mathbf{A}_{2\times3}$, lalu hasilnya (misalnya) dinamakan matrik $\mathbf{D}_{2\times3}$

$$\mathbf{D} = \mathbf{A} + \mathbf{C}$$

$$\begin{aligned}\mathbf{D} &= \begin{bmatrix}3 & 8 & 5\\ 6 & 4 & 7\end{bmatrix} + \begin{bmatrix}9 & 5 & 3\\ 7 & 2 & 1\end{bmatrix}\\ &= \begin{bmatrix}3+9 & 8+5 & 5+3\\ 6+7 & 4+2 & 7+1\end{bmatrix}\\ &= \begin{bmatrix}12 & 13 & 8\\ 13 & 6 & 8\end{bmatrix}\end{aligned}$$

Tanpa mempedulikan nilai elemen-elemen masing-masing matrik, operasi penjumlahan antara matrik $\mathbf{A}_{2\times3}$ dan $\mathbf{C}_{2\times3}$, bisa juga dinyatakan dalam indeks masing-masing dari kedua matrik tersebut, yaitu

$$\begin{bmatrix}d_{11} & d_{12} & d_{13}\\ d_{21} & d_{22} & d_{23}\end{bmatrix} = \begin{bmatrix}a_{11}+c_{11} & a_{12}+c_{12} & a_{13}+c_{13}\\ a_{21}+c_{21} & a_{22}+c_{22} & a_{23}+c_{23}\end{bmatrix}$$

Dijabarkan satu persatu sebagai berikut

$$\begin{aligned}d_{11} &= a_{11} + c_{11}\\ d_{12} &= a_{12} + c_{12}\\ d_{13} &= a_{13} + c_{13}\\ d_{21} &= a_{21} + c_{21}\\ d_{22} &= a_{22} + c_{22}\\ d_{23} &= a_{23} + c_{23}\end{aligned} \tag{1.6}$$

Dari sini dapat diturunkan sebuah rumus umum penjumlahan dua buah matrik

$$d_{ij} = a_{ij} + c_{ij} \tag{1.7}$$

dimana i=1,2 dan j=1,2,3. **Perhatikan baik-baik! Batas i hanya sampai angka 2 sementara batas j sampai angka 3. Kemampuan anda dalam menentukan batas indeks sangat penting dalam dunia *programming*.**

1.5.2 Komputasi penjumlahan matrik

Berdasarkan contoh operasi penjumlahan di atas, indeks j pada persamaan (1.7) **lebih cepat** berubah dibanding indeks i sebagaimana ditulis pada 3 baris pertama dari Persamaan (1.6),

$$\begin{aligned}d_{11} &= a_{11} + c_{11}\\ d_{12} &= a_{12} + c_{12}\\ d_{13} &= a_{13} + c_{13}\end{aligned}$$

Jelas terlihat, ketika indeks i masih bernilai 1, indeks j sudah berubah dari nilai 1 sampai 3. Hal ini membawa konsekuensi pada *script* pemrograman, dimana *looping* untuk indeks j harus diletakkan di dalam *looping* indeks i. **Aturan mainnya adalah yang *looping*-nya paling cepat harus diletakkan paling dalam; sebaliknya, *looping* terluar adalah *looping* yang indeksnya paling jarang berubah.**

Bila anda masih belum paham terhadap kalimat yang dicetak tebal, saya akan berikan contoh *source code* dasar yang nantinya akan kita optimasi selangkah demi selangkah. OK, kita mulai dari *source code* paling mentah berikut ini.

```
1  clear all
2  clc
3
4  A=[3 8 5; 6 4 7];  % inisialisasi matrik A
5
6  C=[9 5 3; 7 2 1];  % inisialisasi matrik B
7
8  % ---proses penjumlahan matrik----
9  D(1,1)=A(1,1)+C(1,1);
10 D(1,2)=A(1,2)+C(1,2);
11 D(1,3)=A(1,3)+C(1,3);
12 D(2,1)=A(2,1)+C(2,1);
13 D(2,2)=A(2,2)+C(2,2);
14 D(2,3)=A(2,3)+C(2,3);
15
16 % ---menampilkan matrik A, C dan D----
17 A
18 C
19 D
```

Tanda % berfungsi untuk memberikan komentar atau keterangan. Komentar atau keterangan tidak akan diproses oleh Matlab. Saya yakin anda paham dengan logika yang ada pada bagian % —*proses penjumlahan matrik*—- dalam source code di atas. Misalnya pada baris ke-9, elemen d_{11} adalah hasil penjumlahan antara elemen a_{11} dan c_{11}, sesuai dengan baris pertama Persamaan 1.6.

Tahap pertama penyederhanaan *source code* dilakukan dengan menerapkan perintah *for - end* untuk proses *looping*. *Source code* tersebut berubah menjadi

```
1  clear all
2  clc
3
4  A=[3 8 5; 6 4 7];  % inisialisasi matrik A
5
6  C=[9 5 3; 7 2 1];  % inisialisasi matrik B
7
8  % ---proses penjumlahan matrik----
9  for j=1:3
10    D(1,j)=A(1,j)+C(1,j);
11 end
12
13 for j=1:3
14    D(2,j)=A(2,j)+C(2,j);
15 end
```

```
16
17 % ---menampilkan matrik A, C dan D----
18 A
19 C
20 D
```

Pada baris ke-9 dan ke-13, saya mengambil huruf j sebagai nama indeks dimana j bergerak dari 1 sampai 3. Coba anda pikirkan, mengapa j hanya bergerak dari 1 sampai 3?

Modifikasi tahap kedua adalah sebagai berikut

```
1  clear all
2  clc
3
4  A=[3 8 5; 6 4 7];  % inisialisasi matrik A
5
6  C=[9 5 3; 7 2 1];  % inisialisasi matrik B
7
8  % ---proses penjumlahan matrik----
9  i=1
10 for j=1:3
11    D(i,j)=A(i,j)+C(i,j);
12 end
13
14 i=2
15 for j=1:3
16    D(i,j)=A(i,j)+C(i,j);
17 end
18
19 % ---menampilkan matrik A, C dan D----
20 A
21 C
22 D
```

Saya gunakan indeks i pada baris ke-9 dan ke-14 yang masing-masing berisi angka 1 dan 2. Dengan begitu indeks i bisa menggantikan angka 1 dan 2 yang semula ada di baris ke-11 dan ke-16. Nah sekarang coba anda perhatikan, statemen pada baris ke-10, ke-11 dan ke-12 sama persis dengan statemen pada baris ke-15, ke-16 dan ke-17, sehingga mereka bisa disatukan kedalam sebuah *looping* yang baru dimana i menjadi nama indeksnya.

```
1  clear all
2  clc
3
4  A=[3 8 5; 6 4 7];  % inisialisasi matrik A
5
6  C=[9 5 3; 7 2 1];  % inisialisasi matrik B
7
8  % ---proses penjumlahan matrik----
9  for i=1:2
10 for j=1:3
11    D(i,j)=A(i,j)+C(i,j);
12 end
13 end
14
15 % ---menampilkan matrik A, C dan D----
```

```
16 A
17 C
18 D
```

Coba anda pahami dari baris ke-9, mengapa indeks i hanya bergerak dari 1 sampai 2?

Source code di atas memang sudah tidak perlu dimodifikasi lagi, namun ada sedikit saran untuk penulisan *looping* bertingkat dimana sebaiknya *looping* terdalam ditulis agak menjorok kedalam seperti berikut ini

```
1  clear all
2  clc
3
4  A=[3 8 5; 6 4 7];  % inisialisasi matrik A
5
6  C=[9 5 3; 7 2 1];  % inisialisasi matrik B
7
8  % ---proses penjumlahan matrik----
9  for i=1:2
10   for j=1:3
11     D(i,j)=A(i,j)+C(i,j);
12   end
13 end
14
15 % ---menampilkan matrik A, C dan D----
16 A
17 C
18 D
```

Sekarang anda lihat bahwa *looping* indeks j ditulis lebih masuk kedalam dibandingkan looping indeks i. Semoga contoh ini bisa memperjelas **aturan umum pemrograman dimana yang *looping*-nya paling cepat harus diletakkan paling dalam; sebaliknya, *looping* terluar adalah *looping* yang indeksnya paling jarang berubah.** Dalam contoh ini *looping* indeks j bergerak lebih cepat dibanding *looping* indeks i.

1.5.3 Perkalian matrik

Operasi perkalian dua buah matrik hanya bisa dilakukan bila jumlah kolom matrik pertama sama dengan jumlah baris matrik kedua. Jadi kedua matrik tersebut tidak harus berukuran sama seperti pada penjumlahan dua matrik. Misalnya matrik $\mathbf{A}_{2\times 3}$ dikalikan dengan matrik $\mathbf{B}_{3\times 2}$, lalu hasilnya (misalnya) dinamakan matrik $\mathbf{E}_{2\times 2}$

$$\mathbf{E}_{2\times 2} = \mathbf{A}_{2\times 3}.\mathbf{B}_{3\times 2}$$

$$\begin{aligned}\mathbf{E} &= \begin{bmatrix}3 & 8 & 5\\ 6 & 4 & 7\end{bmatrix}\begin{bmatrix}1 & 3\\ 5 & 9\\ 2 & 4\end{bmatrix}\\ &= \begin{bmatrix}3.1+8.5+5.2 & 3.3+8.9+5.4\\ 6.1+4.5+7.2 & 6.3+4.9+7.4\end{bmatrix}\\ &= \begin{bmatrix}53 & 101\\ 40 & 82\end{bmatrix}\end{aligned}$$

Tanpa mempedulikan nilai elemen-elemen masing-masing matrik, operasi perkalian antara matrik $\mathbf{A}_{2\times 3}$ dan $\mathbf{B}_{3\times 2}$, bisa juga dinyatakan dalam indeks masing-masing dari kedua matrik tersebut, yaitu

$$\begin{bmatrix}e_{11} & e_{12}\\ e_{21} & e_{22}\end{bmatrix} = \begin{bmatrix}a_{11}.b_{11}+a_{12}.b_{21}+a_{13}.b_{31} & a_{11}.b_{12}+a_{12}.b_{22}+a_{13}.b_{32}\\ a_{21}.b_{11}+a_{22}.b_{21}+a_{23}.b_{31} & a_{21}.b_{12}+a_{22}.b_{22}+a_{23}.b_{32}\end{bmatrix}$$

Bila dijabarkan, maka elemen-elemen matrik $\mathbf{E}_{2\times 2}$ adalah

$$e_{11} = a_{11}.b_{11}+a_{12}.b_{21}+a_{13}.b_{31} \tag{1.8}$$
$$e_{12} = a_{11}.b_{12}+a_{12}.b_{22}+a_{13}.b_{32} \tag{1.9}$$
$$e_{21} = a_{21}.b_{11}+a_{22}.b_{21}+a_{23}.b_{31} \tag{1.10}$$
$$e_{22} = a_{21}.b_{12}+a_{22}.b_{22}+a_{23}.b_{32} \tag{1.11}$$

Sejenak, mari kita amati perubahan pasangan angka-angka indeks yang mengiringi elemen e, elemen a dan elemen b mulai dari persamaan (1.8) sampai persamaan (1.11). Perhatikan perubahan angka-indeks-pertama pada elemen e seperti berikut ini

$$e_{1..} = ..$$
$$e_{1..} = ..$$
$$e_{2..} = ..$$
$$e_{2..} = ..$$

Pola perubahan yang sama akan kita dapati pada angka-indeks-pertama dari elemen a

$$e_{1..} = a_{1..}.b_{...}+a_{1..}.b_{...}+a_{1..}.b_{...}$$
$$e_{1..} = a_{1..}.b_{...}+a_{1..}.b_{...}+a_{1..}.b_{...}$$
$$e_{2..} = a_{2..}.b_{...}+a_{2..}.b_{...}+a_{2..}.b_{...}$$
$$e_{2..} = a_{2..}.b_{...}+a_{2..}.b_{...}+a_{2..}.b_{...}$$

Dengan demikian kita bisa mencantumkan huruf i sebagai pengganti angka-angka indeks

yang polanya sama

$$e_{i..} = a_{i..}.b_{...} + a_{i..}.b_{...} + a_{i..}.b_{...}$$
$$e_{i..} = a_{i..}.b_{...} + a_{i..}.b_{...} + a_{i..}.b_{...}$$
$$e_{i..} = a_{i..}.b_{...} + a_{i..}.b_{...} + a_{i..}.b_{...}$$
$$e_{i..} = a_{i..}.b_{...} + a_{i..}.b_{...} + a_{i..}.b_{...}$$

dimana i bergerak mulai dari angka 1 hingga angka 2, atau kita nyatakan i=1,2. Selanjutnya, masih dari persamaan (1.8) sampai persamaan (1.11), marilah kita perhatikan perubahan angka-indeks-kedua pada elemen e dan elemen b,

$$e_{i1} = a_{i..}.b_{..1} + a_{i..}.b_{..1} + a_{i..}.b_{..1}$$
$$e_{i2} = a_{i..}.b_{..2} + a_{i..}.b_{..2} + a_{i..}.b_{..2}$$
$$e_{i1} = a_{i..}.b_{..1} + a_{i..}.b_{..1} + a_{i..}.b_{..1}$$
$$e_{i2} = a_{i..}.b_{..2} + a_{i..}.b_{..2} + a_{i..}.b_{..2}$$

Dengan demikian kita bisa mencantumkan huruf j sebagai pengganti angka-angka indeks yang polanya sama

$$e_{ij} = a_{i..}.b_{..j} + a_{i..}.b_{..j} + a_{i..}.b_{..j}$$
$$e_{ij} = a_{i..}.b_{..j} + a_{i..}.b_{..j} + a_{i..}.b_{..j}$$
$$e_{ij} = a_{i..}.b_{..j} + a_{i..}.b_{..j} + a_{i..}.b_{..j}$$
$$e_{ij} = a_{i..}.b_{..j} + a_{i..}.b_{..j} + a_{i..}.b_{..j}$$

dimana j bergerak mulai dari angka 1 hingga angka 2, atau kita nyatakan j=1,2. Selanjutnya, masih dari persamaan (1.8) sampai persamaan (1.11), mari kita perhatikan perubahan angka-indeks-kedua elemen a dan angka-indeks-pertama elemen b, dimana kita akan dapati pola sebagai berikut

$$e_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + a_{i3}.b_{3j}$$
$$e_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + a_{i3}.b_{3j}$$
$$e_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + a_{i3}.b_{3j}$$
$$e_{ij} = a_{i1}.b_{1j} + a_{i2}.b_{2j} + a_{i3}.b_{3j}$$

Dan kita bisa mencantumkan huruf k sebagai pengganti angka-angka indeks yang polanya

sama, dimana k bergerak mulai dari angka 1 hingga angka 3, atau kita nyatakan k=1,2,3.

$$e_{ij} = a_{ik}.b_{kj} + a_{ik}.b_{kj} + a_{ik}.b_{kj}$$
$$e_{ij} = a_{ik}.b_{kj} + a_{ik}.b_{kj} + a_{ik}.b_{kj}$$
$$e_{ij} = a_{ik}.b_{kj} + a_{ik}.b_{kj} + a_{ik}.b_{kj}$$
$$e_{ij} = a_{ik}.b_{kj} + a_{ik}.b_{kj} + a_{ik}.b_{kj}$$

Kemudian secara sederhana dapat ditulis sebagai berikut

$$e_{ij} = a_{ik}.b_{kj} + a_{ik}.b_{kj} + a_{ik}.b_{kj} \tag{1.12}$$

Selanjutnya dapat ditulis pula formula berikut

$$e_{ij} = \sum_{k=1}^{3} a_{ik} b_{kj} \tag{1.13}$$

dimana i=1,2; j=1,2; dan k=1,2,3.
Berdasarkan contoh ini, maka secara umum bila ada matrik $\mathbf{A}_{n\times m}$ yang dikalikan dengan matrik $\mathbf{B}_{m\times p}$, akan didapatkan matrik $\mathbf{E}_{n\times p}$ dimana elemen-elemen matrik **E** memenuhi

$$e_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \tag{1.14}$$

dengan i=1,2,...,n; j=1,2...,p; dan k=1,2...,m.

1.5.4 Komputasi perkalian matrik

Mari kita mulai lagi dari *source code* paling dasar dari operasi perkalian matrik sesuai dengan contoh di atas.

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

% ---proses perkalian matrik----
E(1,1)=A(1,1)*B(1,1)+A(1,2)*B(2,1)+A(1,3)*B(3,1);
E(1,2)=A(1,1)*B(1,2)+A(1,2)*B(2,2)+A(1,3)*B(3,2);
E(2,1)=A(2,1)*B(1,1)+A(2,2)*B(2,1)+A(2,3)*B(3,1);
E(2,2)=A(2,1)*B(1,2)+A(2,2)*B(2,2)+A(2,3)*B(3,2);

% ---menampilkan matrik A, B dan E----
A
B
E
```

Sejenak, mari kita amati dengan cermat statemen dari baris ke-9 sampai ke-12 sambil dikaitkan dengan bentuk umum penulisan indeks pada perkalian matrik yaitu

$$e_{ij} = a_{ik}.b_{kj} + a_{ik}.b_{kj} + a_{ik}.b_{kj} \tag{1.15}$$

Dari sana ada 4 *point* yang perlu dicatat:

- elemen e memiliki indeks i dan indeks j dimana indeks j lebih cepat berubah dibanding indeks i.
- pada baris statemen ke-8 sampai ke-11 ada tiga kali operasi perkalian dan dua kali operasi penjumlahan yang semuanya melibatkan indeks i, indeks j dan indeks k. Namun indeks k selalu berubah pada masing-masing perkalian. Jadi indeks k paling cepat berubah dibanding indeks i dan indeks j.
- elemen a memiliki indeks i dan indeks k dimana indeks k lebih cepat berubah dibanding indeks i.
- elemen b memiliki indeks k dan indeks j dimana indeks k lebih cepat berubah dibanding indeks j.

Tahapan modifikasi source code perkalian matrik tidak semudah penjumlahan matrik. Dan mengajarkan logika dibalik source code perkalian matrik jauh lebih sulit daripada sekedar memodifikasi source code tersebut. Tapi akan saya coba semampu saya lewat tulisan ini walau harus perlahan-lahan. Mudah-mudahan mudah untuk dipahami.

Saya mulai dengan memecah operasi pada statemen baris ke-8 yang bertujuan menghitung nilai $E(1,1)$

```
1  clear all
2  clc
3
4  A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5  B = [1 3; 5 9; 2 4]; % inisialisasi matrik B
6
7  % ---proses perkalian matrik----
8  % ---E(1,1) dihitung 3 kali
9  E(1,1)=A(1,1)*B(1,1);
10 E(1,1)=E(1,1)+A(1,2)*B(2,1);
11 E(1,1)=E(1,1)+A(1,3)*B(3,1);
12
13 % ---E(1,2); E(2,1); dan E(2,2) masih seperti semula
14 E(1,2)=A(1,1)*B(1,2)+A(1,2)*B(2,2)+A(1,3)*B(3,2);
15 E(2,1)=A(2,1)*B(1,1)+A(2,2)*B(2,1)+A(2,3)*B(3,1);
16 E(2,2)=A(2,1)*B(1,2)+A(2,2)*B(2,2)+A(2,3)*B(3,2);
17
18 % ---menampilkan matrik A, B dan E----
19 A
20 B
21 E
```

Agar baris ke-9 memiliki pola yang sama dengan baris ke-11 dan ke-12, upaya yang dilakukan adalah

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

% ---proses perkalian matrik----
% ---E(1,1) dihitung 3 kali
E(1,1)=0;
E(1,1)=E(1,1)+A(1,1)*B(1,1);
E(1,1)=E(1,1)+A(1,2)*B(2,1);
E(1,1)=E(1,1)+A(1,3)*B(3,1);

% ---E(1,2); E(2,1); dan E(2,2) masih seperti semula
E(1,2)=A(1,1)*B(1,2)+A(1,2)*B(2,2)+A(1,3)*B(3,2);
E(2,1)=A(2,1)*B(1,1)+A(2,2)*B(2,1)+A(2,3)*B(3,1);
E(2,2)=A(2,1)*B(1,2)+A(2,2)*B(2,2)+A(2,3)*B(3,2);

% ---menampilkan matrik A, B dan E----
A
B
E
```

Dari sini kita bisa munculkan indeks k

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

% ---proses perkalian matrik----
E(1,1)=0;
for k=1:3            % k bergerak dari 1 sampai 3
   E(1,1)=E(1,1)+A(1,k)*B(k,1);
end

% ---E(1,2); E(2,1); dan E(2,2) masih seperti semula
E(1,2)=A(1,1)*B(1,2)+A(1,2)*B(2,2)+A(1,3)*B(3,2);
E(2,1)=A(2,1)*B(1,1)+A(2,2)*B(2,1)+A(2,3)*B(3,1);
E(2,2)=A(2,1)*B(1,2)+A(2,2)*B(2,2)+A(2,3)*B(3,2);

% ---menampilkan matrik A, B dan E----
A
B
E
```

Kemudian cara yang sama dilakukan pada $E(1,2)$, $E(2,1)$, dan $E(2,2)$. Anda mesti cermat dan hati-hati dalam menulis angka-angka indeks!!!

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

```

```
% ---proses perkalian matrik----
E(1,1)=0;
for k=1:3
   E(1,1)=E(1,1)+A(1,k)*B(k,1);
end

E(1,2)=0;
for k=1:3
   E(1,2)=E(1,2)+A(1,k)*B(k,2);
end

E(2,1)=0;
for k=1:3
   E(2,1)=E(2,1)+A(2,k)*B(k,1);
end

E(2,2)=0;
for k=1:3
   E(2,2)=E(2,2)+A(2,k)*B(k,2);
end

% ---menampilkan matrik A, B dan E----
A
B
E
```

Inisialisasi elemen-elemen matrik E dengan angka nol, bisa dilakukan diawal proses perkalian yang sekaligus memunculkan indeks i dan j untuk elemen E

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

% ---proses perkalian matrik----
for i=1:2            % i bergerak dari 1 sampai 2
   for j=1:2         % j bergerak dari 1 sampai 2
      E(i,j)=0;
   end
end

for k=1:3
   E(1,1)=E(1,1)+A(1,k)*B(k,1);
end

for k=1:3
   E(1,2)=E(1,2)+A(1,k)*B(k,2);
end

for k=1:3
   E(2,1)=E(2,1)+A(2,k)*B(k,1);
end

for k=1:3
   E(2,2)=E(2,2)+A(2,k)*B(k,2);
end

```

```
30 % ---menampilkan matrik A, B dan E----
31 A
32 B
33 E
```

Sekarang coba anda perhatikan statemen pada baris ke-15 dan ke-19, lalu bandingkan indeks i dan indeks j pada elemen E. Indeks mana yang berubah? Ya. Jawabannya adalah indeks j. Dengan demikian kita bisa munculkan indeks j

```
1  clear all
2  clc
3
4  A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5  B = [1 3; 5 9; 2 4]; % inisialisasi matrik B
6
7  % ---proses perkalian matrik----
8  for i=1:2            % i bergerak dari 1 sampai 2
9     for j=1:2         % j bergerak dari 1 sampai 2
10       E(i,j)=0;
11    end
12 end
13
14 j=1;
15 for k=1:3
16    E(1,j)=E(1,j)+A(1,k)*B(k,j);
17 end
18
19 j=2;
20 for k=1:3
21    E(1,j)=E(1,j)+A(1,k)*B(k,j);
22 end
23
24 for k=1:3
25    E(2,1)=E(2,1)+A(2,k)*B(k,1);
26 end
27
28 for k=1:3
29    E(2,2)=E(2,2)+A(2,k)*B(k,2);
30 end
31
32 % ---menampilkan matrik A, B dan E----
33 A
34 B
35 E
```

Lihatlah, statemen dari baris ke-15 sampai ke-17 memiliki pola yang sama dengan statemen dari baris ke-20 sampai ke-22, sehingga mereka bisa disatukan kedalam *looping* indeks j

```
1  clear all
2  clc
3
4  A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5  B = [1 3; 5 9; 2 4]; % inisialisasi matrik B
6
7  % ---proses perkalian matrik----
8  for i=1:2            % i bergerak dari 1 sampai 2
```

```
   for j=1:2         % j bergerak dari 1 sampai 2
      E(i,j)=0;
   end
end

for j=1:2
   for k=1:3
      E(1,j)=E(1,j)+A(1,k)*B(k,j);
   end
end

for k=1:3
   E(2,1)=E(2,1)+A(2,k)*B(k,1);
end

for k=1:3
   E(2,2)=E(2,2)+A(2,k)*B(k,2);
end

% ---menampilkan matrik A, B dan E----
A
B
E
```

Sekarang coba sekali lagi anda perhatikan statemen pada baris ke-21 dan ke-25, lalu bandingkan indeks i dan indeks j pada elemen E. Indeks mana yang berubah? Ya. Jawabannya tetap indeks j. Dengan demikian kita bisa munculkan juga indeks j disana

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

% ---proses perkalian matrik----
for i=1:2            % i bergerak dari 1 sampai 2
   for j=1:2         % j bergerak dari 1 sampai 2
      E(i,j)=0;
   end
end

for j=1:2
   for k=1:3
      E(1,j)=E(1,j)+A(1,k)*B(k,j);
   end
end

j=1;
for k=1:3
   E(2,j)=E(2,j)+A(2,k)*B(k,j);
end

j=2;
for k=1:3
   E(2,j)=E(2,j)+A(2,k)*B(k,j);
end

```

```
30 % ---menampilkan matrik A, B dan E----
31 A
32 B
33 E
```

Cermatilah, statemen dari baris ke-21 sampai ke-23 memiliki pola yang sama dengan statemen dari baris ke-25 sampai ke-27, sehingga mereka pun bisa disatukan kedalam *looping* indeks j

```
1  clear all
2  clc
3
4  A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5  B = [1 3; 5 9; 2 4]; % inisialisasi matrik B
6
7  % ---proses perkalian matrik----
8  for i=1:2            % i bergerak dari 1 sampai 2
9     for j=1:2         % j bergerak dari 1 sampai 2
10       E(i,j)=0;
11    end
12 end
13
14 for j=1:2
15    for k=1:3
16       E(1,j)=E(1,j)+A(1,k)*B(k,j);
17    end
18 end
19
20 for j=1:2
21    for k=1:3
22       E(2,j)=E(2,j)+A(2,k)*B(k,j);
23    end
24 end
25
26 % ---menampilkan matrik A, B dan E----
27 A
28 B
29 E
```

Akhirnya kita sampai pada bagian akhir tahapan modifikasi. Perhatikan baris ke-16 dan ke-22. Indeks i pada elemen E dan A bergerak dari 1 ke 2, sehingga indeks i bisa dimunculkan

```
1  clear all
2  clc
3
4  A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5  B = [1 3; 5 9; 2 4]; % inisialisasi matrik B
6
7  % ---proses perkalian matrik----
8  for i=1:2            % i bergerak dari 1 sampai 2
9     for j=1:2         % j bergerak dari 1 sampai 2
10       E(i,j)=0;
11    end
12 end
13
14 i=1;
15 for j=1:2
```

```
16     for k=1:3
17        E(i,j)=E(i,j)+A(i,k)*B(k,j);
18     end
19  end
20
21  i=2;
22  for j=1:2
23     for k=1:3
24        E(i,j)=E(i,j)+A(i,k)*B(k,j);
25     end
26  end
27
28  % ---menampilkan matrik A, B dan E----
29  A
30  B
31  E
```

Sekarang, statemen dari baris ke-15 sampai ke-19 memiliki pola yang sama dengan statemen dari baris ke-22 sampai ke-26. Mereka bisa disatukan oleh *looping* indeks i

```
1   clear all
2   clc
3
4   A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5   B = [1 3; 5 9; 2 4]; % inisialisasi matrik B
6
7   % ---proses perkalian matrik----
8   for i=1:2
9      for j=1:2
10        E(i,j)=0;
11     end
12  end
13
14  for i=1:2
15     for j=1:2
16        for k=1:3
17           E(i,j)=E(i,j)+A(i,k)*B(k,j);
18        end
19     end
20  end
21
22  % ---menampilkan matrik A, B dan E----
23  A
24  B
25  E
```

Inilah hasil akhir dari tahapan-tahapan modifikasi yang selanjutnya saya sebut sebagai proses **optimasi**. Upaya yang baru saja saya perlihatkan, sebenarnya penuh dengan jebakan-jebakan kesalahan, terutama jika anda kurang cermat membaca indeks dan pola. Upaya seperti itu memerlukan konsentrasi dan perhatian yang tidak sebentar. Upaya semacam itu tidak semudah meng-*copy* hasil akhir optimasi. Walaupun bisa di-*copy*, namun saya menyarankan agar anda mencoba melakukan proses optimasi itu sekali lagi di komputer tanpa melihat catatan ini dan tanpa bantuan orang lain. Kalau anda gagal, cobalah berfikir lebih keras untuk mencari jalan keluarnya. Jika masih tetap gagal, silakan lihat catatan ini sebentar saja sekedar untuk

mencari tahu dimana letak kesalahannya. Hanya dengan cara begitu ilmu *programming* ini akan bisa menyatu pada diri anda.

1.5.5 Perkalian matrik dan vektor-kolom

Operasi perkalian antara matrik dan vektor-kolom sebenarnya sama saja dengan perkalian antara dua matrik. Hanya saja ukuran vektor-kolom boleh dibilang spesial yaitu m x 1, dimana m merupakan jumlah baris sementara jumlah kolomnya hanya satu. Misalnya matrik **A**, pada contoh 1, dikalikan dengan vektor-kolom **x** yang berukuran 3 x 1 atau disingkat dengan mengatakan vektor-kolom **x** berukuran 3, lalu hasilnya (misalnya) dinamakan vektor-kolom **y**

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\begin{aligned}
\mathbf{y} &= \begin{bmatrix} 3 & 8 & 5 \\ 6 & 4 & 7 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \\
&= \begin{bmatrix} 3.2 + 8.3 + 5.4 \\ 6.2 + 4.3 + 7.4 \end{bmatrix} \\
&= \begin{bmatrix} 50 \\ 52 \end{bmatrix}
\end{aligned}$$

Sekali lagi, tanpa mempedulikan nilai elemen-elemen masing-masing, operasi perkalian antara matrik **A** dan vektor-kolom **x**, bisa juga dinyatakan dalam indeksnya masing-masing, yaitu

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11}.x_1 + a_{12}.x_2 + a_{13}.x_3 \\ a_{21}.x_1 + a_{22}.x_2 + a_{23}.x_3 \end{bmatrix}$$

Bila dijabarkan, maka elemen-elemen vektor-kolom **y** adalah

$$\begin{aligned}
y_1 &= a_{11}.x_1 + a_{12}.x_2 + a_{13}.x_3 \\
y_2 &= a_{21}.x_1 + a_{22}.x_2 + a_{23}.x_3
\end{aligned}$$

kemudian secara sederhana dapat diwakili oleh rumus berikut

$$y_i = \sum_{j=1}^{3} a_{ij} x_j$$

dimana i=1,2.
Berdasarkan contoh tersebut, secara umum bila ada matrik **A** berukuran n x m yang dikalikan dengan vektor-kolom **x** berukuran m, maka akan didapatkan vektor-kolom **y** berukuran n x 1

dimana elemen-elemen vektor-kolom **y** memenuhi

$$y_i = \sum_{j=1}^{m} a_{ij} x_j \tag{1.16}$$

dengan i=1,2,... ,n.

1.5.6 Komputasi perkalian matrik dan vektor-kolom

Mari kita mulai lagi dari *source code* paling dasar dari operasi perkalian antara matrik dan vektor-kolom sesuai dengan contoh di atas

```
1  clear all
2  clc
3
4  A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5  x = [2; 3; 4];       % inisialisasi vektor x
6
7  % ---proses perkalian matrik dan vektor----
8  y(1,1)=A(1,1)*x(1,1)+A(1,2)*x(2,1)+A(1,3)*x(3,1);
9  y(2,1)=A(2,1)*x(1,1)+A(2,2)*x(2,1)+A(2,3)*x(3,1);
10
11 % ---menampilkan matrik A, B dan E----
12 A
13 x
14 y
```

Sejenak, mari kita amati dengan cermat statemen dari baris ke-8 dan ke-9 sambil dikaitkan dengan bentuk umum penulisan indeks pada perkalian antara matrik dan vektor-kolom yaitu

$$y_{i1} = a_{ij}.x_{j1} + a_{ij}.x_{j1} + a_{ij}.x_{j1} \tag{1.17}$$

Dari sana ada 3 *point* yang perlu dicatat:

- elemen y dan elemen x sama-sama memiliki indeks i yang berpasangan dengan angka 1.
- pada baris statemen ke-8 dan ke-9 ada tiga kali operasi perkalian dan dua kali operasi penjumlahan yang semuanya melibatkan indeks i dan indeks j. Namun indeks j selalu berubah pada masing-masing perkalian. Jadi indeks j lebih cepat berubah dibanding indeks i.
- elemen a memiliki indeks i dan indeks j dimana indeks j lebih cepat berubah dibanding indeks i.

Kita mulai dengan memecah operasi pada statemen baris ke-8 yang bertujuan menghitung nilai $y(1,1)$

```
1  clear all
2  clc
3
```

```
A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
y(1,1)=A(1,1)*x(1,1);
y(1,1)=y(1,1)+A(1,2)*x(2,1);
y(1,1)=y(1,1)+A(1,3)*x(3,1);

y(2,1)=A(2,1)*x(1,1)+A(2,2)*x(2,1)+A(2,3)*x(3,1);

% ---menampilkan matrik A, B dan E----
A
x
y
```

Agar baris ke-8 memiliki pola yang sama dengan baris ke-9 dan ke-10, upaya yang dilakukan adalah

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
y(1,1)=0;
y(1,1)=y(1,1)+A(1,1)*x(1,1);
y(1,1)=y(1,1)+A(1,2)*x(2,1);
y(1,1)=y(1,1)+A(1,3)*x(3,1);

y(2,1)=A(2,1)*x(1,1)+A(2,2)*x(2,1)+A(2,3)*x(3,1);

% ---menampilkan matrik A, B dan E----
A
x
y
```

Dari sini kita bisa munculkan indeks j

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
y(1,1)=0;
for j=1:3
   y(1,1)=y(1,1)+A(1,j)*x(j,1);
end

y(2,1)=A(2,1)*x(1,1)+A(2,2)*x(2,1)+A(2,3)*x(3,1);

% ---menampilkan matrik A, B dan E----
A
x
y
```

Dengan cara yang sama, baris ke-13 dimodifikasi menjadi

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
y(1,1)=0;
for j=1:3
   y(1,1)=y(1,1)+A(1,j)*x(j,1);
end

y(2,1)=0;
for j=1:3
   y(2,1)=y(2,1)+A(2,j)*x(j,1);
end

% ---menampilkan matrik A, B dan E----
A
x
y
```

Inisialisasi vektor $\mathbf{y}$ dengan angka nol dapat dilakukan diawal proses perkalian, sekaligus memunculkan indeks i

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
for i=1:2
   y(i,1)=0;
end

for j=1:3
   y(1,1)=y(1,1)+A(1,j)*x(j,1);
end

for j=1:3
   y(2,1)=y(2,1)+A(2,j)*x(j,1);
end

% ---menampilkan matrik A, B dan E----
A
x
y
```

Kemudian, untuk menyamakan pola statemen baris ke-13 dan ke-17, indeks i kembali dimunculkan

```
clear all
clc

```

```
A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
for i=1:2
   y(i,1)=0;
end

i=1;
for j=1:3
   y(i,1)=y(i,1)+A(i,j)*x(j,1);
end

i=2;
for j=1:3
   y(i,1)=y(i,1)+A(i,j)*x(j,1);
end

% ---menampilkan matrik A, B dan E----
A
x
y
```

Akhir dari proses optimasi adalah sebagai berikut

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
for i=1:2
   y(i,1)=0;
end

for i=1:2
   for j=1:3
      y(i,1)=y(i,1)+A(i,j)*x(j,1);
   end
end

% ---menampilkan matrik A, B dan E----
A
x
y
```

1.6 Penutup

Demikianlah catatan singkat dan sederhana mengenai jenis-jenis matrik dasar dan operasi penjumlahan dan perkalian yang seringkali dijumpai dalam pengolahan data secara numerik. Semuanya akan dijadikan acuan atau referensi pada pembahasan topik-topik numerik yang akan datang.

1.7 Latihan

Diketahui matrik **A**, matrik **B**, dan vektor **x** sebagai berikut

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -6 & -2 \\ 5 & 9 & 7 & 5.6 \\ 2 & 4 & 8 & -1 \\ 2.3 & 1.4 & 0.8 & -2.3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 8 & 1 & 4 & 21 \\ 3 & 10 & 5 & 0.1 \\ 7 & -2 & 9 & -5 \\ 2.7 & -12 & -8.9 & 5.7 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 0.4178 \\ -2.9587 \\ 56.3069 \\ 8.1 \end{bmatrix}$$

1. Buatlah script untuk menyelesaikan penjumlahan matrik **A** dan matrik **B**.
2. Buatlah script untuk menyelesaikan perkalian matrik **A** dan matrik **B**.
3. Buatlah script untuk menyelesaikan perkalian matrik **A** dan vektor **x**.
4. Buatlah script untuk menyelesaikan perkalian matrik **B** dan vektor **x**.

Bab 2

Fungsi

Objektif :

- ▷ Mengenalkan fungsi internal.
- ▷ Membuat fungsi ekstenal.
- ▷ Membuat fungsi ekternal untuk penjumlahan matrik.
- ▷ Membuat fungsi ekternal untuk perkalian matrik.

2.1 Fungsi internal

Pada bab terdahulu kita sudah melakukan proses optimasi penjumlahan matrik dengan *source code* akhir seperti ini

```
clear all
clc

A=[3 8 5; 6 4 7];  % inisialisasi matrik A
C=[9 5 3; 7 2 1];  % inisialisasi matrik B

% ---proses penjumlahan matrik----
for i=1:2
  for j=1:3
    D(i,j)=A(i,j)+C(i,j);
  end
end

% ---menampilkan matrik A, C dan D----
A
C
D
```

Pertanyaan yang segera muncul adalah apakah *source code* tersebut bisa digunakan untuk menyelesaikan penjumlahan matrik yang dimensinya bukan 2x3 ? Misalnya

$$\mathbf{D} = \mathbf{A} + \mathbf{C}$$

$$\mathbf{D} = \begin{bmatrix} 4 & 3 & 8 & 6 \\ 5 & 1 & 2 & 3 \\ 6 & 7 & 9 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 6 & 7 & 2 \\ 9 & 1 & 3 & 8 \\ 5 & 8 & 4 & 7 \end{bmatrix}$$

Tentu saja bisa, asal indeks i bergerak dari 1 sampai 3 dan indeks j bergerak dari 1 sampai 4. Lihat *source code* berikut

```
1  clear all
2  clc
3
4  A=[4 3 8 6; 5 1 2 3; 6 7 9 1];  % inisialisasi matrik A
5  C=[2 6 7 2; 9 1 3 8; 5 8 4 7];  % inisialisasi matrik B
6
7  % ---proses penjumlahan matrik----
8  for i=1:3
9    for j=1:4
10     D(i,j)=A(i,j)+C(i,j);
11   end
12 end
13
14 % ---menampilkan matrik A, C dan D----
15 A
16 C
17 D
```

Walaupun bisa digunakan, namun cara modifikasi seperti itu sangat tidak fleksibel dan beresiko salah jika kurang teliti. Untuk menghindari resiko kesalahan dan agar lebih fleksibel, *source code* tersebut perlu dioptimasi sedikit lagi menjadi

```
1  clear all
2  clc
3
4  A=[4 3 8 6; 5 1 2 3; 6 7 9 1];  % inisialisasi matrik A
5  C=[2 6 7 2; 9 1 3 8; 5 8 4 7];  % inisialisasi matrik B
6
7  % ---proses penjumlahan matrik----
8  dim=size(A);
9  n=dim(1);
10 m=dim(2);
11 for i=1:n
12   for j=1:m
13     D(i,j)=A(i,j)+C(i,j);
14   end
15 end
16
17 % ---menampilkan matrik A, C dan D----
18 A
19 C
20 D
```

Perhatikan, ada tambahan 3 statemen yaitu mulai dari baris ke-8 sampai ke-10. Sementara baris ke-11 dan ke-12 hanya mengalami sedikit perubahan. Statemen di baris ke-8 bermaksud mendeklarasikan variabel dim untuk diisi oleh hasil perhitungan fungsi internal yang bernama

$size$. Matrik **A** dijadikan parameter input fungsi $size$. Fungsi $size$ berguna untuk menghitung jumlah baris dan jumlah kolom dari matrik **A**. Hasilnya adalah $dim(1)$ untuk jumlah baris dan $dim(2)$ untuk jumlah kolom. Pada baris ke-9, variabel n dideklarasikan untuk menerima informasi jumlah baris dari $dim(1)$, sementara variabel m diisi dengan informasi jumlah kolom dari $dim(2)$ pada baris ke-10. Adapun baris ke-11 dan ke-12 hanya mengubah angka indeks batas atas, masing-masing menjadi n dan m.

Sekarang kalau kita balik lagi menghitung penjumlahan matrik dari contoh sebelumnya yang berukuran 2x3, maka source code akan seperti ini

```
1  clear all
2  clc
3
4  A=[3 8 5; 6 4 7];  % inisialisasi matrik A
5  C=[9 5 3; 7 2 1];  % inisialisasi matrik B
6
7  % ---proses penjumlahan matrik----
8  dim=size(A);
9  n=dim(1);
10 m=dim(2);
11 for i=1:n
12   for j=1:m
13     D(i,j)=A(i,j)+C(i,j);
14   end
15 end
16
17 % ---menampilkan matrik A, C dan D----
18 A
19 C
20 D
```

Ajaib bukan!? Tidak ada statemen yang berubah kecuali hanya pada baris ke-4 dan ke-5. Perubahan itu tidak bisa dihindari karena memang di kedua baris itulah deklarasi elemen-elemen matrik **A** dan matrik **C** dilakukan.

2.2 Fungsi eksternal penjumlahan matrik

Saatnya kita memasuki topik tentang pembuatan fungsi eksternal. Dari *source code* yang terakhir tadi, mari kita ambil bagian proses penjumlahan matrik-nya saja

```
1  dim=size(A);
2  n=dim(1);
3  m=dim(2);
4  for i=1:n
5    for j=1:m
6      D(i,j)=A(i,j)+C(i,j);
7    end
8  end
```

Kita akan jadikan potongan *source code* ini menjadi fungsi eksternal, dengan menambahkan statemen *function* seperti ini

```
function D=jumlah(A,C)
dim=size(A);
n=dim(1);
m=dim(2);
for i=1:n
  for j=1:m
    D(i,j)=A(i,j)+C(i,j);
  end
end
```

kemudian ia harus di-*save* dengan nama *jumlah.m*. Sampai dengan langkah ini kita telah membuat fungsi eksternal dan diberi nama fungsi *jumlah*. Sederhana sekali bukan? Untuk menguji kerja fungsi eksternal tersebut, coba jalankan *source code* berikut ini

```
clear all
clc

A=[3 8 5; 6 4 7];  % inisialisasi matrik A
C=[9 5 3; 7 2 1];  % inisialisasi matrik B

% ---proses penjumlahan matrik----
D=jumlah(A,C)

% ---menampilkan matrik A, C dan D----
A
C
D
```

atau anda jalankan *source code* yang berikut ini

```
clear all
clc

A=[4 3 8 6; 5 1 2 3; 6 7 9 1];  % inisialisasi matrik A
C=[2 6 7 2; 9 1 3 8; 5 8 4 7];  % inisialisasi matrik B

% ---proses penjumlahan matrik----
D=jumlah(A,C)

% ---menampilkan matrik A, C dan D----
A
C
D
```

atau coba iseng-iseng anda ganti matrik-nya menjadi

```
clear all
clc

V=[4 3; 5 1];  % inisialisasi matrik V
W=[2 6; 9 3];  % inisialisasi matrik W

% ---proses penjumlahan matrik----
U=jumlah(V,W)
```

```
9
10 % ---menampilkan matrik V, W dan U----
11 W
12 V
13 U
```

Periksa hasilnya, betul atau salah? Pasti betul! Kesimpulannya adalah setelah fungsi eksternal berhasil anda dapatkan, maka seketika itu pula anda tidak perlu menggubrisnya lagi. Bahkan anda tidak perlu mengingat nama matrik aslinya yang tertulis di fungsi *jumlah* yaitu matrik **A**, matrik **C** dan matrik **D**. Ditambah lagi, *source code* anda menjadi terlihat lebih singkat dan elegan. Dan kini, perhatian anda bisa lebih difokuskan pada deklarasi matrik-nya saja.

2.3 Fungsi eksternal perkalian matrik

Mari kita beralih ke perkalian matrik. Kita akan membuat fungsi eksternal untuk perkalian matrik. Berikut ini adalah *source code* perkalian matrik hasil akhir optimasi yang telah ditulis panjang lebar pada bab sebelumnya

```
1  clear all
2  clc
3
4  A = [3 8 5; 6 4 7];  % inisialisasi matrik A
5  B = [1 3; 5 9; 2 4]; % inisialisasi matrik B
6
7  % ---proses perkalian matrik----
8  for i=1:2
9     for j=1:2
10       E(i,j)=0;
11    end
12 end
13
14 for i=1:2
15    for j=1:2
16       for k=1:3
17          E(i,j)=E(i,j)+A(i,k)*B(k,j);
18       end
19    end
20 end
21
22 % ---menampilkan matrik A, B dan E----
23 A
24 B
25 E
```

Source code tersebut digunakan untuk menghitung perkalian matrik berikut

$$\mathbf{E}_{2\times 2} = \mathbf{A}_{2\times 3} \cdot \mathbf{B}_{3\times 2}$$

Dan kita bisa sepakati simbol indeks m, n, dan p untuk men-generalisir dimensi matrik

$$\mathbf{E}_{m\times n} = \mathbf{A}_{m\times p} \cdot \mathbf{B}_{p\times n}$$

Dengan demikian, *source code* tersebut dapat dioptimasi menjadi

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

% ---proses perkalian matrik----
dim=size(A);
m=dim(1);
p=dim(2);
dim=size(B);
n=dim(2);
for i=1:m
   for j=1:n
      E(i,j)=0;
   end
end

for i=1:m
   for j=1:n
      for k=1:p
         E(i,j)=E(i,j)+A(i,k)*B(k,j);
      end
   end
end

% ---menampilkan matrik A, B dan E----
A
B
E
```

Selanjutnya kita ambil bagian *proses perkalian matrik* nya untuk dibuat fungsi eksternal

```
function E=kali(A,B)
dim=size(A);
m=dim(1);
p=dim(2);
dim=size(B);
n=dim(2);
for i=1:m
   for j=1:n
      E(i,j)=0;
   end
end

for i=1:m
   for j=1:n
      for k=1:p
         E(i,j)=E(i,j)+A(i,k)*B(k,j);
      end
   end
end
```

lalu di-*save* dengan nama *kali.m*, maka terciptalah fungsi eksternal yang bernama fungsi *kali*. Kemudian coba anda uji fungsi *kali* tersebut dengan menjalankan *source code* berikut

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
B = [1 3; 5 9; 2 4]; % inisialisasi matrik B

% ---proses perkalian matrik----
E = kali(A,B)

% ---menampilkan matrik A, B dan E----
A
B
E
```

Silakan anda periksa hasil perhitungannya. Pasti betul! Anda bisa mencoba perkalian matrik lainnya dengan menggunakan source code tersebut. Bahkan anda bisa mengganti nama matriknya untuk selain **A**, **B** dan **E**.

2.4 Fungsi eksternal perkalian matrik dan vektor-kolom

Mari kita beralih ke perkalian matrik dan vektor-kolom. Kita akan membuat fungsi eksternal untuk perkalian matrik dan vektor-kolom. Berikut ini adalah *source code* perkalian matrik dan vektor-kolom hasil akhir optimasi yang telah ditulis panjang lebar pada bab sebelumnya

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
for i=1:2
   y(i,1)=0;
end

for i=1:2
   for j=1:3
      y(i,1)=y(i,1)+A(i,j)*x(j,1);
   end
end

% ---menampilkan matrik A, B dan E----
A
x
y
```

Source code tersebut digunakan untuk menghitung perkalian matrik dan vektor-kolom berikut

$$\mathbf{y}_{2\times 1} = \mathbf{A}_{2\times 3} \cdot \mathbf{x}_{3\times 1}$$

Dan kita bisa sepakati simbol indeks m dan n untuk men-generalisir dimensi matrik

$$\mathbf{y}_{m\times 1} = \mathbf{A}_{m\times n} \cdot \mathbf{x}_{n\times 1}$$

Dengan demikian, *source code* tersebut dapat dioptimasi menjadi

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
dim=size(A);
m=dim(1);
n=dim(2);
for i=1:m
   y(i,1)=0;
end

for i=1:m
   for j=1:n
      y(i,1)=y(i,1)+A(i,j)*x(j,1);
   end
end

% ---menampilkan matrik A, B dan E----
A
x
y
```

Selanjutnya kita ambil bagian *proses perkalian matrik dan vektor* nya untuk dibuat fungsi eksternal

```
function y=kalivektor(A,x)
dim=size(A);
m=dim(1);
n=dim(2);
for i=1:m
   y(i,1)=0;
end

for i=1:m
   for j=1:n
      y(i,1)=y(i,1)+A(i,j)*x(j,1);
   end
end
```

lalu di-*save* dengan nama *kalivektor.m*, maka terciptalah fungsi eksternal yang bernama fungsi *kalivektor*. Kemudian coba anda uji fungsi *kalivektor* tersebut dengan menjalankan *source code* berikut

```
clear all
clc

A = [3 8 5; 6 4 7];  % inisialisasi matrik A
x = [2; 3; 4];       % inisialisasi vektor x

% ---proses perkalian matrik dan vektor----
```

```
y = kalivektor(A,x);

% ---menampilkan matrik A, B dan E----
A
x
```

Silakan anda periksa hasil perhitungannya. Pasti betul! Anda bisa mencoba perkalian matrik dan vektor-kolom dengan angka elemen yang berbeda menggunakan *source code* tersebut. Bahkan anda bisa mengganti nama matrik dan vektor nya untuk selain **A**, **x** dan **y**.

2.5 Penutup

Ada tiga pilar yang harus dikuasai oleh seorang calon *programmer*. Pertama, ia harus tahu bagaimana cara mendeklarasikan data. Kedua, ia harus tahu bagaimana mendayagunakan *flow-control*, yang dalam bab ini dan bab sebelumnya menggunakan pasangan *for-end*. Dan ketiga, ia harus bisa membuat fungsi eksternal.

Tidak ada yang melarang anda beralih ke Fortran, atau ke Delphi, atau ke C++, atau ke Python, atau bahasa pemrograman apa saja. Saran saya, ketika anda berkenalan dengan suatu bahasa pemrograman, yang pertama kali anda lakukan adalah menguasai ketiga pilar itu. Insya Allah ia akan membantu anda lebih cepat mempelajari bahasa pemrograman yang sedang anda geluti.

Penguasaan atas ketiga pilar tersebut akan mengarahkan *programmer* untuk membuat source code yang bersifat *modular* atau *extention*. Ini adalah modal untuk memasuki apa yang disebut *object oriented programming*.

Sesungguhnya Matlab memiliki banyak fungsi internal yang bisa langsung dipakai. Anda bisa coba sendiri suatu saat nanti. Kekuatan bahasa pemrograman salah satunya terletak pada seberapa kaya dia memilik banyak fungsi. *Library* adalah kata lain untuk fungsi. Jadi, suatu bahasa pemrograman akan semakin unggul bila dia memiliki semakin banyak *library*. Menurut saya, yang terdepan saat ini masih dimenangkan oleh Python. Dengan Python, *source code* anda akan bisa berjalan di Windows, Linux dan Machintos serta beberapa platform lainnya.

Bab 3

Metode Eliminasi Gauss

Objektif :

- ▷ Mengenalkan sistem persamaan linear.
- ▷ Mengenalkan teknik triangularisasi dan substitusi mundur.
- ▷ Aplikasi metode Eliminasi Gauss menggunakan matrik.
- ▷ Membuat algoritma metode Eliminasi Gauss.
- ▷ Menghitung invers matrik menggunakan metode Eliminasi Gauss.

3.1 Sistem persamaan linear

Secara umum, sistem persamaan linear dinyatakan sebagai berikut

$$P_n : \qquad a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \tag{3.1}$$

dimana a dan b merupakan konstanta, x adalah variable, $n = 1, 2, 3, ...$

Berikut ini adalah sistem persamaan linear yang terdiri dari empat buah persamaan yaitu P_1, P_2, P_3, dan P_4

$$\begin{array}{lrcrcrcrcl} P_1: & x_1 & + & x_2 & & & + & 3x_4 & = & 4 \\ P_2: & 2x_1 & + & x_2 & - & x_3 & + & x_4 & = & 1 \\ P_3: & 3x_1 & - & x_2 & - & x_3 & + & 2x_4 & = & \text{-3} \\ P_4: & -x_1 & + & 2x_2 & + & 3x_3 & - & x_4 & = & 4 \end{array}$$

Problem dari sistem persamaan linear adalah bagaimana mencari nilai pengganti bagi variabel x_1, x_2, x_3, dan x_4 sehingga semua persamaan diatas menjadi benar. Langkah awal penyelesaian problem tersebut adalah dengan melakukan penyederhanaan sistem persamaan linear.

3.2 Teknik penyederhanaan

Ada banyak jalan untuk menyederhanakan sistem persamaan linear. Namun tantangannya, kita ingin agar pekerjaan ini dilakukan oleh komputer. Oleh karena itu, kita harus menciptakan

algoritma yang nantinya bisa berjalan di komputer. Untuk mencapai tujuan itu, kita akan berpatokan pada tiga buah aturan operasi matematika, yaitu

- Persamaan P_i dapat dikalikan dengan sembarang konstanta λ, lalu hasilnya ditempatkan di posisi persamaan P_i. Simbol operasi ini adalah $(\lambda P_i) \rightarrow (P_i)$. Contoh

$$P_1 : \qquad x_1 + x_2 + 3x_4 = 4$$

  jika $\lambda = 2$, maka

$$2P_1 : \qquad 2x_1 + 2x_2 + 6x_4 = 8$$

- Persamaan P_j dapat dikalikan dengan sembarang konstanta λ kemudian dijumlahkan dengan persamaan P_i, lalu hasilnya ditempatkan di posisi persamaan P_i. Simbol operasi ini adalah $(P_i - \lambda P_j) \rightarrow (P_i)$. Contoh

$$\begin{aligned} P_2 : \qquad 2x_1 + x_2 - x_3 + x_4 &= 1 \\ 2P_1 : \qquad 2x_1 + 2x_2 + 6x_4 &= 8 \end{aligned}$$

  maka operasi $(P_2 - 2P_1) \rightarrow (P_2)$ mengakibatkan perubahan pada P_2 menjadi

$$P_2 : \qquad -x_2 - x_3 - 5x_4 = -7$$

  Dengan cara ini, maka variabel x_1 berhasil dihilangkan dari P_2. Upaya untuk menghilangkan suatu variabel merupakan tahapan penting dalam metode Eliminasi Gauss.

- Persamaan P_i dan P_j dapat bertukar posisi. Simbol operasi ini adalah $(P_i) \leftrightarrow (P_j)$. Contoh

$$\begin{aligned} P_2 : \qquad 2x_1 + x_2 - x_3 + x_4 &= 1 \\ P_3 : \qquad 3x_1 - x_2 - x_3 + 2x_4 &= -3 \end{aligned}$$

  maka operasi $(P_2) \leftrightarrow (P_3)$ mengakibatkan pertukaran posisi masing-masing persamaan, menjadi

$$\begin{aligned} P_2 : \qquad 3x_1 - x_2 - x_3 + 2x_4 &= -3 \\ P_3 : \qquad 2x_1 + x_2 - x_3 + x_4 &= 1 \end{aligned}$$

3.2.1 Cara menghilangkan sebuah variabel

Sebelum dilanjut, saya ingin mengajak anda untuk fokus memahami aturan operasi yang kedua. Misalnya ada 2 persamaan linear yaitu

$$\begin{aligned} P_1 : \qquad 3x_1 + 2x_2 - 5x_3 + 8x_4 &= 3 \\ P_2 : \qquad 4x_1 + 7x_2 - x_3 + 6x_4 &= 9 \end{aligned}$$

Kemudian anda diminta untuk menghilangkan variabel x_1 dari P_2. Itu artinya, anda diminta untuk memodifikasi P_2 sedemikian rupa sehingga didapat P_2 yang baru, yang didalamnya tidak ada x_1.

Berdasarkan rumus operasi $(P_i - \lambda P_j) \rightarrow (P_i)$, maka operasi yang tepat adalah $(P_2 - \frac{4}{3}P_1) \rightarrow (P_2)$. Perhatikan! Bilangan λ, yaitu $\frac{4}{3}$, harus dikalikan dengan P_1, BUKAN dengan P_2. Sedangkan angka $\frac{4}{3}$ adalah satu-satunya angka yang bisa menghapus variabel x_1 dari P_2 lewat operasi $(P_2 - \frac{4}{3}P_1)$. Selengkapnya adalah sebagai berikut

$$\begin{array}{rrcl} P_2: & 4x_1 + 7x_2 - x_3 + 6x_4 & = & 9 \\ \frac{4}{3}P_1: & \frac{4}{3}3x_1 + \frac{4}{3}2x_2 - \frac{4}{3}5x_3 + \frac{4}{3}8x_4 & = & \frac{4}{3}3 \end{array}$$

Kemudian, hasil operasi $(P_2 - \frac{4}{3}P_1)$ disimpan sebagai P_2 yang baru

$$P_2: \quad \left(4 - \frac{4}{3}3\right)x_1 + \left(7 - \frac{4}{3}2\right)x_2 - \left(1 - \frac{4}{3}5\right)x_3 + \left(6 - \frac{4}{3}8\right)x_4 = \left(9 - \frac{4}{3}3\right)$$

Dengan sendirinya x_1 akan lenyap dari $P2$. Mudah-mudahan jelas sampai disini. Demikianlah cara untuk menghilangkan x_1 dari $P2$.

3.2.2 Permainan indeks

Sekarang, mari kita tinjau hal yang sama, yaitu menghilangkan x_1 dari P_2, namun menggunakan 'permainan' indeks. Secara umum, P_1 dan P_2 bisa dinyatakan sebagai

$$\begin{array}{rrcl} P_1: & a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 & = & a_{15} \\ P_2: & a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 & = & a_{25} \end{array}$$

Agar x_1 hilang dari P_2, operasi yang benar adalah $(P_2 - \lambda P_1) \rightarrow (P_2)$, dimana $\lambda = \frac{a_{21}}{a_{11}}$. Dengan demikian, P_2 yang baru akan memenuhi

$$P_2: \left(a_{21} - \frac{a_{21}}{a_{11}}a_{11}\right)x_1 + \left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \left(a_{23} - \frac{a_{21}}{a_{11}}a_{13}\right)x_3 + \left(a_{24} - \frac{a_{21}}{a_{11}}a_{14}\right)x_4 = \left(a_{25} - \frac{a_{21}}{a_{11}}a_{15}\right)$$

Perhatikanlah variasi indeks pada persamaan diatas. Semoga intuisi anda bisa menangkap keberadaan suatu pola perubahan indeks. Jika belum, mari kita kembangkan persoalan ini.

Sekarang saya ketengahkan kehadapan anda tiga buah persamaan, yaitu P_1, P_2 dan P_3

$$\begin{array}{rrcl} P_1: & a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 & = & a_{15} \\ P_2: & a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 & = & a_{25} \\ P_3: & a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 & = & a_{35} \end{array}$$

Bagaimana cara menghilangkan x_1 dari P_3 dengan memanfaatkan P_1??

Begini caranya, $(P_3 - \lambda P_1) \rightarrow (P_3)$, dengan $\lambda = \frac{a_{31}}{a_{11}}$..

$$P_3 : \left(a_{31} - \frac{a_{31}}{a_{11}}a_{11}\right)x_1 + \left(a_{32} - \frac{a_{31}}{a_{11}}a_{12}\right)x_2 + \left(a_{33} - \frac{a_{31}}{a_{11}}a_{13}\right)x_3 + \left(a_{34} - \frac{a_{31}}{a_{11}}a_{14}\right)x_4 = \left(a_{35} - \frac{a_{31}}{a_{11}}a_{15}\right)$$

Mudah-mudahan, pola perubahan indeksnya semakin jelas terlihat. Selanjutnya jika ada persamaan P_4 yang ingin dihilangkan $x_1$nya dengan memanfaatkan P_1, bagaimana caranya?

Tentu saja operasinya adalah $(P_4 - \lambda P_1) \rightarrow (P_4)$, dengan $\lambda = \frac{a_{41}}{a_{11}}$

$$P_4 : \left(a_{41} - \frac{a_{41}}{a_{11}}a_{11}\right)x_1 + \left(a_{42} - \frac{a_{41}}{a_{11}}a_{12}\right)x_2 + \left(a_{43} - \frac{a_{41}}{a_{11}}a_{13}\right)x_3 + \left(a_{44} - \frac{a_{41}}{a_{11}}a_{14}\right)x_4 = \left(a_{45} - \frac{a_{41}}{a_{11}}a_{15}\right)$$

3.3 Triangularisasi dan Substitusi Mundur

3.3.1 Contoh pertama

Sekarang, mari kita kembali kepada sistem persamaan linear yang sudah ditulis di awal bab ini

$$\begin{array}{lrcrcrcrcl} P_1 : & x_1 & + & x_2 & & & + & 3x_4 & = & 4 \\ P_2 : & 2x_1 & + & x_2 & - & x_3 & + & x_4 & = & 1 \\ P_3 : & 3x_1 & - & x_2 & - & x_3 & + & 2x_4 & = & \text{-3} \\ P_4 : & -x_1 & + & 2x_2 & + & 3x_3 & - & x_4 & = & 4 \end{array}$$

Sekali lagi saya tegaskan bahwa problem dari sistem persamaan linear adalah bagaimana mendapatkan angka-angka yang bisa menggantikan variabel x_1, x_2, x_3, dan x_4 sehingga semua persamaan di atas menjadi benar. Dengan berpegang pada ketiga teknik penyederhanaan tadi, maka sistem persamaan linear di atas dapat disederhanakan dengan langkah-langkah berikut ini:

1. Gunakan persamaan P_1 untuk menghilangkan variabel x_1 dari persamaan P_2, P_3 dan P_4 dengan cara $(P_2 - 2P_1) \rightarrow (P_2)$, $(P_3 - 3P_1) \rightarrow (P_3)$ dan $(P_4 + P_1) \rightarrow (P_4)$. Hasilnya akan seperti ini

$$\begin{array}{rrcl} P_1 : & x_1 + x_2 + 3x_4 & = & 4, \\ P_2 : & -x_2 - x_3 - 5x_4 & = & -7, \\ P_3 : & -4x_2 - x_3 - 7x_4 & = & -15, \\ P_4 : & 3x_2 + 3x_3 + 2x_4 & = & 8 \end{array}$$

   Silakan anda cermati bahwa x_1 kini telah hilang dari P_2, P_3 dan P_4.

2. Gunakan persamaan P_2 untuk menghilangkan variabel x_2 dari persamaan P_3 dan P_4

dengan cara $(P_3 - 4P_2) \rightarrow (P_3)$ dan $(P_4 + 3P_2) \rightarrow (P_4)$. Hasilnya akan seperti ini

$$\begin{aligned} P_1: \quad x_1 + x_2 + 3x_4 &= 4, \\ P_2: \quad -x_2 - x_3 - 5x_4 &= -7, \\ P_3: \quad 3x_3 + 13x_4 &= 13, \\ P_4: \quad -13x_4 &= -13 \end{aligned}$$

Kalau x_3 masih ada di persamaan P_4, dibutuhkan satu operasi lagi untuk menghilangkannya. Namun hasil operasi pada langkah ke-2 ternyata sudah otomatis menghilangkan x_3 dari P_4. Bentuk akhir dari sistem persamaan linear di atas, dikenal sebagai bentuk **triangular**.

Sampai dengan langkah ke-2 ini, kita berhasil mendapatkan sistem persamaan linear yang lebih sederhana. Apa yang dimaksud dengan sederhana dalam konteks ini? Suatu sistem persamaan linear dikatakan sederhana bila kita bisa mendapatkan seluruh nilai pengganti variabelnya dengan cara yang lebih mudah atau dengan usaha yang tidak memakan waktu lama dibandingkan sebelum disederhanakan.

3. Selanjutnya kita jalankan proses **backward-substitution** untuk mendapatkan angka-angka pengganti bagi x_1, x_2, x_3 dan x_4. Melalui proses **backward-substitution**, yang pertama kali didapat adalah angka pengganti bagi variabel x_4, kemudian x_3, lalu diikuti x_2, dan akhirnya x_1. Silakan cermati yang berikut ini

$$\begin{aligned} P_4: \quad x_4 = \frac{-13}{-13} &= 1, \\ P_3: \quad x_3 = \frac{1}{3}(13 - 13x_4) &= \frac{1}{3}(13 - 13) = 0, \\ P_2: \quad x_2 = -(-7 + 5x_4 + x_3) &= -(-7 + 5 + 0) = 2, \\ P_1: \quad x_1 = 4 - 3x_4 - x_2 &= 4 - 3 - 2 = -1 \end{aligned}$$

Jadi solusinya adalah $x_1 = -1$, $x_2 = 2$, $x_3 = 0$ dan $x_4 = 1$. Coba sekarang anda cek, apakah semua solusi ini cocok dan tepat bila dimasukan ke sistem persamaan linear yang pertama, yaitu yang belum disederhanakan?

OK, mudah-mudahan ngerti ya... Kalau belum paham, coba dibaca sekali lagi. Atau, sekarang kita beralih kecontoh yang lain.

3.3.2 Contoh kedua

Diketahui sistem persamaan linear, terdiri dari empat buah persamaan yaitu P_1, P_2, P_3, dan P_4 seperti berikut ini:

$$\begin{array}{lrcrcrcrcr} P_1: & x_1 & - & x_2 & + & 2x_3 & - & x_4 & = & \text{-8} \\ P_2: & 2x_1 & - & 2x_2 & + & 3x_3 & - & 3x_4 & = & \text{-20} \\ P_3: & x_1 & + & x_2 & + & x_3 & & & = & \text{-2} \\ P_4: & x_1 & - & x_2 & + & 4x_3 & + & 3x_4 & = & 4 \end{array}$$

Seperti contoh pertama, solusi sistem persamaan linear di atas akan dicari dengan langkah-langkah berikut ini:

1. Gunakan persamaan P_1 untuk menghilangkan x_1 dari persamaan P_2, P_3 dan P_4 dengan cara $(P_2 - 2P_1) \rightarrow (P_2)$, $(P_3 - P_1) \rightarrow (P_3)$ dan $(P_4 - P_1) \rightarrow (P_4)$. Hasilnya akan seperti ini

$$\begin{aligned} P_1: \quad & x_1 - x_2 + 2x_3 - x_4 = -8, \\ P_2: \quad & -x_3 - x_4 = -4, \\ P_3: \quad & 2x_2 - x_3 + x_4 = 6, \\ P_4: \quad & 2x_3 + 4x_4 = 12 \end{aligned}$$

   Perhatikan persamaan P_2! Akibat dari langkah yang pertama tadi, ternyata tidak hanya x_1 saja yang hilang dari persamaan P_2, variabel x_2 pun turut hilang dari persamaan P_2. Kondisi ini bisa menggagalkan proses triangularisasi. Untuk itu, posisi P_2 mesti ditukar dengan persamaan yang berada dibawahnya, yang masih memiliki variabel x_2. Maka yang paling cocok adalah ditukar dengan P_3.

2. Tukar posisi persamaan P_2 dengan persamaan P_3, $(P_2 \leftrightarrow P_3)$. Hasilnya akan seperti ini

$$\begin{aligned} P_1: \quad & x_1 - x_2 + 2x_3 - x_4 = -8, \\ P_2: \quad & 2x_2 - x_3 + x_4 = 6, \\ P_3: \quad & -x_3 - x_4 = -4, \\ P_4: \quad & 2x_3 + 4x_4 = 12 \end{aligned}$$

3. Agar sistem persamaan linear di atas menjadi berbentuk triangular, maka kita harus menghilangkan variabel x_3 dari persamaan P_4. Karenanya, gunakan persamaan P_3 untuk menghilangkan x_3 dari persamaan P_4 dengan cara $(P_4 + 2P_3) \rightarrow (P_4)$. Hasilnya akan seperti ini

$$\begin{aligned} P_1: \quad & x_1 - x_2 + 2x_3 - x_4 = -8, \\ P_2: \quad & 2x_2 - x_3 + x_4 = 6, \\ P_3: \quad & -x_3 - x_4 = -4, \\ P_4: \quad & 2x_4 = 4 \end{aligned}$$

Sampai disini proses triangularisasi telah selesai.

4. Selanjutnya adalah proses backward-substitution. Melalui proses ini, yang pertama kali didapat solusinya adalah x_4, kemudian x_3, lalu diikuti x_2, dan akhirnya x_1.

$$\begin{aligned} P_4: \quad & x_4 = \frac{4}{2} &&= 2, \\ P_3: \quad & x_3 = \frac{-4 + x_4}{-1} &&= 2, \\ P_2: \quad & x_2 = \frac{6 + x_3 - x_4}{2} &&= 3, \\ P_1: \quad & x_1 = -8 + x_2 - 2x_3 + x_4 &&= -7 \end{aligned}$$

Jadi solusinya adalah $x_1 = -7$, $x_2 = 3$, $x_3 = 2$ dan $x_4 = 2$.

Berdasarkan kedua contoh di atas, untuk mendapatkan solusi sistem persamaan linear, diperlukan operasi **triangularisasi** dan proses **backward-substitution**. Kata **backward-substitution** kalau diterjemahkan kedalam bahasa indonesia, menjadi **substitusi-mundur**. Gabungan proses triangularisasi dan substitusi-mundur untuk menyelesaikan sistem persamaan linear dikenal sebagai metode **Eliminasi Gauss**.

3.4 Matrik dan Eliminasi Gauss

3.4.1 Matrik Augmentasi

Sejumlah matrik bisa digunakan untuk menyatakan suatu sistem persamaan linear. Sejenak, mari kita kembali lagi melihat sistem persamaan linear secara umum seperti berikut ini:

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

Sementara, kalau dinyatakan dalam bentuk operasi matrik, maka akan seperti ini:

$$\begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3.2}$$

Dalam mencari solusi suatu sistem persamaan linear dengan metode eliminasi gauss, bentuk operasi matrik di atas dimanipulasi menjadi **matrik augment**, yaitu suatu matrik yang beruku-

ran n x $(n+1)$ seperti berikut ini:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & a_{1,n+1} \\ a_{21} & a_{22} & \dots & a_{2n} & a_{2,n+1} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & a_{n,n+1} \end{array}\right] \tag{3.3}$$

Inilah source code Matlab untuk membentuk matrik augmentasi yang terdiri atas matrik A dan vektor b,

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 0 3
   2 1 -1 1
   3 -1 -1 2
   -1 2 3 -1];

%---- inisialisasi vektor b ----
b = [4 ; 1 ; -3 ; 4];

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end
```

3.4.2 Penerapan pada contoh pertama

Pada contoh pertama di atas, diketahui sistem persamaan linear yang terdiri dari empat buah persamaan yaitu P_1, P_2, P_3, dan P_4

$$\begin{array}{lrcrcrcrcl} P_1: & x_1 & + & x_2 & & & + & 3x_4 & = & 4 \\ P_2: & 2x_1 & + & x_2 & - & x_3 & + & x_4 & = & 1 \\ P_3: & 3x_1 & - & x_2 & - & x_3 & + & 2x_4 & = & \text{-3} \\ P_4: & -x_1 & + & 2x_2 & + & 3x_3 & - & x_4 & = & 4 \end{array}$$

Sistem persamaan linear tersebut dapat dinyatakan dalam operasi matrik

$$\left[\begin{array}{cccc} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\right] = \left[\begin{array}{c} 4 \\ 1 \\ -3 \\ 4 \end{array}\right]$$

Setelah itu matrik augment disusun seperti ini (perhatikan angka-angka indeks pada matriks

disebelahnya)

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{array}\right]$$

Kemudian kita lakukan operasi triangularisai terhadap matrik augment, dimulai dari kolom pertama (yang tujuannya untuk menghilangkan variabel x_1 dari P_2, P_3, dan P_4), yaitu

$$P_2 : \left(a_{21} - \frac{a_{21}}{a_{11}}a_{11}\right)x_1 + \left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \left(a_{23} - \frac{a_{21}}{a_{11}}a_{13}\right)x_3 + \left(a_{24} - \frac{a_{21}}{a_{11}}a_{14}\right)x_4 = \left(a_{25} - \frac{a_{21}}{a_{11}}a_{15}\right)$$

$$P_3 : \left(a_{31} - \frac{a_{31}}{a_{11}}a_{11}\right)x_1 + \left(a_{32} - \frac{a_{31}}{a_{11}}a_{12}\right)x_2 + \left(a_{33} - \frac{a_{31}}{a_{11}}a_{13}\right)x_3 + \left(a_{34} - \frac{a_{31}}{a_{11}}a_{14}\right)x_4 = \left(a_{35} - \frac{a_{31}}{a_{11}}a_{15}\right)$$

$$P_4 : \left(a_{41} - \frac{a_{41}}{a_{11}}a_{11}\right)x_1 + \left(a_{42} - \frac{a_{41}}{a_{11}}a_{12}\right)x_2 + \left(a_{43} - \frac{a_{41}}{a_{11}}a_{13}\right)x_3 + \left(a_{44} - \frac{a_{41}}{a_{11}}a_{14}\right)x_4 = \left(a_{45} - \frac{a_{41}}{a_{11}}a_{15}\right)$$

Sekarang akan saya tulis *source code* Matlab untuk menyelesaikan perhitungan diatas. Saran saya, anda jangan hanya duduk sambil membaca buku ini, kalau bisa nyalakan komputer/laptop dan ketik ulang *source-code* ini agar anda memperoleh *feeling*-nya! OK, mari kita mulai..

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 0 3
   2 1 -1 1
   3 -1 -1 2
   -1 2 3 -1];

%---- inisialisasi vektor b ----
b = [4 ; 1 ; -3 ; 4];

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end

%---- menghilangkan variabel x1 ----
m=A(2,1)/A(1,1);          % huruf m mewakili simbol lambda
A(2,1)=A(2,1)-m*A(1,1);
A(2,2)=A(2,2)-m*A(1,2);
A(2,3)=A(2,3)-m*A(1,3);
A(2,4)=A(2,4)-m*A(1,4);
A(2,5)=A(2,5)-m*A(1,5);

m=A(3,1)/A(1,1);
A(3,1)=A(3,1)-m*A(1,1);
A(3,2)=A(3,2)-m*A(1,2);
A(3,3)=A(3,3)-m*A(1,3);
```

```
A(3,4)=A(3,4)-m*A(1,4);
A(3,5)=A(3,5)-m*A(1,5);

m=A(4,1)/A(1,1);
A(4,1)=A(4,1)-m*A(1,1);
A(4,2)=A(4,2)-m*A(1,2);
A(4,3)=A(4,3)-m*A(1,3);
A(4,4)=A(4,4)-m*A(1,4);
A(4,5)=A(4,5)-m*A(1,5);
```

Hasilnya akan seperti ini

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 0 & -1 & -1 & -5 & -7 \\ 0 & -4 & -1 & -7 & -15 \\ 0 & 3 & 3 & 2 & 8 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{array}\right]$$

Pada kolom pertama, seluruh elemen berubah menjadi nol ($a_{21} = 0$, $a_{31} = 0$, dan $a_{41} = 0$) kecuali elemen yang paling atas a_{11}. Itu berarti kita sudah menghilangkan variabel x_1 dari P_2, P_3, dan P_4. Sekarang dilanjutkan ke kolom kedua, dengan operasi yang hampir sama, untuk membuat elemen a_{32} dan a_{42} bernilai nol

$$P_3 : \left(a_{31} - \frac{a_{32}}{a_{22}}a_{21}\right)x_1 + \left(a_{32} - \frac{a_{32}}{a_{22}}a_{22}\right)x_2 + \left(a_{33} - \frac{a_{32}}{a_{22}}a_{23}\right)x_3 + \left(a_{34} - \frac{a_{32}}{a_{22}}a_{24}\right)x_4 = \left(a_{35} - \frac{a_{32}}{a_{22}}a_{25}\right)$$

$$P_4 : \left(a_{41} - \frac{a_{42}}{a_{22}}a_{21}\right)x_1 + \left(a_{42} - \frac{a_{42}}{a_{22}}a_{22}\right)x_2 + \left(a_{43} - \frac{a_{42}}{a_{22}}a_{23}\right)x_3 + \left(a_{44} - \frac{a_{42}}{a_{22}}a_{24}\right)x_4 = \left(a_{45} - \frac{a_{42}}{a_{22}}a_{25}\right)$$

Source-code berikut ini adalah kelanjutan dari *source-code* diatas. Jadi jangan dipisah dalam file lain!!!

```
m=A(3,2)/A(2,2);
A(3,1)=A(3,1)-m*A(2,1);
A(3,2)=A(3,2)-m*A(2,2);
A(3,3)=A(3,3)-m*A(2,3);
A(3,4)=A(3,4)-m*A(2,4);
A(3,5)=A(3,5)-m*A(2,5);

m=A(4,2)/A(2,2);
A(4,1)=A(4,1)-m*A(2,1);
A(4,2)=A(4,2)-m*A(2,2);
A(4,3)=A(4,3)-m*A(2,3);
A(4,4)=A(4,4)-m*A(2,4);
A(4,5)=A(4,5)-m*A(2,5);

```

Hasilnya akan seperti dibawah ini. Itu berarti kita telah menghilangkan variabel x_2 dari P_3,

dan P_4; bahkan tanpa disengaja x_3 juga hilang dari P_4. Inilah bentuk **triangular**

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 0 & -1 & -1 & -5 & -7 \\ 0 & 0 & 3 & 13 & 13 \\ 0 & 0 & 0 & -13 & -13 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{array}\right]$$

Walaupun x_3 sudah hilang dari P_4, sebaiknya source-code penghapusan x_3 dari P_4 tetap ditambahkan pada source-code sebelumnya agar source-code tersebut menjadi lengkap.

```
m=A(4,3)/A(3,3);
A(4,1)=A(4,1)-m*A(3,1);
A(4,2)=A(4,2)-m*A(3,2);
A(4,3)=A(4,3)-m*A(3,3);
A(4,4)=A(4,4)-m*A(3,4);
A(4,5)=A(4,5)-m*A(3,5);
```

Dengan memperhatikan angka-angka indeks pada matrik augment di atas, kita akan mencoba membuat rumusan proses substitusi-mundur untuk mendapatkan seluruh nilai pengganti variabel x. Dimulai dari x_4,

$$x_4 = \frac{a_{45}}{a_{44}} = \frac{-13}{-13} = 1$$

lalu dilanjutkan dengan x_3, x_2, dan x_1.

$$\begin{aligned} x_3 = \frac{a_{35} - a_{34}x_4}{a_{33}} &= \frac{13 - [(13)(1)]}{3} = 0 \\ x_2 = \frac{a_{25} - (a_{23}x_3 + a_{24}x_4)}{a_{22}} &= \frac{(-7) - [(-1)(0) + (-5)(1)]}{(-1)} = 2 \\ x_1 = \frac{a_{15} - (a_{12}x_2 + a_{13}x_3 + a_{14}x_4)}{a_{11}} &= \frac{4 - [(1)(2) + (0)(0) + (3)(1)]}{1} = -1 \end{aligned}$$

Inilah *source code* proses substitusi mundur sesuai rumusan di atas

```
x(4,1)=A(4,5)/A(4,4);
x(3,1)=(A(3,5)-A(3,4)*x(4,1))/A(3,3);
x(2,1)=(A(2,5)-(A(2,3)*x(3,1)+A(2,4)*x(4,1)))/A(2,2);
x(1,1)=(A(1,5)-(A(1,2)*x(2,1)+A(1,3)*x(3,1)+A(1,4)*x(4,1)))/A(1,1);
```

3.4.3 *Source-code* dasar

Proses triangularisasi dan substitusi-mundur dibakukan menjadi algoritma metode eliminasi gauss. Berikut ini saya tampilkan *source-code* dalam Matlab sebagaimana langkah-langkah diatas

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 0 3
```

```
   2 1 -1 1
   3 -1 -1 2
   -1 2 3 -1];

%---- inisialisasi vektor b ----
b = [4 ; 1 ; -3 ; 4];

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end

%==== Proses Triangularisasi ====
%---- menghilangkan variabel x1 dari P2, P3 dan P4 ----
m=A(2,1)/A(1,1);          % huruf m mewakili simbol lambda
A(2,1)=A(2,1)-m*A(1,1);
A(2,2)=A(2,2)-m*A(1,2);
A(2,3)=A(2,3)-m*A(1,3);
A(2,4)=A(2,4)-m*A(1,4);
A(2,5)=A(2,5)-m*A(1,5);

m=A(3,1)/A(1,1);
A(3,1)=A(3,1)-m*A(1,1);
A(3,2)=A(3,2)-m*A(1,2);
A(3,3)=A(3,3)-m*A(1,3);
A(3,4)=A(3,4)-m*A(1,4);
A(3,5)=A(3,5)-m*A(1,5);

m=A(4,1)/A(1,1);
A(4,1)=A(4,1)-m*A(1,1);
A(4,2)=A(4,2)-m*A(1,2);
A(4,3)=A(4,3)-m*A(1,3);
A(4,4)=A(4,4)-m*A(1,4);
A(4,5)=A(4,5)-m*A(1,5);

%---- menghilangkan variabel x2 dari P3 dan P4 ----
m=A(3,2)/A(2,2);
A(3,1)=A(3,1)-m*A(2,1);
A(3,2)=A(3,2)-m*A(2,2);
A(3,3)=A(3,3)-m*A(2,3);
A(3,4)=A(3,4)-m*A(2,4);
A(3,5)=A(3,5)-m*A(2,5);

m=A(4,2)/A(2,2);
A(4,1)=A(4,1)-m*A(2,1);
A(4,2)=A(4,2)-m*A(2,2);
A(4,3)=A(4,3)-m*A(2,3);
A(4,4)=A(4,4)-m*A(2,4);
A(4,5)=A(4,5)-m*A(2,5);

%---- menghilangkan variabel x3 dari P4 ----
m=A(4,3)/A(3,3);
A(4,1)=A(4,1)-m*A(3,1);
A(4,2)=A(4,2)-m*A(3,2);
A(4,3)=A(4,3)-m*A(3,3);
A(4,4)=A(4,4)-m*A(3,4);
A(4,5)=A(4,5)-m*A(3,5);
```

```
65
66 %==== Proses Substitusi Mundur ====
67 x(4,1)=A(4,5)/A(4,4);
68 x(3,1)=(A(3,5)-A(3,4)*x(4,1))/A(3,3);
69 x(2,1)=(A(2,5)-(A(2,3)*x(3,1)+A(2,4)*x(4,1)))/A(2,2);
70 x(1,1)=(A(1,5)-(A(1,2)*x(2,1)+A(1,3)*x(3,1)+A(1,4)*x(4,1)))/A(1,1);
```

3.4.4 Optimasi *source code*

Singkatnya, tujuan dari dilakukannya proses optimasi adalah untuk memperkecil jumlah baris statemen pada *source code* dasar. Seperti kita ketahui bersama, *source code* dasar eliminasi gauss yang tertulis di atas terdiri atas 70 baris statemen, sehingga perlu dilakukan proses optimasi untuk memperkecil jumlah baris statemen (tanpa menyalahi hasil perhitungan).

3.4.4.1 Optimasi *source code* bagian triangular

Langkah optimasi source code bagian triangularisasi dimulai dari baris statemen ke 23 hingga ke 27, yaitu

```
m=A(2,1)/A(1,1);          % huruf m mewakili simbol lambda
A(2,1)=A(2,1)-m*A(1,1);
A(2,2)=A(2,2)-m*A(1,2);
A(2,3)=A(2,3)-m*A(1,3);
A(2,4)=A(2,4)-m*A(1,4);
A(2,5)=A(2,5)-m*A(1,5);
```

Bagian ini dapat dioptimasi menjadi

```
for k = 1:5
   A(2,k) = A(2,k)-m*A(1,k);
end
```

Langkah optimasi yang sama juga bisa diterapkan untuk rangkaian baris statemen dari baris ke 30 hingga 34 dan baris ke 37 hingga 41 (yang terdapat pada *source-code* dasar), sehingga masing-masing akan menjadi

```
for k = 1:5
   A(3,k) = A(3,k)-m*A(1,k);
end
```

dan

```
for k = 1:5
   A(4,k) = A(4,k)-m*A(1,k);
end
```

Ternyata, pola optimasi yang sama juga masih bisa ditemui mulai baris ke 45 hingga baris statemen ke 64. Dengan demikian, setidaknya, tahapan pertama ini akan menghasil *source-code* baru hasil optimasi awal yaitu

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 0 3
   2 1 -1 1
   3 -1 -1 2
   -1 2 3 -1];

%---- inisialisasi vektor b ----
b = [4 ; 1 ; -3 ; 4];

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end

%==== Proses Triangularisasi ====
%---- menghilangkan variabel x1 dari P2, P3 dan P4 ----
m=A(2,1)/A(1,1);          % huruf m mewakili simbol lambda
for k = 1:5
   A(2,k) = A(2,k)-m*A(1,k);
end

m=A(3,1)/A(1,1);
for k = 1:5
   A(3,k) = A(3,k)-m*A(1,k);
end

m=A(4,1)/A(1,1);
for k = 1:5
   A(4,k) = A(4,k)-m*A(1,k);
end

%---- menghilangkan variabel x2 dari P3 dan P4 ----
m=A(3,2)/A(2,2);
for k = 1:5
   A(3,k) = A(3,k)-m*A(2,k);
end

m=A(4,2)/A(2,2);
for k = 1:5
   A(4,k) = A(4,k)-m*A(2,k);
end

%---- menghilangkan variabel x3 dari P4 ----
m=A(4,3)/A(3,3);
for k = 1:5
   A(4,k) = A(4,k)-m*A(3,k);
end

%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);
x(3,1)=(A(3,5)-A(3,4)*x(4,1))/A(3,3);
x(2,1)=(A(2,5)-(A(2,3)*x(3,1)+A(2,4)*x(4,1)))/A(2,2);
x(1,1)=(A(1,5)-(A(1,2)*x(2,1)+A(1,3)*x(3,1)+A(1,4)*x(4,1)))/A(1,1);
```

Sekarang, *source-code* eliminasi gauss telah mengecil menjadi hanya 58 baris statemen saja (sebelumnya ada 70 baris statemen). Namun ini belum merupakan akhir proses optimasi. *Source-code* yang terakhir ini masih bisa dioptimasi kembali.

Coba anda perhatikan pola yang nampak mulai pada baris statemen ke-22 hingga ke-35. Optimasi tahap dua dilakukan untuk menyederhanakan bagian tersebut, yaitu

```
for j = 2:4
   m=A(j,1)/A(1,1);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(1,k);
   end
end
```

Demikian halnya untuk baris ke-38 sampai baris ke-46

```
for j = 3:4
   m=A(j,2)/A(2,2);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(2,k);
   end
end
```

serta baris ke-49 hingga baris ke-52

```
for j = 4:4
   m=A(j,3)/A(3,3);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(3,k);
   end
end
```

Dengan demikian hasil optimasi sampai dengan tahap ini adalah

```
1  clear all
2  clc
3
4  %---- inisialisasi matrik A ----
5  A = [1 1 0 3
6     2 1 -1 1
7     3 -1 -1 2
8     -1 2 3 -1];
9
10 %---- inisialisasi vektor b ----
11 b = [4 ; 1 ; -3 ; 4];
12
13 %---- membentuk matrik augmentasi ----
14 dim = size(A);
15 n = dim(1);
16 for i = 1:n
17    A(i,n+1) = b(i);
18 end
19
20 %==== Proses Triangularisasi ====
21 %---- menghilangkan variabel x1 dari P2, P3 dan P4 ----
```

```
for j = 2:4
   m=A(j,1)/A(1,1);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(1,k);
   end
end

%---- menghilangkan variabel x2 dari P3 dan P4 ----
for j = 3:4
   m=A(j,2)/A(2,2);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(2,k);
   end
end

%---- menghilangkan variabel x3 dari P4 ----
for j = 4:4
   m=A(j,3)/A(3,3);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(3,k);
   end
end

%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);
x(3,1)=(A(3,5)-A(3,4)*x(4,1))/A(3,3);
x(2,1)=(A(2,5)-(A(2,3)*x(3,1)+A(2,4)*x(4,1)))/A(2,2);
x(1,1)=(A(1,5)-(A(1,2)*x(2,1)+A(1,3)*x(3,1)+A(1,4)*x(4,1)))/A(1,1);
```

Jika saya munculkan indeks i pada bagian proses triangularisasi

```
%==== Proses Triangularisasi ====
%---- menghilangkan variabel x1 dari P2, P3 dan P4 ----
i = 1;
for j = i+1:4
   m=A(j,i)/A(i,i);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(i,k);
   end
end

%---- menghilangkan variabel x2 dari P3 dan P4 ----
i = 2;
for j = i+1:4
   m=A(j,i)/A(i,i);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(i,k);
   end
end

%---- menghilangkan variabel x3 dari P4 ----
i = 3;
for j = i+1:4
   m=A(j,i)/A(i,i);
   for k = 1:5
      A(j,k) = A(j,k)-m*A(i,k);
   end
end
```

maka saya bisa gabungkan semua i tersebut menjadi

```
%==== Proses Triangularisasi ====
for i = 1:3
   for j = i+1:4
      m=A(j,i)/A(i,i);
      for k = 1:5
         A(j,k) = A(j,k)-m*A(i,k);
      end
   end
end
```

Sehingga hasil optimasi sampai tahapan ini telah mengecilkan jumlah baris statemen dari semula 70 baris menjadi hanya 34 baris saja. Inilah hasilnya

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 0 3
   2 1 -1 1
   3 -1 -1 2
   -1 2 3 -1];

%---- inisialisasi vektor b ----
b = [4 ; 1 ; -3 ; 4];

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end

%==== Proses Triangularisasi ====
for i = 1:3
   for j = i+1:4
      m=A(j,i)/A(i,i);
      for k = 1:5
         A(j,k) = A(j,k)-m*A(i,k);
      end
   end
end

%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);
x(3,1)=(A(3,5)-A(3,4)*x(4,1))/A(3,3);
x(2,1)=(A(2,5)-(A(2,3)*x(3,1)+A(2,4)*x(4,1)))/A(2,2);
x(1,1)=(A(1,5)-(A(1,2)*x(2,1)+A(1,3)*x(3,1)+A(1,4)*x(4,1)))/A(1,1);
```

3.4.4.2 Optimasi *source code* bagian substitusi-mundur

OK, sekarang kita beralih ke bagian substitusi-mundur. Saya mulai dengan memodifikasi bagian tersebut menjadi seperti ini

```
%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);

S = 0;
S = S + A(3,4)*x(4,1);
x(3,1)=(A(3,5)-S)/A(3,3);

S = 0;
S = S + A(2,3)*x(3,1);
S = S + A(2,4)*x(4,1);
x(2,1)=(A(2,5)-S)/A(2,2);

S = 0;
S = S + A(1,2)*x(2,1);
S = S + A(1,3)*x(3,1);
S = S + A(1,4)*x(4,1);
x(1,1)=(A(1,5)-S)/A(1,1);
```

Dari situ, saya modifikasi kembali menjadi seperti ini

```
%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);

S = 0;
for k = 4:4
   S = S + A(3,k)*x(k,1);
end
x(3,1)=(A(3,5)-S)/A(3,3);

S = 0;
for k = 3:4
   S = S + A(2,k)*x(k,1);
end
x(2,1)=(A(2,5)-S)/A(2,2);

S = 0;
for k = 2:4
   S = S + A(1,k)*x(k,1);
end
x(1,1)=(A(1,5)-S)/A(1,1);
```

Lalu saya munculkan indeks i, coba perhatikan dengan teliti

```
%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);

i = 3;
S = 0;
for k = i+1:4
   S = S + A(i,k)*x(k,1);
end
x(i,1)=(A(i,5)-S)/A(i,i);

i = 2;
S = 0;
for k = i+1:4
   S = S + A(i,k)*x(k,1);
```

```
end
x(i,1)=(A(i,5)-S)/A(i,i);

i = 1;
S = 0;
for k = i+1:4
   S = S + A(i,k)*x(k,1);
end
x(i,1)=(A(1,5)-S)/A(i,i);
```

dengan demikian saya bisa ringkas menjadi seperti ini

```
%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);

for i = 3:-1:1
   S = 0;
   for k = i+1:4
      S = S + A(i,k)*x(k,1);
   end
   x(i,1)=(A(i,5)-S)/A(i,i);
end
```

Dan inilah hasil optimasi sampai tahapan yang terakhir

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 0 3
   2 1 -1 1
   3 -1 -1 2
   -1 2 3 -1];

%---- inisialisasi vektor b ----
b = [4 ; 1 ; -3 ; 4];

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end

%==== Proses Triangularisasi ====
for i = 1:3
   for j = i+1:4
      m=A(j,i)/A(i,i);
      for k = 1:5
         A(j,k) = A(j,k)-m*A(i,k);
      end
   end
end

%==== Proses Substitusi Mundur ====
x(4,1)=A(4,5)/A(4,4);

```

```
for i = 3:-1:1
   S = 0;
   for k = i+1:4
      S = S + A(i,k)*x(k,1);
   end
   x(i,1)=(A(i,5)-S)/A(i,i);
end
```

3.4.5 Pentingnya nilai *n*

Pada baris ke-15, nilai n adalah nilai ukuran matrik A yang berbentuk bujursangkar. Dalam contoh ini, *n* bernilai 4. Dengan menggunakan angka 4 (atau n) sebagai acuan, maka source code hasil optimasi terakhir dimodifikasi kembali menjadi seperti ini

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 0 3
   2 1 -1 1
   3 -1 -1 2
   -1 2 3 -1];

%---- inisialisasi vektor b ----
b = [4 ; 1 ; -3 ; 4];

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end

%==== Proses Triangularisasi ====
for i = 1:n-1
   for j = i+1:n
      m=A(j,i)/A(i,i);
      for k = 1:n+1
         A(j,k) = A(j,k)-m*A(i,k);
      end
   end
end

%==== Proses Substitusi Mundur ====
x(n,1)=A(n,n+1)/A(n,n);

for i = n-1:-1:1
   S = 0;
   for k = i+1:n
      S = S + A(i,k)*x(k,1);
   end
   x(i,1)=(A(i,n+1)-S)/A(i,i);
end
```

Sekarang, source code di atas akan bisa memproses matrik bujursangkar yang ukurannya sembarang; tidak hanya 4x4. Demikianlah akhir dari proses optimasi yang cukup melelahkan.

3.4.6 Jangan puas dulu..

Walaupun memiliki jumlah baris statemen yang lebih sedikit, *source-code* ini masih mengandung *bug* yang bisa berakibat fatal. Sekarang coba anda ganti angka-angka pada bagian inisialisasi matrik menjadi angka-angka baru yang disesuaikan dengan sistem persamaan linear berikut ini

$$\begin{array}{lccccccccc} P_1: & x_1 & - & x_2 & + & 2x_3 & - & x_4 & = & \text{-8} \\ P_2: & 2x_1 & - & 2x_2 & + & 3x_3 & - & 3x_4 & = & \text{-20} \\ P_3: & x_1 & + & x_2 & + & x_3 & & & = & \text{-2} \\ P_4: & x_1 & - & x_2 & + & 4x_3 & + & 3x_4 & = & 4 \end{array}$$

Saya jamin *source code* yang tadi akan berhenti sebelum tugasnya selesai. Artinya ia gagal menjalankan tugas mencari solusi sistem persamaan linear. Mengapa bisa begitu?

3.4.7 *Pivoting*

Pada baris ke-23, yang merupakan bagian dari proses triangularisasi dalam *source code* di atas, terdapat

```
m=A[j,i]/A[i,i]
```

elemen $A[i,i]$ tentunya tidak boleh bernilai nol. Jika itu terjadi, maka proses triangularisasi otomatis akan berhenti dan itu sekaligus menggagalkan metode eliminasi Gauss. Dilihat dari indeks-nya yang kembar yaitu $[i,i]$, maka tidak diragukan lagi bahwa ia pasti menempati posisi di elemen diagonal dari matrik A. Nama lain elemen ini adalah elemen *pivot*. Jadi apa yang harus dilakukan jika secara tidak disengaja didalam aliran proses terdapat elemen *pivot* yang bernilai nol?

Salah satu cara untuk mengatasinya adalah dengan menukar seluruh elemen yang sebaris dengan elemen diagonal bernilai nol. Ia harus ditukar posisinya dengan baris yang ada dibawahnya, sampai elemen diagonal matrik menjadi tidak nol, $a_{ii} \neq 0$. Cara ini disebut *pivoting*. Penambahan proses *pivoting* kedalam *source code* eliminasi Gauss dimulai dari baris ke-23 sampai baris ke-30 berikut ini

```
1  clear all
2  clc
3
4  %---- inisialisasi matrik A ----
5  A = [1 -1 2 -1
6     2 -2 3 -3
7     1 1 1 0
8     1 -1 4 3];
9
10 %---- inisialisasi vektor b ----
11 b = [-8 ; -20 ; -2 ; 4];
12
13 %---- membentuk matrik augmentasi ----
14 dim = size(A);
15 n = dim(1);
```

```
for i = 1:n
   A(i,n+1) = b(i);
end

%==== Proses Triangularisasi ====
for i = 1:n-1
   %---- awal proses pivoting -----
   if A(i,i) == 0
      for s = 1:n+1
         v = A(i,s);
         u = A(i+1,s);
         A(i,s) = u;
         A(i+1,s) = v;
      end
   end
   %---- akhir proses pivoting -----

   for j = i+1:n
      m=A(j,i)/A(i,i);
      for k = 1:n+1
         A(j,k) = A(j,k)-m*A(i,k);
      end
   end
end

%==== Proses Substitusi Mundur ====
x(n,1)=A(n,n+1)/A(n,n);

for i = n-1:-1:1
   S = 0;
   for k = i+1:n
      S = S + A(i,k)*x(k,1);
   end
   x(i,1)=(A(i,n+1)-S)/A(i,i);
end
```

3.5 Function Eliminasi Gauss

Pendefinisian *function* eliminasi gauss, yang akan diberi nama *elgauss* merupakan langkah paling akhir dari proses optimasi *source code* ini. Berdasarkan *source code* di atas, *function* eliminasi gauss bisa dimulai dari baris ke-13 hingga baris ke-50. Berikut ini adalah cara pendefinisiannya

```
function x=elgauss(A,b)

%---- membentuk matrik augmentasi ----
dim = size(A);
n = dim(1);
for i = 1:n
   A(i,n+1) = b(i);
end

%==== Proses Triangularisasi ====
for i = 1:n-1
   %---- awal proses pivoting -----
   if A(i,i) == 0
      for s = 1:n+1
```

```
15          v = A(i,s);
16          u = A(i+1,s);
17          A(i,s) = u;
18          A(i+1,s) = v;
19       end
20    end
21    %---- akhir proses pivoting -----
22
23    for j = i+1:n
24       m=A(j,i)/A(i,i);
25       for k = 1:n+1
26          A(j,k) = A(j,k)-m*A(i,k);
27       end
28    end
29 end
30
31 %==== Proses Substitusi Mundur ====
32 x(n,1)=A(n,n+1)/A(n,n);
33
34 for i = n-1:-1:1
35    S = 0;
36    for k = i+1:n
37       S = S + A(i,k)*x(k,1);
38    end
39    x(i,1)=(A(i,n+1)-S)/A(i,i);
40 end
```

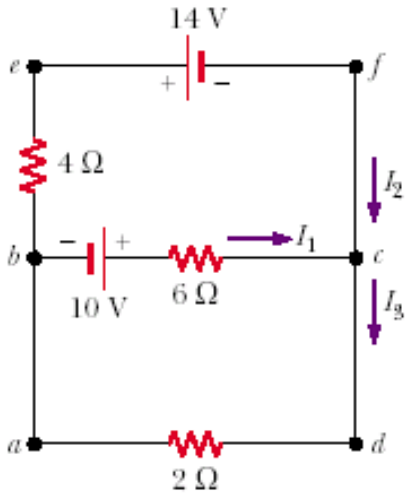
Dengan adanya *function elgauss,* maka *source-code* untuk menyelesaikan sistem persamaan linear dengan metode eliminasi gauss dapat ditulis secara sangat sederhana. Berikut ini contohnya..

```
1  clear all
2  clc
3
4  %---- inisialisasi matrik A ----
5  A = [1 -1 2 -1
6     2 -2 3 -3
7     1 1 1 0
8     1 -1 4 3];
9
10 %---- inisialisasi vektor b ----
11 b = [-8 ; -20 ; -2 ; 4];
12
13 x=elgauss(A,b)
```

3.6 Contoh aplikasi

3.6.1 Menghitung arus listrik

Gunakan metode Eliminasi Gauss untuk menentukan arus i_1, i_2 dan i_3 yang mengalir pada rangkaian berikut ini



jawab:
Berdasarkan Hukum Kirchhoff:

$$\begin{aligned} I_1 + I_2 &= I_3 \\ -14 + 6I_1 - 10 - 4I_2 &= 0 \\ 10 - 6I_1 - 2I_3 &= 0 \end{aligned}$$

Lalu kita susun ulang ketiga persamaan di atas menjadi seperti ini:

$$\begin{aligned} I_1 + I_2 - I_3 &= 0 \\ 6I_1 - 4I_2 &= 24 \\ 6I_1 + 2I_3 &= 10 \end{aligned}$$

Kemudian dinyatakan dalam bentuk matriks:

$$\begin{bmatrix} 1 & 1 & -1 \\ 6 & -4 & 0 \\ 6 & 0 & 2 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 24 \\ 10 \end{bmatrix}$$

Selanjutkan kita susun matriks augmentasi sebagai berikut:

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 6 & -4 & 0 & 24 \\ 6 & 0 & 2 & 10 \end{bmatrix}$$

Langkah berikutnya adalah menghitung matriks triangularisasi dengan langkah-langkah sebagai berikut:

$$\begin{aligned} m = \frac{a_{21}}{a_{11}} = \frac{6}{1} &= 6 \\ a_{21} = a_{21} - m.a_{11} = 6 - (6).(1) &= 0 \\ a_{22} = a_{22} - m.a_{12} = -4 - (6).(1) &= -10 \\ a_{23} = a_{23} - m.a_{13} = 0 - (6).(-1) &= 6 \\ a_{24} = a_{24} - m.a_{14} = 24 - (6).(0) &= 24 \end{aligned}$$

$$\begin{aligned} m = \frac{a_{31}}{a_{11}} = \frac{6}{1} &= 6 \\ a_{31} = a_{31} - m.a_{11} = 6 - (6).(1) &= 0 \\ a_{32} = a_{32} - m.a_{12} = 0 - (6).(1) &= -6 \\ a_{33} = a_{33} - m.a_{13} = 2 - (6).(-1) &= 8 \\ a_{34} = a_{34} - m.a_{14} = 10 - (6).(0) &= 10 \end{aligned}$$

Sampai disini matriks augment mengalami perubahan menjadi

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 0 & -10 & 6 & 24 \\ 0 & -6 & 8 & 10 \end{bmatrix}$$

Kelanjutan langkah menuju triangularisasi adalah

$$\begin{aligned} m = \frac{a_{32}}{a_{22}} &= \frac{-6}{-10} \\ a_{31} = a_{31} - m.a_{21} = 0 - (\frac{-6}{-10}).(0) &= 0 \\ a_{32} = a_{32} - m.a_{22} = -6 - (\frac{-6}{-10}).(-10) &= 0 \\ a_{33} = a_{33} - m.a_{23} = 8 - (\frac{-6}{-10}).(6) &= 4,4 \\ a_{34} = a_{34} - m.a_{24} = 10 - (\frac{-6}{-10}).(24) &= -4,4 \end{aligned}$$

maka matriks triangularisasi berhasil didapat yaitu

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 0 & -10 & 6 & 24 \\ 0 & 0 & 4,4 & -4,4 \end{bmatrix}$$

Sekarang tinggal melakukan proses substitusi mundur

$$\begin{aligned} I_3 = \frac{a_{34}}{a_{33}} &= \frac{-4,4}{4,4} = -1 \\ I_2 = \frac{a_{24} - a_{23}.I_3}{a_{22}} &= \frac{24 - (6).(-1)}{-10} = -3 \\ I_1 = \frac{a_{14} - (a_{13}.I_3 + a_{12}.I_2)}{a_{11}} &= \frac{(0 - [(-1).(-1) + (1).(-3)]}{1} = 2 \end{aligned}$$

Jadi besar masing-masing arus pada rangkaian di atas adalah $I_1 = 2A$, $I_2 = -3A$ dan $I_3 = -1A$. Tanda minus (-) memiliki arti bahwa arah arus yang sesungguhnya berlawanan arah dengan asumsi awal yang kita gunakan. Keseluruhan tahapan perhitungan di atas cukup diselesaikan oleh *source-code* berikut ini

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 1 -1
   6 -4 0
   6 0 2];

%---- inisialisasi vektor b ----
b = [0 ; 24 ; 10];

I=elgauss(A,b)
```

Isi matrik A diturunkan dari sistem persamaan linear yang mengacu kepada Hukum Kirchhoff sebagai berikut

$$\begin{aligned} I_1 + I_2 - I_3 &= 0 \\ 6I_1 - 4I_2 &= 24 \\ 6I_1 + 2I_3 &= 10 \end{aligned}$$

yang kemudian dinyatakan dalam bentuk matrik A dan vektor b:

$$\begin{bmatrix} 1 & 1 & -1 \\ 6 & -4 & 0 \\ 6 & 0 & 2 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 24 \\ 10 \end{bmatrix}$$

3.6.2 Mencari invers matrik

Dalam kaitannya dengan invers matrik, matrik **A** disebut matrik **non-singular** jika matrik **A** memiliki matrik invers dirinya yaitu $\mathbf{A}^{-1}$. Atau dengan kata lain, matrik $\mathbf{A}^{-1}$ adalah invers dari matrik **A**. Jika matrik **A** tidak memiliki invers, maka matrik **A** disebut **singular**. Bila matrik **A** dikalikan dengan matrik $\mathbf{A}^{-1}$ maka akan menghasilkan matrik identitas **I**, yaitu suatu

matrik yang elemen-elemen diagonalnya bernilai 1.

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{3.4}$$

Misalnya diketahui,

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{bmatrix}, \qquad \mathbf{A}^{-1} = \begin{bmatrix} -\frac{2}{9} & \frac{5}{9} & -\frac{1}{9} \\ \frac{4}{9} & -\frac{1}{9} & \frac{2}{9} \\ -\frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

Bila keduanya dikalikan, maka akan menghasilkan matrik identitas,

$$\mathbf{A}\mathbf{A}^{-1} = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} -\frac{2}{9} & \frac{5}{9} & -\frac{1}{9} \\ \frac{4}{9} & -\frac{1}{9} & \frac{2}{9} \\ -\frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Lalu bagaimana cara memperoleh matrik invers, $\mathbf{A}^{-1}$? Itulah bahan diskusi kita kali ini. Baiklah.., anggap saja kita tidak tahu isi dari $\mathbf{A}^{-1}$. Tapi yang jelas matrik $\mathbf{A}^{-1}$ ukurannya mesti sama dengan matrik $\mathbf{A}$, yaitu 3x3.

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

dalam hal ini matrik $\mathbf{A}^{-1}$ adalah

$$\mathbf{A}^{-1} = \begin{bmatrix} i_{11} & i_{12} & i_{13} \\ i_{21} & i_{22} & i_{23} \\ i_{31} & i_{32} & i_{33} \end{bmatrix}$$

Elemen-elemen matrik invers, $\mathbf{A}^{-1}$ dapat diperoleh dengan menerapkan metode eliminasi gauss pada persamaan 3.5 yang telah dipecah 3 menjadi

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} i_{11} \\ i_{21} \\ i_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} i_{21} \\ i_{22} \\ i_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} i_{31} \\ i_{32} \\ i_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Ketiganya dapat diselesaikan satu persatu menggunakan source code Eliminasi Gauss. Source code untuk mendapatkan kolom pertama dari matrik invers adalah

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 2 -1
   2 1 0
   -1 1 2];

%---- inisialisasi vektor b ----
for j = 1:3
   b(j,1) = 0;
end
b(1,1) = 1;

I=elgauss(A,b)
```

Sementara, source code untuk mendapatkan kolom kedua dari matrik invers adalah

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 2 -1
   2 1 0
   -1 1 2];

%---- inisialisasi vektor b ----
for j = 1:3
   b(j,1) = 0;
end
b(2,1) = 1;

I=elgauss(A,b)
```

Dan untuk memperoleh kolom ketiga matrik invers, caranya adalah

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 2 -1
```

```
   2 1 0
   -1 1 2];

%---- inisialisasi vektor b ----
for j = 1:3
   b(j,1) = 0;
end
b(3,1) = 1;

I=elgauss(A,b)
```

Memang pada prinsipnya, dengan menjalankan tiga source-code di atas, akan diperoleh matrik invers. Namun cara seperti ini tentunya kurang efektif. Mungkinkah ketiganya digabung menjadi satu? Jelas bisa!

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 2 -1
   2 1 0
   -1 1 2];


%---- inisialisasi vektor b ----
for j = 1:3
   b(j,1) = 0;
end
b(1,1) = 1;
I=elgauss(A,b);
for k = 1:3
   AI(k,1) = I(k,1);
end

for j = 1:3
   b(j,1) = 0;
end
b(2,1) = 1;
I=elgauss(A,b);
for k = 1:3
   AI(k,2) = I(k,1);
end

for j = 1:3
   b(j,1) = 0;
end
b(3,1) = 1;
I=elgauss(A,b);
for k = 1:3
   AI(k,3) = I(k,1);
end
```

Jika kita munculkan indeks i seperti ini

```
clear all
clc
```

```
3
4  %---- inisialisasi matrik A ----
5  A = [1 2 -1
6     2 1 0
7     -1 1 2];
8
9
10 %---- inisialisasi vektor b ----
11 i = 1;
12 for j = 1:3
13    b(j,1) = 0;
14 end
15 b(i,1) = 1;
16 I=elgauss(A,b);
17 for k = 1:3
18    AI(k,i) = I(k,1);
19 end
20
21 i = 2;
22 for j = 1:3
23    b(j,1) = 0;
24 end
25 b(i,1) = 1;
26 I=elgauss(A,b);
27 for k = 1:3
28    AI(k,i) = I(k,1);
29 end
30
31 i = 3;
32 for j = 1:3
33    b(j,1) = 0;
34 end
35 b(i,1) = 1;
36 I=elgauss(A,b);
37 for k = 1:3
38    AI(k,i) = I(k,1);
39 end
```

maka source code tersebut dapat dioptimasi menjadi

```
1  clear all
2  clc
3
4  %---- inisialisasi matrik A ----
5  A = [1 2 -1
6     2 1 0
7     -1 1 2];
8
9
10 %---- menghitung matrik invers ----
11 for i = 1:3
12    for j = 1:3
13       b(j,1) = 0;
14    end
15    b(i,1) = 1;
16    I=elgauss(A,b);
17    for k = 1:3
18       AI(k,i) = I(k,1);
```

```
   end
end
```

Diperlukan sedikit lagi modifikasi agar source code tersebut dapat berlaku umum

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 2 -1
   2 1 0
   -1 1 2];

%---- menghitung matrik invers ----
dim = size(A);
n = dim(1);
for i = 1:n
   for j = 1:n
      b(j,1) = 0;
   end
   b(i,1) = 1;
   I=elgauss(A,b);
   for k = 1:n
      AI(k,i) = I(k,1);
   end
end
```

3.6.2.1 function invers matrik

Berdasarkan source code yang sudah teroptimasi di atas, kita bisa membuat function untuk menghitung matrik invers.

```
%---- menghitung matrik invers ----
function AI = Ainv(A)
dim = size(A);
n = dim(1);
for i = 1:n
   for j = 1:n
      b(j,1) = 0;
   end
   b(i,1) = 1;
   I=elgauss(A,b);
   for k = 1:n
      AI(k,i) = I(k,1);
   end
end
```

Dengan demikian, untuk mendapatkan matrik invers, cara termudahnya adalah

```
clear all
clc

%---- inisialisasi matrik A ----
A = [1 2 -1
   2 1 0
```

```
7     -1 1 2];
8
9   %---- menghitung matrik invers ----
10  AI = Ainv(A);
```

Keberadaan matrik $\mathbf{A}^{-1}$ bisa digunakan untuk menyelesaikan sistem persamaan linear (*mencari nilai x*), dengan cara sebagai berikut

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{A}^{-1}\mathbf{Ax} &= A^{-1}\mathbf{b} \\ \mathbf{Ix} &= \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \end{aligned} \tag{3.6}$$

Contoh berikut ini akan menjelaskan prosesnya secara lebih rinci. Misalnya diketahui sistem persamaan linear

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 2 \\ 2x_1 + x_2 &= 3 \\ -x_1 + x_2 + 2x_3 &= 4 \end{aligned}$$

Bila dikonversikan kedalam operasi matrik menjadi

$$\begin{bmatrix} 1 & 2 & -1 \\ 2 & 1 & 0 \\ -1 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

Berdasarkan persamaan (3.6), maka elemen-elemen vektor **x** dapat dicari dengan cara

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

$$\mathbf{x} = \begin{bmatrix} -\frac{2}{9} & \frac{5}{9} & -\frac{1}{9} \\ \frac{4}{9} & -\frac{1}{9} & \frac{2}{9} \\ -\frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} \frac{7}{9} \\ \frac{13}{9} \\ \frac{5}{3} \end{bmatrix}$$

Akhirnya diperoleh solusi $x_1 = 7/9$, $x_2 = 13/9$, dan $x_3 = 5/3$. Penyelesaian sistem persamaan linear menjadi lebih mudah bila matrik $\mathbf{A}^{-1}$ sudah diketahui. Sayangnya, untuk mendapatkan matrik $\mathbf{A}^{-1}$, diperlukan langkah-langkah, seperti yang sudah dibahas pada contoh pertama di atas, yang berakibat in-efisiensi proses penyelesaian (secara komputasi) bila dibandingkan dengan metode eliminasi gauss untuk memecahkan sistem persamaan linear. Namun bagaimanapun, secara konseptual kita dianjurkan mengetahui cara bagaimana mendapatkan matrik $\mathbf{A}^{-1}$.

3.7 Penutup

Saya cukupkan sementara sampai disini. Insya Allah akan saya sambung lagi dilain waktu. Kalau ada yang mau didiskusikan, silakan hubungi saya melalui email yang tercantum di halaman paling depan.

Bab 4

Aplikasi Eliminasi Gauss pada Masalah Inversi

Objektif :

- ▷ Mengenalkan model garis.
- ▷ Mengenalkan model parabola.
- ▷ Mengenalkan model bidang.

Pada bab ini, saya mencoba menuliskan aplikasi Metode Eliminasi Gauss sebagai dasar-dasar teknik inversi yaitu meliputi model garis, model parabola dan model bidang. Uraian aplikasi tersebut diawali dari ketersediaan data observasi, lalu sejumlah parameter model mesti dicari dengan teknik inversi. Mari kita mulai dari model garis.

4.1 Inversi Model Garis

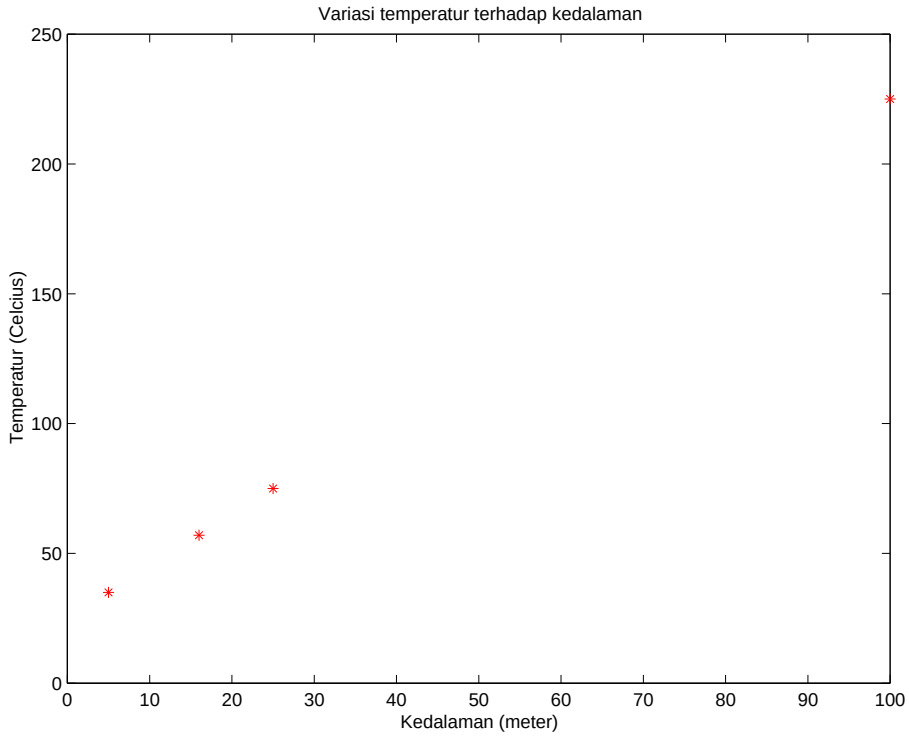
Pengukuran suhu terhadap kedalaman di bawah permukaan bumi menunjukkan bahwa semakin dalam, suhu semakin tinggi. Misalnya telah dilakukan sebanyak empat kali (N = 4) pengukuran suhu (T_i) pada kedalaman yang berbeda beda (z_i). Tabel pengukuran secara sederhana disajikan seperti ini:

Tabel 4.1: Data suhu bawah permukaan tanah terhadap kedalaman

Pengukuran ke-i	Kedalaman (m)	suhu (^OC)
1	$z_1 = 5$	$T_1 = 35$
2	$z_2 = 16$	$T_2 = 57$
3	$z_3 = 25$	$T_3 = 75$
4	$z_4 = 100$	$T_4 = 225$

Grafik sebaran data observasi ditampilkan pada Gambar (4.2). Lalu kita berasumsi bahwa variasi suhu terhadap kedalaman ditentukan oleh rumus berikut ini:

$$m_1 + m_2 z_i = T_i \tag{4.1}$$



Gambar 4.1: Sebaran data observasi antara suhu dan kedalaman

dimana m_1 dan m_2 adalah konstanta-konstanta yang akan dicari. Rumus di atas disebut **model matematika**. Sedangkan m_1 dan m_2 disebut **parameter model**. Pada model matematika di atas terdapat dua buah parameter model, $(M = 2)$. Sementara jumlah data observasi ada empat, $(N = 4)$, yaitu nilai-nilai kedalaman, z_i, dan suhu, T_i. Berdasarkan model tersebut, kita bisa menyatakan suhu dan kedalaman masing-masing sebagai berikut:

$$\begin{aligned} m_1 + m_2 z_1 &= T_1 \\ m_1 + m_2 z_2 &= T_2 \\ m_1 + m_2 z_3 &= T_3 \\ m_1 + m_2 z_4 &= T_4 \end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \\ 1 & z_4 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{4.2}$$

Lalu ditulis secara singkat

$$G\mathbf{m} = \mathbf{d} \tag{4.3}$$

dimana **d** adalah data yang dinyatakan dalam vektor kolom, **m** adalah model parameter, juga dinyatakan dalam vektor kolom, dan G disebut **matrik kernel**. Lantas bagaimana cara menda-

patkan nilai m_1 dan m_2 pada vektor kolom **m**? Manipulasi berikut ini bisa menjawabnya

$$G^T G\mathbf{m} = G^T\mathbf{d} \tag{4.4}$$

dimana t disini maksudnya adalah tanda transpos matrik. Selanjutnya, untuk mendapatkan elemen-elemen **m**, diperlukan langkah-langkah perhitungan berikut ini:

1. Tentukan transpos dari matrik kernel, yaitu G^t

$$G = \begin{bmatrix} 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \\ 1 & z_4 \end{bmatrix} \quad \Rightarrow \quad G^t = \begin{bmatrix} 1 & 1 & 1 & 1 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}$$

2. Tentukan $G^T G$

$$G^t G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \\ 1 & z_4 \end{bmatrix} = \begin{bmatrix} N & \sum z_i \\ \sum z_i & \sum z_i^2 \end{bmatrix}$$

   dimana $N = 4$ dan $i = 1, 2, 3, 4$.

3. Kemudian tentukan pula $G^T\mathbf{d}$

$$G^t\mathbf{d} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \end{bmatrix}$$

4. Sekarang persamaan (4.4) dapat dinyatakan sebagai

$$\begin{bmatrix} N & \sum z_i \\ \sum z_i & \sum z_i^2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \end{bmatrix} \tag{4.5}$$

5. Aplikasikan metode **Eliminasi Gauss dengan Substitusi Mundur**. Untuk itu, tentukan matrik augment-nya

$$\left[\begin{array}{cc|c} N & \sum z_i & \sum T_i \\ \sum z_i & \sum z_i^2 & \sum z_i T_i \end{array}\right]$$

6. Untuk mempermudah perhitungan, kita masukan dulu angka-angka yang tertera pada

tabel pengukuran dihalaman depan.

$$\left[\begin{array}{ccc} 4 & 146 & 392 \\ 146 & 10906 & 25462 \end{array}\right]$$

7. Lakukan proses triangularisasi dengan operasi $(P_2 - (36,5)P_1) \rightarrow P_2$. Saya sertakan pula indeks masing-masing elemen pada matrik augment sebagaimana yang telah saya lakukan pada catatan kuliah yang berjudul **Metode Eliminasi Gauss**. Hasilnya adalah

$$\left[\begin{array}{ccc} 4 & 146 & 392 \\ 0 & 5577 & 11154 \end{array}\right] = \left[\begin{array}{ccc} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{array}\right]$$

8. Terakhir, tentukan konstanta m_1 dan m_2 yang merupakan elemen-elemen vektor kolom **m**, dengan proses substitusi mundur. Pertama tentukan m_2

$$m_2 = \frac{a_{23}}{a_{22}} = \frac{11154}{5577} = 2$$

lalu tentukan m_1

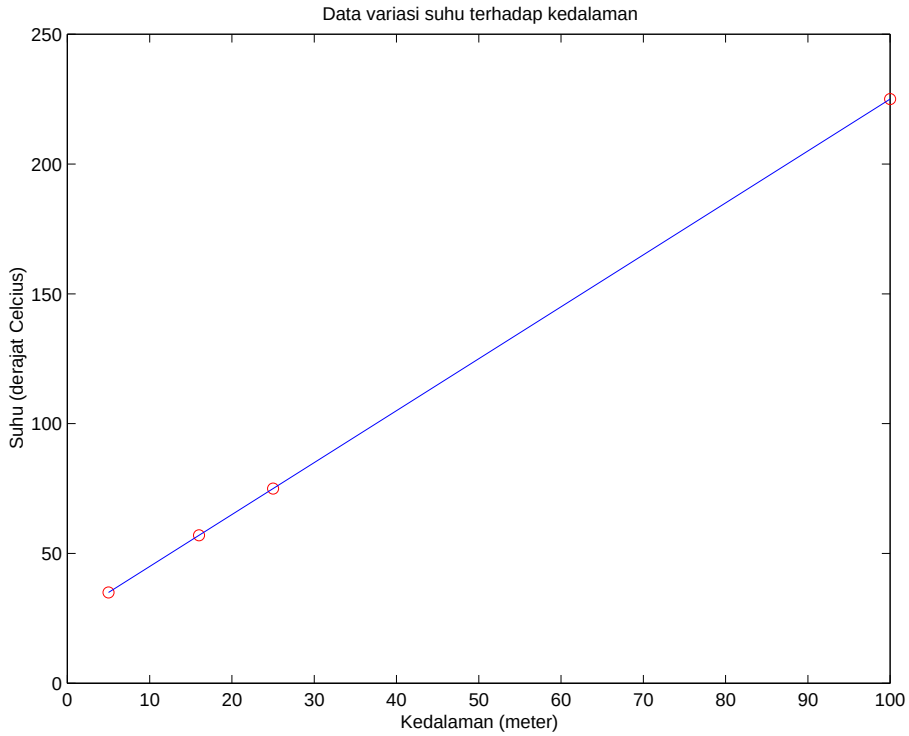
$$m_1 = \frac{a_{13} - a_{12}m_2}{a_{11}} = \frac{392 - (146)(2)}{4} = 25$$

4.1.1 Script matlab inversi model garis

Script inversi model garis ini dibangun dari beberapa script yang sudah kita pelajari sebelumnya, yaitu script transpose matriks, perkalian matrik dan script eliminasi gauss. Silakan pelajari maksud tiap-tiap baris pada script ini.

```
clc
clear all
close all

% ---- data observasi ----
N = 4;      % jumlah data
z = [ 5 ; 16 ; 25 ; 100 ];
T = [ 35 ; 57 ; 75 ; 225 ];


% ---- menentukan matrik kernel, G ----
for i = 1:N
   G(i,1) = 1;
   G(i,2) = z(i,1);
end

% ---- menentukan vektor d ----
d=T;

% ---- proses inversi ----
A = G'*G;
b = G'*d;
m = elgauss(A,b);

```

```
%-------MENGGAMBAR GRAFIK----------------------
plot(z,T,'ro');
xlabel('Kedalaman (meter)');ylabel('Suhu (derajat Celcius)');
title('Data variasi suhu terhadap kedalaman')
hold on;
for i=1:max(z)
   zi(i)=i;
   Ti(i)=m(1)+m(2)*zi(i);
end
plot(zi,Ti);
hold off;
```



Gambar 4.2: Kurva hasil inversi data observasi antara suhu dan kedalaman

Demikianlah contoh aplikasi metode Eliminasi Gauss pada model garis. Anda bisa mengaplikasikan pada kasus lain, dengan syarat kasus yang anda tangani memiliki bentuk model yang sama dengan yang telah dikerjakan pada catatan ini, yaitu **model persamaan garis** atau disingkat **model garis**: $y = m1 + m2x$. Selanjutnya mari kita pelajari inversi model parabola.

4.2 Inversi Model Parabola

Pengukuran suhu terhadap kedalaman di bawah permukaan bumi menunjukkan bahwa semakin dalam, suhu semakin tinggi. Misalnya telah dilakukan sebanyak delapan kali $(N = 8)$ pengukuran suhu (T_i) pada kedalaman yang berbeda beda (z_i). Tabel 4.2 menyajikan data observasi pada kasus ini.

Lalu kita berasumsi bahwa variasi suhu terhadap kedalaman ditentukan oleh rumus berikut ini:

$$m_1 + m_2 z_i + m_3 z_i^2 = T_i \tag{4.6}$$

Tabel 4.2: Data suhu bawah permukaan tanah terhadap kedalaman

Pengukuran ke-i	Kedalaman (m)	suhu (^OC)
1	$z_1 = 5$	$T_1 = 21,75$
2	$z_2 = 8$	$T_2 = 22,68$
3	$z_3 = 14$	$T_3 = 25,62$
4	$z_4 = 21$	$T_4 = 30,87$
5	$z_5 = 30$	$T_5 = 40,5$
6	$z_6 = 36$	$T_6 = 48,72$
7	$z_7 = 45$	$T_7 = 63,75$
8	$z_8 = 60$	$T_8 = 96$

dimana m_1, m_2 dan m_3 adalah konstanta-konstanta yang akan dicari. Rumus di atas disebut **model**. Sedangkan m_1, m_2 dan m_3 disebut **model parameter**. Jadi pada model di atas terdapat tiga buah model parameter, $(M = 3)$. Adapun yang berlaku sebagai **data** adalah nilai-nilai suhu T_1, T_2,..., dan T_8. Berdasarkan model tersebut, kita bisa menyatakan suhu dan kedalaman masing-masing sebagai berikut:

$$\begin{aligned}
m_1 + m_2 z_1 + m_3 z_1^2 &= T_1 \\
m_1 + m_2 z_2 + m_3 z_2^2 &= T_2 \\
m_1 + m_2 z_3 + m_3 z_3^2 &= T_3 \\
m_1 + m_2 z_4 + m_3 z_4^2 &= T_4 \\
m_1 + m_2 z_5 + m_3 z_5^2 &= T_5 \\
m_1 + m_2 z_6 + m_3 z_6^2 &= T_6 \\
m_1 + m_2 z_7 + m_3 z_7^2 &= T_7 \\
m_1 + m_2 z_8 + m_3 z_8^2 &= T_8
\end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} 1 & z_1 & z_1^2 \\ 1 & z_2 & z_2^2 \\ 1 & z_3 & z_3^2 \\ 1 & z_4 & z_4^2 \\ 1 & z_5 & z_5^2 \\ 1 & z_6 & z_6^2 \\ 1 & z_7 & z_7^2 \\ 1 & z_8 & z_8^2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \end{bmatrix} \tag{4.7}$$

Lalu ditulis secara singkat

$$G\mathbf{m} = \mathbf{d} \tag{4.8}$$

dimana **d** adalah data yang dinyatakan dalam vektor kolom, **m** adalah model parameter, juga dinyatakan dalam vektor kolom, dan G disebut **matrik kernel**. Lantas bagaimana cara menda-

patkan nilai m_1, m_2 dan m_3 pada vektor kolom **m**? Manipulasi berikut ini bisa menjawabnya

$$G^t G\mathbf{m} = G^t\mathbf{d} \tag{4.9}$$

dimana t disini maksudnya adalah tanda transpos matrik. Selanjutnya, untuk mendapatkan elemen-elemen **m**, diperlukan langkah-langkah perhitungan berikut ini:

1. Tentukan transpos dari matrik kernel, yaitu G^t

$$G = \begin{bmatrix} 1 & z_1 & z_1^2 \\ 1 & z_2 & z_2^2 \\ 1 & z_3 & z_3^2 \\ 1 & z_4 & z_4^2 \\ 1 & z_5 & z_5^2 \\ 1 & z_6 & z_6^2 \\ 1 & z_7 & z_7^2 \\ 1 & z_8 & z_8^2 \end{bmatrix} \quad \Rightarrow \quad G^t = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 & z_8 \\ z_1^2 & z_2^2 & z_3^2 & z_4^2 & z_5^2 & z_6^2 & z_7^2 & z_8^2 \end{bmatrix}$$

2. Tentukan $G^t G$

$$G^t G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 & z_8 \\ z_1^2 & z_2^2 & z_3^2 & z_4^2 & z_5^2 & z_6^2 & z_7^2 & z_8^2 \end{bmatrix} \begin{bmatrix} 1 & z_1 & z_1^2 \\ 1 & z_2 & z_2^2 \\ 1 & z_3 & z_3^2 \\ 1 & z_4 & z_4^2 \\ 1 & z_5 & z_5^2 \\ 1 & z_6 & z_6^2 \\ 1 & z_7 & z_7^2 \\ 1 & z_8 & z_8^2 \end{bmatrix} = \begin{bmatrix} N & \sum z_i & \sum z_i^2 \\ \sum z_i & \sum z_i^2 & \sum z_i^3 \\ \sum z_i^2 & \sum z_i^3 & \sum z_i^4 \end{bmatrix}$$

dimana $N = 8$ dan $i = 1, 2, 3, ..., 8$.

3. Kemudian tentukan pula $G^t\mathbf{d}$

$$G^t\mathbf{d} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 & z_8 \\ z_1^2 & z_2^2 & z_3^2 & z_4^2 & z_5^2 & z_6^2 & z_7^2 & z_8^2 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \\ \sum z_i^2 T_i \end{bmatrix}$$

4. Sekarang persamaan (4.14) dapat dinyatakan sebagai (ini khan **least square** juga...!?)

$$\begin{bmatrix} N & \sum z_i & \sum z_i^2 \\ \sum z_i & \sum z_i^2 & \sum z_i^3 \\ \sum z_i^2 & \sum z_i^3 & \sum z_i^4 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \\ \sum z_i^2 T_i \end{bmatrix} \tag{4.10}$$

5. Aplikasikan metode **Eliminasi Gauss dengan Substitusi Mundur**. Untuk itu, tentukan matrik augment-nya

$$\left[\begin{array}{ccc|c} N & \sum z_i & \sum z_i^2 & \sum T_i \\ \sum z_i & \sum z_i^2 & \sum z_i^3 & \sum z_i T_i \\ \sum z_i^2 & \sum z_i^3 & \sum z_i^4 & \sum z_i^2 T_i \end{array}\right]$$

6. Untuk mempermudah perhitungan, kita masukan dulu angka-angka yang tertera pada tabel pengukuran dihalaman depan.

$$\left[\begin{array}{ccc|c} 8 & 219 & 8547 & 349,89 \\ 219 & 8547 & 393423 & 12894,81 \\ 8547 & 393423 & 19787859 & 594915,33 \end{array}\right]$$

7. Lakukan proses triangularisasi dengan operasi $(P_2 - (219/8)P_1) \rightarrow P_2$. Hasilnya adalah

$$\left[\begin{array}{ccc|c} 8 & 219 & 8547 & 349,89 \\ 0 & 2551,88 & 159448,88 & 3316,57 \\ 8547 & 393423 & 19787859 & 594915,33 \end{array}\right]$$

8. Masih dalam proses triangularisai, operasi berikutnya $(P_3 - (8547/8)P_1) \rightarrow P_3$. Hasilnya adalah

$$\left[\begin{array}{ccc|c} 8 & 219 & 8547 & 349,89 \\ 0 & 2551,88 & 159448,88 & 3316,57 \\ 0 & 159448.88 & 10656457,88 & 221101,6 \end{array}\right]$$

9. Masih dalam proses triangularisai, operasi berikutnya $(P_3 - (159448,88/2551,88)P_2) \rightarrow P_3$. Hasilnya adalah

$$\left[\begin{array}{ccc|c} 8 & 219 & 8547 & 349,89 \\ 0 & 2551,88 & 159448,88 & 3316,57 \\ 0 & 0 & 693609,48 & 13872,19 \end{array}\right] \tag{4.11}$$

Seperti catatan yang lalu, saya ingin menyertakan pula notasi masing-masing elemen

pada matrik augment sebelum melakukan proses substitusi mundur.

$$\left[\begin{array}{ccc|c} 8 & 219 & 8547 & 349,89 \\ 0 & 2551,88 & 159448,88 & 3316,57 \\ 0 & 0 & 693609,48 & 13872,19 \end{array}\right] \Leftrightarrow \left[\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{array}\right]$$

10. Terakhir, tentukan konstanta m_1, m_2 dan m_3 yang merupakan elemen-elemen vektor kolom **m**, dengan proses substitusi mundur. Pertama tentukan m_3

$$m_3 = \frac{a_{34}}{a_{33}} = \frac{13872,19}{693609,48} = 0,02$$

lalu m_2

$$m_2 = \frac{a_{24} - a_{23}m_3}{a_{22}} = \frac{3316,57 - (159448,88)(0,02)}{2551,88} = 0,05$$

dan m_1

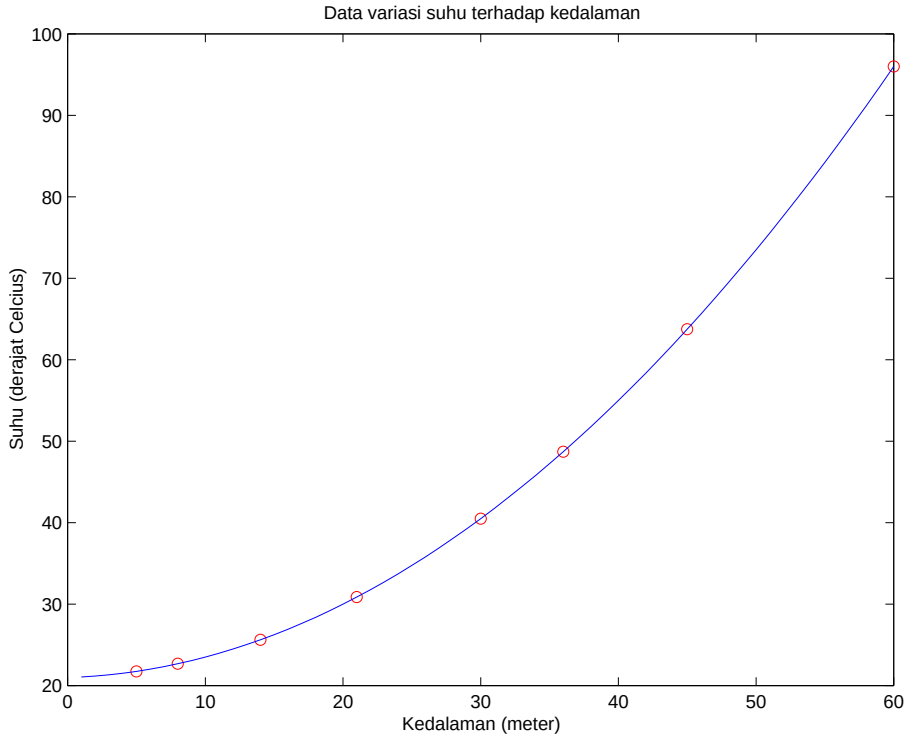
$$m_1 = \frac{a_{14} - (a_{12}m_2 + a_{13}m_3)}{a_{11}} = \frac{349,89 - [(219)(0,05) + (8547)(0,02)}{8} = 21$$

4.2.1 Script matlab inversi model parabola

Perbedaan utama script ini dengan script inversi model garis terletak pada inisialisasi elemen-elemen matrik kernel. Elemen-elemen matrik kernel sangat ditentukan oleh model matematika yang digunakan. Seperti pada script ini, matrik kernelnya diperoleh dari model parabola.

```
clc
clear all
close all

% ---- data observasi ----
N = 8;    % Jumlah data
z = [5; 8; 14; 21; 30; 36; 45; 60];
T = [21.75; 22.68; 25.62; 30.87; 40.5; 48.72; 63.75; 96];

% ---- menentukan matrik kernel, G ----
for i = 1:N
   G(i,1) = 1;
   G(i,2) = z(i,1);
   G(i,3) = z(i,1)^2;
end

% ---- menentukan vektor d ----
d=T;

% ---- proses inversi ----
A = G'*G;
b = G'*d;
m = elgauss(A,b);

%-------MENGGAMBAR GRAFIK----------------------
plot(z,T,'ro');
xlabel('Kedalaman (meter)');ylabel('Suhu (derajat Celcius)');
```

```
28 title('Data variasi suhu terhadap kedalaman');
29 hold on;
30 for i=1:max(z)
31    zi(i)=i;
32    Ti(i)=m(1)+m(2)*zi(i)+m(3)*zi(i)^2;
33 end
34 plot(zi,Ti);
35 hold off;
```



Gambar 4.3: Kurva hasil inversi data observasi antara suhu dan kedalaman

Demikianlah contoh aplikasi metode Eliminasi Gauss pada model parabola. Anda bisa mengaplikasikan pada kasus lain, dengan syarat kasus yang anda tangani memiliki bentuk **model** yang sama dengan yang telah dikerjakan pada catatan ini, yaitu memiliki tiga buah model parameter yang tidak diketahui dalam bentuk persamaan parabola: $y = m_1 + m_2 x + m_3 x^2$. Pada catatan berikutnya, saya akan membahas model yang mengandung tiga model parameter dalam 2 dimensi.

4.3 Inversi Model Bidang

Dalam catatan ini saya belum sempat mencari contoh pengukuran yang sesuai untuk model 2-dimensi. Maka, saya ingin langsung saja mengajukan sebuah model untuk 2-dimensi berikut ini:

$$m_1 + m_2 x_i + m_3 y_i = d_i \tag{4.12}$$

dimana m_1, m_2 dan m_3 merupakan model parameter yang akan dicari. Adapun yang berlaku sebagai **data** adalah $d_1, d_2, d_3, ..., d_i$. Berdasarkan model tersebut, kita bisa menyatakan suhu

dan kedalaman masing-masing sebagai berikut:

$$\begin{aligned} m_1 + m_2 x_1 + m_3 y_1 &= d_1 \\ m_1 + m_2 x_2 + m_3 y_2 &= d_2 \\ m_1 + m_2 x_3 + m_3 y_3 &= d_3 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots & \\ m_1 + m_2 x_N + m_3 y_N &= d_N \end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ \vdots & \vdots & \vdots \\ 1 & x_N & y_N \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix}$$

Lalu ditulis secara singkat

$$G\mathbf{m} = \mathbf{d} \tag{4.13}$$

dimana **d** adalah data yang dinyatakan dalam vektor kolom, **m** adalah model parameter, juga dinyatakan dalam vektor kolom, dan G disebut **matrik kernel**. Lantas bagaimana cara mendapatkan nilai m_1, m_2 dan m_3 pada vektor kolom **m**? Manipulasi berikut ini bisa menjawabnya

$$G^t G\mathbf{m} = G^t\mathbf{d} \tag{4.14}$$

dimana t disini maksudnya adalah tanda transpos matrik. Selanjutnya, untuk mendapatkan elemen-elemen **m**, diperlukan langkah-langkah perhitungan berikut ini:

1. Tentukan transpos dari matrik kernel, yaitu G^t

$$G = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ \vdots & \vdots & \vdots \\ 1 & x_N & y_N \end{bmatrix} \quad \Rightarrow \quad G^t = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_1 & x_2 & x_3 & \cdots & x_N \\ y_1 & y_2 & y_3 & \cdots & y_N \end{bmatrix}$$

2. Tentukan $G^t G$

$$G^t G = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_1 & x_2 & x_3 & \cdots & x_N \\ y_1 & y_2 & y_3 & \cdots & y_N \end{bmatrix} \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ \vdots & \vdots & \vdots \\ 1 & x_N & y_N \end{bmatrix} = \begin{bmatrix} N & \sum x_i & \sum y_i \\ \sum x_i & \sum x_i^2 & \sum x_i y_i \\ \sum y_i & \sum x_i y_i & \sum y_i^2 \end{bmatrix}$$

dimana N = jumlah data. dan $i = 1, 2, 3, ..., N$.

3. Kemudian tentukan pula $G^t\mathbf{d}$

$$G^t\mathbf{d} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_1 & x_2 & x_3 & \cdots & x_N \\ y_1 & y_2 & y_3 & \cdots & y_N \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix} = \begin{bmatrix} \sum d_i \\ \sum x_i d_i \\ \sum y_i d_i \end{bmatrix}$$

4. Sekarang, persamaan (4.14) dapat dinyatakan sebagai

$$\begin{bmatrix} N & \sum x_i & \sum y_i \\ \sum x_i & \sum x_i^2 & \sum x_i y_i \\ \sum y_i & \sum x_i y_i & \sum y_i^2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} \sum d_i \\ \sum x_i d_i \\ \sum y_i d_i \end{bmatrix} \tag{4.15}$$

5. Aplikasikan metode **Eliminasi Gauss dengan Substitusi Mundur**. Untuk itu, tentukan matrik augment-nya

$$\left[\begin{array}{ccc|c} N & \sum x_i & \sum y_i & \sum d_i \\ \sum x_i & \sum x_i^2 & \sum x_i y_i & \sum x_i d_i \\ \sum y_i & \sum x_i y_i & \sum y_i^2 & \sum y_i d_i \end{array}\right]$$

6. Langkah-langkah selanjutnya akan sama persis dengan catatan sebelumnya (**model linear** dan **model parabola**)

Anda bisa mengaplikasikan data pengukuran yang anda miliki, dengan syarat kasus yang anda tangani memiliki bentuk **model** yang sama dengan yang telah dikerjakan pada catatan ini, yaitu memiliki tiga buah model parameter yang tidak diketahui dalam bentuk persamaan bidang (atau 2-dimensi): $d = m_1 + m_2 x + m_3 y$.

Saya cukupkan sementara sampai disini. Insya Allah akan saya sambung lagi dilain waktu. Kalau ada yang mau didiskusikan, silakan hubungi saya melalui email: $supri@fisika.ui.ac.id$.

4.4 Contoh aplikasi

4.4.1 Menghitung gravitasi di planet X

Seorang astronot tiba di suatu planet yang tidak dikenal. Setibanya disana, ia segera mengeluarkan kamera otomatis, lalu melakukan ekperimen kinematika yaitu dengan melempar batu vertikal ke atas. Hasil foto-foto yang terekam dalam kamera otomatis adalah sebagai berikut Plot data pengukuran waktu vs ketinggian diperlihatkan pada Gambar 4.4. Anda diminta untuk membantu proses pengolahan data sehingga diperoleh nilai konstanta gravitasi di planet tersebut dan kecepatan awal batu. Jelas, ini adalah persoalan inversi, yaitu mencari *unkown parameter* (konstanta gravitasi dan kecepatan awal batu) dari data observasi (hasil foto gerak

Tabel 4.3: Data ketinggian terhadap waktu dari planet X

Waktu (dt)	Ketinggian (m)	Waktu (dt)	Ketinggian (m)
0,00	5,00	2,75	7,62
0,25	5,75	3,00	7,25
0,50	6,40	3,25	6,77
0,75	6,94	3,50	6,20
1,00	7,38	3,75	5,52
1,25	7,72	4,00	4,73
1,50	7,96	4,25	3,85
1,75	8,10	4,50	2,86
2,00	8,13	4,75	1,77
2,25	8,07	5,00	0,58
2,50	7,90		

sebuah batu).

Langkah awal untuk memecahkan persoalan ini adalah dengan mengajukan asumsi model matematika, yang digali dari konsep-konsep fisika, yang kira-kira paling cocok dengan situasi pengambilan data observasi. Salah satu konsep dari fisika yang bisa diajukan adalah konsep tentang Gerak-Lurus-Berubah-Beraturan (GLBB), yang formulasinya seperti ini

$$h_o + v_o t - \frac{1}{2} g t^2 = h$$

Berdasarkan tabel data observasi, ketinggian pada saat $t = 0$ adalah 5 m. Itu artinya $h_o = 5$ m. Sehingga model matematika (formulasi GLBB) dapat dimodifikasi sedikit menjadi

$$v_o t - \frac{1}{2} g t^2 = h - h_o \tag{4.16}$$

Selanjut, didefinisikan m_1 dan m_2 sebagai berikut

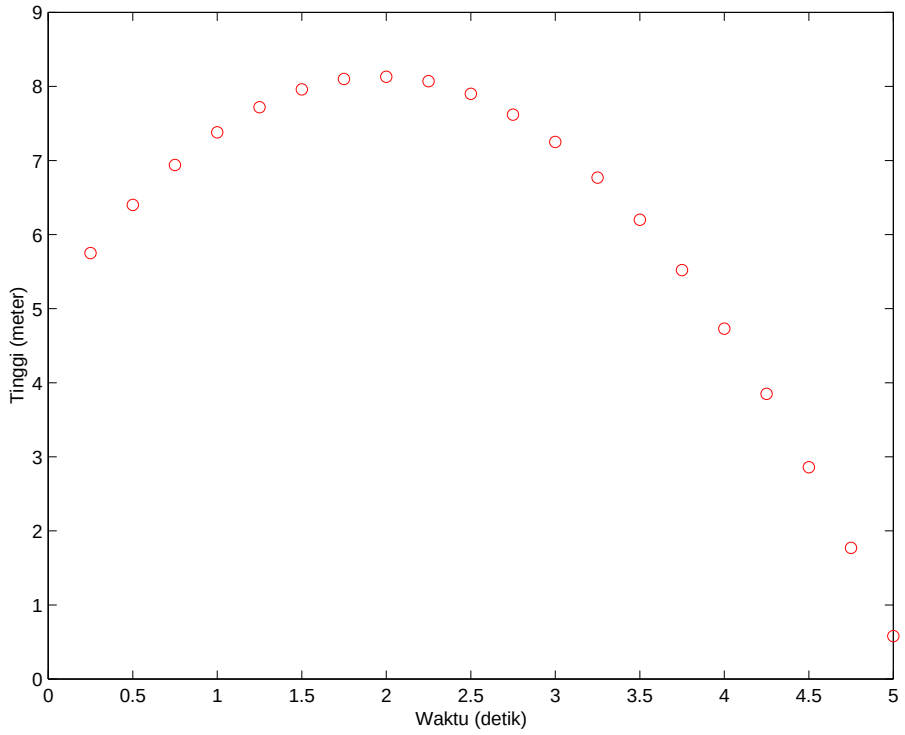
$$m_1 = v_o \qquad m_2 = -\frac{1}{2} g \tag{4.17}$$

sehingga persamaan model GLBB menjadi

$$m_1 t_i + m_2 t_i^2 = h_i - 5 \tag{4.18}$$

dimana i menunjukkan data ke-i.

Langkah berikutnya adalah menentukan nilai tiap-tiap elemen matrik kernel, yaitu dengan



Gambar 4.4: Grafik data pengukuran gerak batu

memasukan data observasi kedalam model matematika (persamaan (4.18))

$$\begin{aligned} m_1 t_1 + m_2 t_1^2 &= h_1 - 5 \\ m_1 t_2 + m_2 t_2^2 &= h_2 - 5 \\ m_1 t_3 + m_2 t_3^2 &= h_3 - 5 \\ \vdots \quad \vdots \quad &= \vdots \\ m_1 t_{20} + m_2 t_{20}^2 &= h_{20} - 5 \end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} t_1 & t_1^2 \\ t_2 & t_2^2 \\ t_3 & t_3^2 \\ t_4 & t_4^2 \\ \vdots & \vdots \\ t_{19} & t_{19}^2 \\ t_{20} & t_{20}^2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} h_1 - 5 \\ h_2 - 5 \\ h_3 - 5 \\ \vdots \\ h_{19} - 5 \\ h_{20} - 5 \end{bmatrix}$$

Operasi matrik di atas memenuhi persamaan matrik

$$G\mathbf{m} = \mathbf{d}$$

Seperti yang sudah dipelajari pada bab ini, penyelesaian masalah inversi dimulai dari proses manipulasi persamaan matrik sehingga perkalian antara G^t dan G menghasilkan matriks bujursangkar

$$G^t G\mathbf{m} = G^t\mathbf{d} \tag{4.19}$$

Selanjutnya, untuk mendapatkan m_1 dan m_2, prosedur inversi dilakukan satu-per-satu

1. Menentukan transpos matrik kernel, yaitu G^t

$$G = \begin{bmatrix} t_1 & t_1^2 \\ t_2 & t_2^2 \\ t_3 & t_3^2 \\ t_4 & t_4^2 \\ \vdots & \vdots \\ t_{19} & t_{19}^2 \\ t_{20} & t_{20}^2 \end{bmatrix} \quad \Rightarrow \quad G^t = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & \dots & t_{19} & t_{20} \\ t_1^2 & t_2^2 & t_3^2 & t_4^2 & \dots & t_{19}^2 & t_{20}^2 \end{bmatrix}$$

2. Menentukan $G^t G$

$$G^t G = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & \dots & t_{19} & t_{20} \\ t_1^2 & t_2^2 & t_3^2 & t_4^2 & \dots & t_{19}^2 & t_{20}^2 \end{bmatrix} \begin{bmatrix} t_1 & t_1^2 \\ t_2 & t_2^2 \\ t_3 & t_3^2 \\ t_4 & t_4^2 \\ \vdots & \vdots \\ t_{19} & t_{19}^2 \\ t_{20} & t_{20}^2 \end{bmatrix} = \begin{bmatrix} \sum t_i^2 & \sum t_i^3 \\ \sum t_i^3 & \sum t_i^4 \end{bmatrix}$$

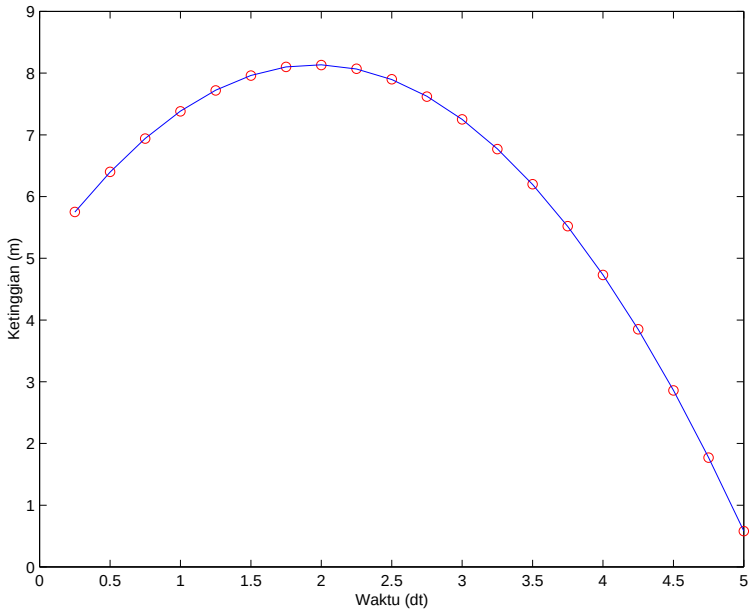
   dimana $N = 20$ dan $i = 1, 2, ..., N$.

3. Kemudian menentukan hasil perkalian $G^t\mathbf{d}$

$$G^t\mathbf{d} = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & \dots & t_{19} & t_{20} \\ t_1^2 & t_2^2 & t_3^2 & t_4^2 & \dots & t_{19}^2 & t_{20}^2 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ \vdots \\ h_{19} \\ h_{20} \end{bmatrix} = \begin{bmatrix} \sum t_i h_i \\ \sum t_i^2 h_i \end{bmatrix}$$

4. Sekarang persamaan (4.19) dapat dinyatakan sebagai

$$\begin{bmatrix} \sum t_i^2 & \sum t_i^3 \\ \sum t_i^3 & \sum t_i^4 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} \sum t_i h_i \\ \sum t_i^2 h_i \end{bmatrix} \tag{4.20}$$



Gambar 4.5: Grafik hasil inversi parabola

Berdasarkan data observasi, diperoleh

$$\begin{bmatrix} 179,4 & 689,1 \\ 689,1 & 2822,9 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} 273,7 \\ 796,3 \end{bmatrix}$$

Hasil inversinya adalah nilai kecepatan awal yaitu saat batu dilempar ke atas adalah sebesar $m_1 = v_o$ = 3,2009 m/dt. Adapun percepatan gravitasi diperoleh dari m_2 dimana $m_2 = -\frac{1}{2}g$ = -0,8169; maka disimpulkan nilai g adalah sebesar 1,6338 m/dt^2.

Gambar 4.5 memperlihatkan kurva hasil inversi berserta sebaran titik data observasi. Garis berwarna biru merupakan garis kurva *fitting* hasil inversi parabola. Sedangkan bulatan berwarna merah adalah data pengukuran ketinggian (m) terhadap waktu (dt). Jelas terlihat bahwa garis kurva berwarna biru benar-benar cocok melewati semua titik data pengukuran. Ini menunjukkan tingkat akurasi yang sangat tinggi. Sehingga nilai kecepatan awal dan gravitasi hasil inversi cukup valid untuk menjelaskan gerak batu di planet X.

Berikut adalah script inversi dalam Matlab untuk memecahkan masalah ini

```
clc
clear all
close all

% ---- data observasi ----
N = 20;      % jumlah data
for i=1:N
   t(i)=i*0.25;
end
h = [5.75;6.40;6.94;7.38;7.72;7.96;8.10;8.13;8.07;
   7.90;7.62;7.25;6.77;6.20;5.52;4.73;3.85;2.86;1.77;0.58];
```

```
12
13 % ---- menentukan matrik kernel, G ----
14 for i=1:N
15    G(i,1)=t(i);
16    G(i,2)=t(i)^2;
17 end
18
19 % ---- menentukan vektor d ----
20 for i=1:N
21    d(i,1)=h(i)-5;
22 end
23
24 % ---- proses inversi ----
25 A = G'*G;
26 b = G'*d;
27 m = elgauss(A,b);
28
29 %-------MENGGAMBAR GRAFIK----------------------
30 plot(t,h,'ro');
31 xlabel('Waktu (detik)');ylabel('ketinggian (meter)');
32 title('Data variasi waktu terhadap ketinggian')
33 hold on;
34 for i=1:N
35    hi(i)=m(1)*t(i)+m(2)*t(i)^2+5;
36 end
37 plot(t,hi);
38 hold off;
```

Bab 5

Metode *LU Decomposition*

✍ **Objektif :**

▷ Mengenalkan teknik faktorisasi matrik.

▷ Mengenalkan aplikasi LU Decomposition pada sistem persamaan linear.

▷ Merumuskan algoritma LU Decomposition.

5.1 Faktorisasi matrik

Pada semua catatan yang terdahulu, telah diulas secara panjang lebar bahwa sistem persamaan linear dapat dicari solusinya secara langsung dengan metode eliminasi gauss. Namun perlu juga diketahui bahwa eliminasi gauss bukan satu-satunya metode dalam mencari solusi sistem persamaan linear, misalnya ada metode matrik inversi seperti yang dijelaskan pada catatan yang paling terakhir. Terlepas dari masalah in-efisiensi penyelesaiannya, yang jelas metode invers matrik bisa digunakan untuk menyelesaikan sistem persamaan linear.

Nah, pada catatan kali ini, saya ingin mengetengahkan sebuah metode yang lain untuk menyelesaikan sistem persamaan linear, yaitu **metode faktorisasi matrik** yang umum dikenal sebagai *LU-decomposition*. Metode ini sekaligus menjadi pengantar menuju metode *Singular Value Decomposition, (SVD)*, suatu metode yang saat ini paling "handal" dalam menyelesaikan sistem persamaan linear dan merupakan bagian dari metode *least square*.

Seperti biasa, kita berasumsi bahwa sistem persamaan linear dapat dinyatakan dalam operasi matrik

$$A\mathbf{x} = \mathbf{b} \tag{5.1}$$

Pada metode *LU-decomposition*, matrik A difaktorkan menjadi matrik L dan matrik U, dimana dimensi atau ukuran matrik L dan U harus sama dengan dimensi matrik A. Atau dengan kata lain, hasil perkalian matrik L dan matrik U adalah matrik A,

$$A = LU \tag{5.2}$$

sehingga persamaan (7.4) menjadi

$$LU\mathbf{x} = \mathbf{b}$$

Langkah penyelesaian sistem persamaan linear dengan metode *LU-decomposition*, diawali dengan menghadirkan vektor $\mathbf{y}$ dimana,

$$U\mathbf{x} = \mathbf{y} \tag{5.3}$$

Langkah tersebut tidak bermaksud untuk menghitung vektor $\mathbf{y}$, melainkan untuk menghitung vektor $\mathbf{x}$. Artinya, sebelum persamaan (5.3) dieksekusi, nilai-nilai yang menempati elemen-elemen vektor $\mathbf{y}$ harus sudah diketahui. Lalu bagaimana cara memperoleh vektor $\mathbf{y}$? Begini caranya,

$$L\mathbf{y} = \mathbf{b} \tag{5.4}$$

Kesimpulannya, metode *LU-decomposition* dilakukan dengan tiga langkah sebagai berikut:

- Melakukan faktorisasi matrik A menjadi matrik L dan matrik U $\rightarrow A = LU$.
- Menghitung vektor $\mathbf{y}$ dengan operasi matrik $L\mathbf{y} = \mathbf{b}$. Ini adalah proses *forward-substitution* atau substitusi-maju.
- Menghitung vektor $\mathbf{x}$ dengan operasi matrik $U\mathbf{x} = \mathbf{y}$. Ini adalah proses *backward-substitution* atau substitusi-mundur.

Metode *LU-decomposition* bisa dibilang merupakan modifikasi dari eliminasi gauss, karena beberapa langkah yang mesti dibuang pada eliminasi gauss, justru harus dipakai oleh *LU-decomposition*. Untuk lebih jelasnya, perhatikan contoh berikut ini. Diketahui sistem persamaan linear sebagai berikut

$$\begin{array}{lrcrcrcrcl}
P_1: & x_1 & + & x_2 & & & + & 3x_4 & = & 4 \\
P_2: & 2x_1 & + & x_2 & - & x_3 & + & x_4 & = & 1 \\
P_3: & 3x_1 & - & x_2 & - & x_3 & + & 2x_4 & = & \text{-3} \\
P_4: & -x_1 & + & 2x_2 & + & 3x_3 & - & x_4 & = & 4
\end{array}$$

Sistem tersebut dapat dinyatakan dalam operasi matrik $A\mathbf{x} = \mathbf{y}$,

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix} \tag{5.5}$$

Pada metode eliminasi gauss, matrik A dikonversi menjadi matrik triangular melalui urutan operasi-operasi berikut: $(P_2 - 2P_1) \rightarrow (P_2)$, $(P_3 - 3P_1) \rightarrow (P_3)$, $(P_4 - (-1)P_1) \rightarrow (P_4)$, $(P_3 - 4P_2) \rightarrow (P_3)$, $(P_4 - (-3)P_2) \rightarrow (P_4)$. Disisi lain, vektor $\mathbf{b}$ ikut berubah nilainya menyesuaikan

proses triangularisasi,

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \\ 13 \\ -13 \end{bmatrix} \tag{5.6}$$

Lain halnya dengan metode *LU-decomposition* dimana vektor **b** tidak mengalami perubahan. Yang berubah hanya matrik A saja, yaitu menjadi matrik L dan matrik U, $A = LU$

$$A = \begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix}$$

Jadi matrik L dan U masing-masing adalah

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix}$$

Coba bandingkan matrik U di atas dengan matrik hasil triangularisasi dari metode eliminasi gauss pada persamaan (5.6), sama persis bukan? Jadi, cara memperoleh matrik U adalah dengan proses triangularisasi! Lantas, bagaimana cara memperoleh matrik L? Begini caranya: (1) elemen-elemen diagonal matrik L diberi nilai 1 (Asal tahu saja, cara ini dikenal dengan metode *Doolittle*). (2) elemen-elemen matrik L yang berada di atas elemen-elemen diagonal diberi nilai 0. (3) sedangkan, elemen-elemen matrik L yang berada di bawah elemen-elemen diagonal diisi dengan faktor pengali yang digunakan pada proses triangularisasi eliminasi gauss. Misalnya pada operasi $(P_2 - 2P_1) \rightarrow (P_2)$, maka faktor pengalinya adalah 2; pada operasi $(P_3 - 3P_1) \rightarrow (P_3)$, maka faktor pengalinya adalah 3, dan seterusnya.

Inilah letak perbedaannya, seluruh faktor pengali tersebut sangat dibutuhkan pada metode *LU-decomposition* untuk membentuk matrik L. Padahal dalam metode eliminasi gauss, seluruh faktor pengali tersebut tidak dimanfaatkan alias dibuang begitu saja. Disisi lain, vektor **b** tidak mengalami proses apapun sehingga nilainya tetap. Jadi, proses konversi matrik pada metode *LU-decomposition* hanya melibatkan matrik A saja!

Setelah langkah faktorisasi matrik A dilalui, maka operasi matrik pada persamaan (5.5) menjadi,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix} \tag{5.7}$$

Langkah berikutnya adalah menentukan vektor **y**, dimana $L\mathbf{y} = \mathbf{b}$,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix}$$

Dengan proses substitusi-maju, elemen-elemen vektor **y** dapat ditentukan,

$$\begin{aligned} y_1 &= 4, \\ 2y_1 + y_2 &= 1, \\ 3y_1 + 4y_2 + y_3 &= -3, \\ -y_1 - 3y_2 + y_4 &= 4 \end{aligned}$$

maka diperoleh $y_1 = 4$, $y_2 = -7$, $y_3 = 13$, $y_4 = -13$.

Langkah terakhir adalah proses substitusi-mundur untuk menghitung vektor **x**, dimana $U\mathbf{x} = \mathbf{y}$,

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \\ 13 \\ -13 \end{bmatrix}$$

Melalui proses ini, yang pertama kali didapat solusinya adalah x_4, kemudian x_3, lalu diikuti x_2, dan akhirnya x_1.

$$\begin{aligned} x_4 &= 1 \\ x_3 &= \frac{1}{3}(13 - 13x_4) = 0 \\ x_2 &= -(-7 + 5x_4 + x_3) = 2 \\ x_1 &= 4 - 3x_4 - x_2 = -1 \end{aligned}$$

akhirnya diperoleh solusi $x_1 = -1$, $x_2 = 2$, $x_3 = 0$, dan $y_4 = 1$. Demikianlah contoh penyelesaian sistem persamaan linear dengan metode *LU-decomposition*.

Sekali matrik A difaktorkan, maka vektor **b** bisa diganti nilainya sesuai dengan sistem persamaan linear yang lain, misalnya seluruh nilai di ruas kanan diganti menjadi

$$\begin{array}{lrcrcrcrcr} P_1: & x_1 & + & x_2 & & & + & 3x_4 & = & 8 \\ P_2: & 2x_1 & + & x_2 & - & x_3 & + & x_4 & = & 7 \\ P_3: & 3x_1 & - & x_2 & - & x_3 & + & 2x_4 & = & 14 \\ P_4: & -x_1 & + & 2x_2 & + & 3x_3 & - & x_4 & = & \text{-7} \end{array}$$

Dalam operasi matrik menjadi

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 14 \\ -7 \end{bmatrix} \tag{5.8}$$

Perhatikan baik-baik! Matrik A sama persis dengan contoh sebelumnya. Perbedaannya hanya pada vektor **b**. Selanjutnya, dengan metode *LU-decomposition*, persamaan (5.8) menjadi

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 14 \\ -7 \end{bmatrix} \tag{5.9}$$

Silakan anda lanjutkan proses perhitungannya dengan mencari vektor **y** sesuai contoh yang telah diberikan sebelumnya. Pada akhirnya akan diperoleh solusi sebagai berikut: $x_1 = 3$, $x_2 = -1$, $x_3 = 0$, dan $y_4 = 2$.

5.2 Algoritma

Sekarang saatnya saya tunjukkan algoritma metode LU decomposition. Algoritma ini dibuat untuk menyelesaikan sistem persamaan linear, dengan cara menfaktorkan matrik $A = (a_{ij})$ berukuran n x n menjadi matrik $L = (l_{ij})$ dan matrik $U = (u_{ij})$ dengan ukuran yang sama. Algoritma LU-decomposition yang anda lihat sekarang merupakan modifikasi dari algoritma eliminasi gauss. Silakan anda periksa langkah-langkah mana saja yang telah mengalami modifikasi! Tapi asal tahu saja bahwa ini bukan satu-satunya algoritma untuk mendapatkan matrik LU. Sejauh yang saya tahu, ada algoritma lain untuk tujuan yang sama, dimana algoritma tersebut membutuhkan matrik permutasi untuk menggeser elemen pivot yang bernilai nol agar terhindar dari singular. Nah, sedangkan algoritma yang akan anda baca saat ini, sama sekali tidak "berurusan" dengan matrik permutasi. Algoritma ini cuma memanfaatkan "trik" tukar posisi yang sudah pernah dibahas di awal-awal catatan khususnya ketika membahas konsep eliminasi gauss.

Satu lagi yang harus saya sampaikan juga adalah bahwa dalam algoritma ini, elemen-elemen matrik L dan matrik U digabung jadi satu dan menggantikan seluruh elemen-elemen matrik A. Perhatian! cara ini jangan diartikan sebagai perkalian matrik L dan matrik U menjadi matrik A kembali. Cara ini dimaksudkan untuk menghemat memori komputer. Suatu aspek yang tidak boleh diabaikan oleh para programer. Marilah kita simak algoritmanya bersama-sama!

INPUT: dimensi n; nilai elemen a_{ij}, $1 \leq i, j \leq n$; nilai elemen b_i.

OUTPUT: solusi $x_1, x_2, x_3, ..., x_n$ atau pesan kesalahan yang mengatakan bahwa *faktorisasi tidak mungkin dilakukan*.

- **Langkah 1:** Inputkan konstanta-konstanta dari sistem persamaan linear kedalam elemen-elemen matrik A dan vektor **b**, seperti berikut ini:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{5.10}$$

- **Langkah 2:** Untuk $i = 1, ..., n-1$, lakukan Langkah 3 sampai Langkah 5.

  - **Langkah 3:** Definisikan p sebagai integer dimana $i \leq p \leq n$. Lalu pastikan bahwa $a_{pi} \neq 0$. Langkah dilakukan bila ditemukan elemen diagonal yang bernilai nol ($a_{ii} = 0$). Ketika ada elemen diagonal yang bernilai nol, maka program harus mencari dan memeriksa elemen-elemen yang tidak bernilai nol dalam kolom yang sama dengan kolom tempat elemen diagonal tersebut berada. Jadi saat proses ini berlangsung, integer i (indeks dari kolom) dibuat konstan, sementara integer p (indeks dari baris) bergerak dari $p = i$ sampai $p = n$. Bila ternyata setelah mencapai elemen paling bawah dalam kolom tersebut, yaitu saat $p = n$ tetap didapat nilai $a_{pi} = 0$, maka sebuah pesan dimunculkan: *sistem persamaan linear tidak memiliki solusi yang unik*. Lalu program berakhir: STOP.

  - **Langkah 4:** Namun jika sebelum integer p mencapai nilai $p = n$ sudah diperoleh elemen yang tidak sama dengan nol ($a_{pi} \neq 0$), maka bisa dipastikan $p \neq i$. Jika $p \neq i$ maka lakukan proses pertukaran $(P_p) \leftrightarrow (P_i)$.

  - **Langkah 5:** Untuk $j = i + 1, .., n$, lakukan Langkah 6 dan Langkah 7.

    - **Langkah 6:** Tentukan m_{ji},

$$m_{ji} = \frac{a_{ji}}{a_{ii}}$$

    - **Langkah 7:** Lakukan proses triangularisasi,

$$(P_j - m_{ji}P_i) \rightarrow (P_j)$$

    - **Langkah 8:** Nilai m_{ji} disimpan ke a_{ji},

$$a_{ji} = m_{ji}$$

- **Langkah 9:** Nilai b_1 dicopy ke y_1, lalu lakukan substitusi-maju.

$$y_1 = b_1$$

  Untuk $i = 2, ..., n$ tentukan x_i,

$$y_i = b_i - \sum_{j=1}^{i-1} a_{ij} y_j$$

- **Langkah 10:** Lakukan proses substitusi-mundur, dimulai dengan menentukan x_n,

$$x_n = \frac{a_{n,n+1}}{a_{nn}}$$

Untuk $i = n-1, ..., 1$ tentukan x_i,

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

- **Langkah 11:** Diperoleh solusi yaitu $x_1, x_2, ..., x_n$. Algoritma telah dijalankan dengan sukses. STOP.

Algoritma di atas telah diimplementasi kedalam program yang ditulis dengan bahasa Fortran. Program tersebut sudah berhasil dikompilasi dengan visual fortran (windows) dan g77 (debian-linux). Inilah programnya:

```
      DIMENSION A(10,11), B(10), Y(10), X(10)
      REAL MJI
      WRITE(*,*)
      WRITE(*,*) '==> FAKTORISASI MATRIK: LU DECOMPOSITION <=='
      WRITE (*,*)
C     LANGKAH 1: MEMASUKAN NILAI ELEMEN-ELEMEN MATRIK A DAN VEKTOR B
      WRITE (*,'(1X,A)') 'JUMLAH PERSAMAAN ? '
      READ (*,*) N
      WRITE (*,*)
      WRITE (*,*) 'MASUKAN ELEMEN-ELEMEN MATRIK A'
      DO 50 I = 1,N
        DO 60 J = 1,N
          WRITE (*,'(1X,A,I2,A,I2,A)') 'A(',I,',',J,') = '
          READ (*,*) A(I,J)
60      CONTINUE
        WRITE (*,'(1X,A,I2,A)') 'B(',I,') ? '
        READ (*,*) B(I)
        WRITE (*,*)
50    CONTINUE
      WRITE (*,*)
C     MENAMPILKAN MATRIK A
      WRITE (*,'(1X,A)') 'MATRIK A:'
      DO 110 I = 1,N
        WRITE (*,6) (A(I,J),J=1,N)
110   CONTINUE
      WRITE (*,*)
C     LANGKAH 2: MEMERIKSA ELEMEN-ELEMEN PIVOT
      NN = N-1
      DO 10 I=1,NN
C       LANGKAH 3: MENDEFINISIKAN P
        P = I
100     IF (ABS(A(P,I)).GE.1.0E-20 .OR. P.GT.N) GOTO 200
        P = P+1
        GOTO 100
200     IF(P.EQ.N+1)THEN
C       MENAMPILKAN PESAN TIDAK DAPAT DIFAKTORKAN
          WRITE(*,8)
          GOTO 400
```

```
        END IF
C       LANGKAH 4: PROSES TUKAR POSISI
        IF(P.NE.I) THEN
          DO 20 JJ=1,N
            C = A(I,JJ)
            A(I,JJ) = A(P,JJ)
            A(P,JJ) = C
20        CONTINUE
        END IF
C       LANGKAH 5: PERSIAPAN PROSES TRIANGULARISASI
        JJ = I+1
        DO 30 J=JJ,N
C         LANGKAH 6: TENTUKAN MJI
          MJI = A(J,I)/A(I,I)
C         LANGKAH 7: PROSES TRIANGULARISASI
          DO 40 K=JJ,N
            A(J,K) = A(J,K)-MJI*A(I,K)
40        CONTINUE
C         LANGKAH 8: MENYIMPAN MJI KE A(J,I)
          A(J,I) = MJI
30      CONTINUE
10    CONTINUE
C     MENAMPILKAN MATRIK LU
      WRITE (*,'(1X,A)') 'MATRIK LU:'
      DO 120 I = 1,N
        WRITE (*,6) (A(I,J),J=1,N)
120   CONTINUE
      WRITE (*,*)
C     LANGKAH 9: SUBSTITUSI-MAJU
      Y(1) = B(1)
      DO 15 I=2,N
        SUM = 0.0
        DO 16 J=1,I-1
          SUM = SUM+A(I,J)*Y(J)
16      CONTINUE
        Y(I) = B(I)-SUM
15    CONTINUE
C     MENAMPILKAN VEKTOR Y
      WRITE (*,'(1X,A)') 'VEKTOR Y:'
      DO 138 I = 1,N
        WRITE (*,6) Y(I)
138   CONTINUE
      WRITE (*,*)
C     LANGKAH 10: SUBSTITUSI-MUNDUR
      X(N) = Y(N)/A(N,N)
      DO 24 K=1,N-1
        I = N-K
        JJ = I+1
        SUM = 0.0
        DO 26 KK=JJ,N
          SUM = SUM+A(I,KK)*X(KK)
26      CONTINUE
        X(I) = (Y(I)-SUM)/A(I,I)
24    CONTINUE
C     LANGKAH 11: MENAMPILKAN SOLUSI DAN SELESAI
      WRITE (*,'(1X,A)') 'SOLUSI:'
      DO 18 I = 1,N
        WRITE (*,'(1X,A,I2,A,F14.8)') 'X(',I,') = ',X(I)
18    CONTINUE
```

```
98        WRITE(*,*)
99        WRITE(*,*) 'SELESAI --> SUKSES'
100       WRITE(*,*)
101   400 CONTINUE
102   6   FORMAT(1X,5(F14.8))
103   8   FORMAT(1X,'TIDAK DAPAT DIFAKTORKAN')
104       END
```

Demikianlah, sekarang kita punya tiga buah algoritma untuk memecahkan problem sistem persamaan linear, yaitu eliminasi gauss, invers matrik, dan lu-decomposition. Diantara ketiganya, eliminasi gauss adalah algoritma yang paling simpel dan efisien. Dia hanya butuh proses triangularisasi dan substitusi-mundur untuk mendapatkan solusi. Sedangkan dua algoritma yang lainnya membutuhkan proses-proses tambahan untuk mendapatkan solusi yang sama.

Saya cukupkan sementara sampai disini. Insya Allah akan saya sambung lagi dilain waktu. Kalau ada yang mau didiskusikan, silakan hubungi saya melalui email.

Bab 6

Metode Iterasi

Objektif :

- ▷ Mengenalkan konsep Norm.
- ▷ Mengenalkan iterasi Jacobi.
- ▷ Mengenalkan iterasi Gauss-Seidel.
- ▷ Mengenalkan iterasi *Succesive-Over-Relaxation (SOR)*.

6.1 Kelebihan Vektor-kolom

Sebelum kita membahas metode iterasi untuk menyelesaikan problem sistem persamaan linear, saya ingin menyampaikan satu hal yang sangat sederhana, yaitu tentang cara merepresentasikan elemen-elemen suatu vektor-kolom. Sebagaimana tertulis pada bab-bab sebelumnya, biasanya suatu vektor-kolom ditulis sebagai

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{6.1}$$

Dengan operasi transpose, vektor-kolom tersebut dapat dinyatakan sebagai

$$\mathbf{x} = \begin{bmatrix} x_1; & x_2; & \dots & x_n \end{bmatrix}^T \tag{6.2}$$

Contoh:

$$\mathbf{x} = \begin{bmatrix} 3 \\ -2 \\ 8 \\ 5 \end{bmatrix} = \begin{bmatrix} 3; & -2; & 8; & 5 \end{bmatrix}^T$$

Cara penulisan seperti ini digunakan untuk menyatakan vektor-kolom pada suatu kalimat di dalam paragraf. Alasannya supaya tidak terlalu menyita banyak ruang penulisan. Sementara, persamaan (6.1), lebih sering digunakan pada penulisan operasi matrik. Satu hal lagi, pada paragraf-paragraf berikutnya, saya persingkat penulisan istilah vektor-kolom menjadi vektor saja.

6.2 Pengertian Norm

Vektor **x**=$(x_1; x_2; ...; x_n)^T$ memiliki norm ℓ_2 dan ℓ_∞ yang didefinisikan sebagai

$$\ell_2 = \|x\|_2 = \{\sum_{i=1}^{n} x_i^2\}^{1/2} \tag{6.3}$$

dan

$$\ell_\infty = \|x\|_\infty = \max_{1 \le i \le n} |x_i| \tag{6.4}$$

Contoh: **x**=$(3; -2; 8; 5)^T$ memiliki norm ℓ_2 yaitu

$$\ell_2 = \|x\|_2 = \sqrt{(3)^2 + (-2)^2 + (8)^2 + (5)^2} = 10,0995$$

dan norm ℓ_∞ yaitu

$$\ell_\infty = \|x\|_\infty = \max\{(3), (-2), (8), (5)\} = 8$$

Saya menyarankan agar kedua norm ini diingat-ingat dengan baik, karena akan banyak disinggung pada catatan-catatan berikutnya.

6.2.1 *Script* perhitungan norm dua

Script berikut ini merujuk pada contoh di atas, dimana vektor **x** hanya terdiri dari 4 elemen, yaitu $x(1,1)$,$x(2,1)$,$x(3,1)$ dan $x(4,1)$

```
clear all
clc

x = [ 3 ; -2 ; 8 ; 5 ];

dim = size(x);
n = dim(1);

S = 0;
for i = 1:n
   S = S + x(i,1)^2;
end
hasil = sqrt(S);
```

Berdasarkan script di atas, dapat dibuat fungsi eksternal sebagai berikut:

```
function hasil = norm2(x)

```

```
dim = size(x);
n = dim(1);
S = 0;
for i = 1:n
   S = S + x(i,1)^2;
end
hasil = sqrt(S);
```

6.2.2 *Script* perhitungan norm tak hingga

Script berikut ini merujuk pada contoh di atas, dimana vektor **x** hanya terdiri dari 4 elemen, yaitu $x(1,1)$,$x(2,1)$,$x(3,1)$ dan $x(4,1)$

```
clear all
clc

x = [ 3 ; -9 ; 8 ; 5 ];

dim = size(x);
n = dim(1);
xx = x;
for i=1:n
   if xx(i,1) < 0
      xx(i,1) = xx(i,1) * -1;
   end
end
hasil = max(xx);
```

Script ini menggunakan fungsi internal yang bernama *max()* untuk mendapatkan nilai elemen terbesar diantara elemen-elemen yang ada dalam vektor x. Berdasarkan script di atas, dapat dibuat fungsi eksternal sebagai berikut:

```
function hasil = normth(x)

dim = size(x);
n = dim(1);
xx = x;
for i=1:n
   if xx(i,1) < 0
      xx(i,1) = xx(i,1) * -1;
   end
end
hasil = max(xx);
```

6.2.3 Perhitungan norm-selisih

Misalnya kita punya vektor bernama **xlama**. Lalu ada vektor lainnya bernama **xbaru**. Norm selisih dari **xlama** dan **xbaru** dapat dihitung dengan bantuan fungsi eksternal yang baru saja kita buat di atas, yaitu bernama *norm2()* dan *normth()*.

```
clear all
clc
```

```
3
4  xlama = [ 3 ; -2 ; 8 ; 5 ];
5  xbaru = [ 9 ; 4 ; 6 ; 1 ];
6
7  xselisih = xbaru-xlama;
8  hasil1 = norm2(xselisih);
9  hasil2 = normth(xselisih);
```

Cara perhitungan norm-selisih seperti ini akan diterapkan pada kebanyakan metode iterasi. Jadi tolong diingat baik-baik!!

6.3 Iterasi Jacobi

Sekarang kita akan mulai membahas metode iterasi sekaligus penerapannya untuk menyelesaikan sistem persamaan linear. Perbedaan metode iterasi dengan metode-metode yang telah dijelaskan sebelumnya, adalah ia dimulai dari penentuan nilai awal (*initial value*) untuk setiap elemen vektor **x**. Kemudian berdasarkan nilai awal tersebut, dilakukan langkah perhitungan untuk mendapatkan elemen-elemen vektor **x** yang baru. Untuk lebih jelasnya, silakan perhatikan baik-baik contoh berikut. Diketahui sistem persamaan linear terdiri atas empat persamaan, yaitu

$$\begin{aligned} 10x_1 - x_2 + 2x_3 &= 6 \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25 \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11 \\ 3x_2 - x_3 + 8x_4 &= 15 \end{aligned}$$

yang mana solusinya adalah **x**=$(1; 2; -1; 1)^T$. Silakan simpan dulu solusi ini, anggap saja kita belum tahu. Lalu perhatikan baik-baik bagaimana metode iterasi Jacobi bisa menemukan solusi tersebut dengan caranya yang khas.

Langkah pertama dan merupakan langkah terpenting dari metode iterasi Jacobi adalah mengubah cara penulisan sistem persamaan linear di atas menjadi seperti ini

$$\begin{aligned} x_1 &= \frac{1}{10}x_2 - \frac{2}{10}x_3 + \frac{6}{10} \\ x_2 &= \frac{1}{11}x_1 + \frac{1}{11}x_3 - \frac{3}{11}x_4 + \frac{25}{11} \\ x_3 &= -\frac{2}{10}x_1 + \frac{1}{10}x_2 + \frac{1}{10}x_4 - \frac{11}{10} \\ x_4 &= -\frac{3}{8}x_2 + \frac{1}{8}x_3 + \frac{15}{8} \end{aligned}$$

Kita bisa menyatakan bahwa nilai x_1, x_2, x_3 dan x_4 yang berada di ruas kiri tanda = (baca: sama dengan) sebagai $x^{(baru)}$. Sementara nilai x_1, x_2, x_3 dan x_4 yang berada di ruas kanan tanda = (baca: sama dengan) sebagai $x^{(lama)}$. Sehingga sistem persamaan tersebut dapat ditulis seperti

ini

$$\begin{aligned} x_1^{(baru)} &= \frac{1}{10}x_2^{(lama)} - \frac{2}{10}x_3^{(lama)} + \frac{6}{10} \\ x_2^{(baru)} &= \frac{1}{11}x_1^{(lama)} + \frac{1}{11}x_3^{(lama)} - \frac{3}{11}x_4^{(lama)} + \frac{25}{11} \\ x_3^{(baru)} &= -\frac{2}{10}x_1^{(lama)} + \frac{1}{10}x_2 + \frac{1}{10}x_4^{(lama)} - \frac{11}{10} \\ x_4^{(baru)} &= -\frac{3}{8}x_2^{(lama)} + \frac{1}{8}x_3^{(lama)} + \frac{15}{8} \end{aligned}$$

yang secara umum dapat diformulasikan sebagai persamaan matrik berikut ini

$$\mathbf{x}^{(baru)} = J\mathbf{x}^{(lama)} + u \tag{6.5}$$

dimana

$$\begin{bmatrix} x_1^{(baru)} \\ x_2^{(baru)} \\ x_3^{(baru)} \\ x_4^{(baru)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(lama)} \\ x_2^{(lama)} \\ x_3^{(lama)} \\ x_4^{(lama)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

Atau dapat pula ditulis seperti ini

$$\mathbf{x}^k = J\mathbf{x}^{k-1} + u \tag{6.6}$$

dimana $k = 1, 2, 3, ..., n$; sehingga persamaan matrik dapat dinyatakan sebagai berikut

$$\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \\ x_4^{(k)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \\ x_3^{(k-1)} \\ x_4^{(k-1)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

Pada persamaan di atas, indeks k menunjukan jumlah perhitungan iterasi. Pada $k = 1$, maka penulisan sistem persamaan linear menjadi

$$\begin{aligned} x_1^{(1)} &= \frac{1}{10}x_2^{(0)} - \frac{2}{10}x_3^{(0)} + \frac{6}{10} \\ x_2^{(1)} &= \frac{1}{11}x_1^{(0)} + \frac{1}{11}x_3^{(0)} - \frac{3}{11}x_4^{(0)} + \frac{25}{11} \\ x_3^{(1)} &= -\frac{2}{10}x_1^{(0)} + \frac{1}{10}x_2^{(0)} + \frac{1}{10}x_4^{(0)} - \frac{11}{10} \\ x_4^{(1)} &= -\frac{3}{8}x_2^{(0)} + \frac{1}{8}x_3^{(0)} + \frac{15}{8} \end{aligned}$$

Jika kita tentukan nilai-nilai awal $\mathbf{x}^{(0)}$ sebagai berikut $x_1^{(0)} = 0$, $x_2^{(0)} = 0$, $x_3^{(0)} = 0$ dan $x_4^{(0)} = 0$. Atau dinyatakan seperti ini $\mathbf{x}^{(0)} = (0; 0; 0; 0)^T$. Maka kita akan memperoleh nilai-nilai $\mathbf{x}^{(1)}$ yang

tidak lain adalah hasil perhitungan iterasi pertama, yaitu

$$\begin{aligned} x_1^{(1)} &= \frac{6}{10} \\ x_2^{(1)} &= \frac{25}{11} \\ x_3^{(1)} &= -\frac{11}{10} \\ x_4^{(1)} &= \frac{15}{8} \end{aligned}$$

atau $\mathbf{x}^{(1)} = (0,6000; 2,2727; -1,1000; 1,8750)^T$. Setelah nilai-nilai $\mathbf{x}^{(1)}$ diperoleh, perhitungan tersebut diulang kembali guna mendapatkan hasil iterasi kedua, yaitu ketika $k = 2$. Caranya adalah dengan memasukan nilai-nilai $\mathbf{x}^{(1)} = (0,6000; 2,2727; -1,1000; 1,8750)^T$ ke suku-suku pada ruas kanan tanda sama-dengan,

$$\begin{aligned} x_1^{(2)} &= \frac{1}{10}x_2^{(1)} - \frac{2}{10}x_3^{(1)} + \frac{6}{10} \\ x_2^{(2)} &= \frac{1}{11}x_1^{(1)} + \frac{1}{11}x_3^{(1)} - \frac{3}{11}x_4^{(1)} + \frac{25}{11} \\ x_3^{(2)} &= -\frac{2}{10}x_1^{(1)} + \frac{1}{10}x_2^{(1)} + \frac{1}{10}x_4^{(1)} - \frac{11}{10} \\ x_4^{(2)} &= -\frac{3}{8}x_2^{(1)} + \frac{1}{8}x_3^{(1)} + \frac{15}{8} \end{aligned}$$

maka nilai-nilai $\mathbf{x}^{(2)}$ yang kita dapat adalah $\mathbf{x}^{(2)} = (1,0473; 1,7159; -0,8052; 0,8852)^T$. Setelah diperoleh nilai-nilai $\mathbf{x}^{(2)}$, perhitungan tersebut diulangi kembali guna mendapatkan hasil iterasi ketiga, dimana nilai $k = 3$. Caranya adalah dengan memasukan nilai-nilai $\mathbf{x}^{(2)} = (1,0473; 1,7159; -0,8052; 0,8852)^T$ ke ruas kanan kembali,

$$\begin{aligned} x_1^{(3)} &= \frac{1}{10}x_2^{(2)} - \frac{2}{10}x_3^{(2)} + \frac{6}{10} \\ x_2^{(3)} &= \frac{1}{11}x_1^{(2)} + \frac{1}{11}x_3^{(2)} - \frac{3}{11}x_4^{(2)} + \frac{25}{11} \\ x_3^{(3)} &= -\frac{2}{10}x_1^{(2)} + \frac{1}{10}x_2^{(2)} + \frac{1}{10}x_4^{(2)} - \frac{11}{10} \\ x_4^{(3)} &= -\frac{3}{8}x_2^{(2)} + \frac{1}{8}x_3^{(2)} + \frac{15}{8} \end{aligned}$$

maka kita akan memperoleh nilai-nilai $\mathbf{x}^{(3)} = (0,9326; 2,0530; -1,0493; 1,1309)^T$. Lalu proses perhitungan diulangi lagi dengan $k = 4$. Begitulah seterusnya. Proses ini diulangi lagi berkali-kali untuk nilai-nilai k berikutnya. Proses yang berulang ini disebut proses **iterasi**.

Sampai dengan $\mathbf{x}^{(3)}$ di atas, kita sudah melakukan tiga kali proses iterasi. Lantas sampai kapan proses iterasi ini terus berlanjut? Jawabnya adalah sampai $\mathbf{x}^{(baru)}$ mendekati solusi yang tepat, yaitu

$$\mathbf{x} = (1; 2; -1; 1)^T$$

Dengan kata lain, proses iterasi harus dihentikan bila $\mathbf{x}^{(baru)}$ sudah mendekati solusi. Lalu kriteria apa yang digunakan sehingga suatu hasil iterasi bisa dikatakan paling dekat dengan

solusi yang sebenarnya? OK, simpan dulu pertanyaan ini, sebagai gantinya marilah kita pelajari terlebih dahulu *script* Matlab untuk metode iterasi Jacobi.

6.3.1 *Script* metode iterasi Jacobi

Sebagaimana biasa, saya tidak akan memberikan script yang sudah matang kepada anda. Saya lebih suka mengajak anda mengikuti proses optimalisasi script; mulai dari yang mentah hingga matang. Sebagai upaya pembelajaran, sengaja saya mulai dengan menampilkan *script* yang paling kasar terlebih dahulu, lalu selangkah demi selangkah dimodifikasi hingga menjadi *script* efisien.

Mari kita mulai dengan menampilkan kembali sistem persamaan linear pada contoh diatas, yaitu

$$\begin{aligned} 10x_1 - x_2 + 2x_3 &= 6 \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25 \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11 \\ 3x_2 - x_3 + 8x_4 &= 15 \end{aligned}$$

Sistem persamaan linear tersebut dapat dinyatakan dalam persamaan matrik

$$\begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}$$

Langkah pertama adalah mendeklarasikan matrik A dan vektor b sebagai berikut

```
clear all
clc

%--- inisialisasi matrik A --
A = [10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];

%--- inisialisasi vektor b ---
b = [6 ; 25 ; -11 ; 15];
```

Kemudian, sistem persamaan linear di atas dimodifikasi menjadi

$$\begin{aligned} x_1 &= \frac{1}{10}x_2 - \frac{2}{10}x_3 + \frac{6}{10} \\ x_2 &= \frac{1}{11}x_1 + \frac{1}{11}x_3 - \frac{3}{11}x_4 + \frac{25}{11} \\ x_3 &= -\frac{2}{10}x_1 + \frac{1}{10}x_2 + \frac{1}{10}x_4 - \frac{11}{10} \\ x_4 &= -\frac{3}{8}x_2 + \frac{1}{8}x_3 + \frac{15}{8} \end{aligned}$$

Dan ditulis kembali dalam persamaan matrik sebagai berikut

$$\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \\ x_4^{(k)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \\ x_3^{(k-1)} \\ x_4^{(k-1)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

Saya nyatakan suatu matrik **J** dan vektor *u* sebagai berikut

$$J = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \qquad u = \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

Inilah script untuk membuat matrik **J** dan vektor *u*,

```
clear all
clc

A = [ 10 -1 2 0;
   -1 11 -1 3;
   2 -1 10 -1;
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

[m,n] = size(A);  % ---- m = jumlah baris ; n = jumlah kolom

J = zeros(4);   % ---- inisialisasi matrik J dengan angka nol

%---- inisialisasi elemen-elemen matrik J dan vektor u
J(1,2) = -A(1,2)/A(1,1);
J(1,3) = -A(1,3)/A(1,1);
J(1,4) = -A(1,4)/A(1,1);
u(1,1) = b(1,1)/A(1,1);

J(2,1) = -A(2,1)/A(2,2);
J(2,3) = -A(2,3)/A(2,2);
J(2,4) = -A(2,4)/A(2,2);
u(2,1) = b(2,1)/A(2,2);

J(3,1) = -A(3,1)/A(3,3);
```

```
27 J(3,2) = -A(3,2)/A(3,3);
28 J(3,4) = -A(3,4)/A(3,3);
29 u(3,1) = b(3,1)/A(3,3);
30
31 J(4,1) = -A(4,1)/A(4,4);
32 J(4,2) = -A(4,2)/A(4,4);
33 J(4,3) = -A(4,3)/A(4,4);
34 u(4,1) = b(4,1)/A(4,4);
```

Statemen baris 16 sampai 34 berfungsi menghitung elemen matrik **J** dan vektor u. Untuk menyederhanakan baris 16 hingga 19, kita buat proses looping dengan indeks k, tetapi dengan pengecualian pada k=1.

```
for k = 1:4
   if (k ~= 1)
      J(1,k) = -A(1,k)/A(1,1);
   end
end
u(1,1) = b(1,1)/A(1,1);
```

Mulai dari baris 21 hingga 24 juga bisa dibuat proses looping dengan pengecualian pada k=2.

```
for k = 1:4
   if (k ~= 2)
      J(2,k) = -A(2,k)/A(2,2);
   end
end
u(2,1) = b(2,1)/A(2,2);
```

Proses looping yang sama juga diterapkan terhadap baris ke-26 hingga ke-29.

```
for k = 1:4
   if (k ~= 3)
      J(3,k) = -A(3,k)/A(3,3);
   end
end
u(3,1) = b(3,1)/A(3,3);
```

Sementara untuk baris ke-31 hingga ke-34, penyerderhanaan dilakukan dengan cara yang sama pula

```
for k = 1:4
   if (k ~= 4)
      J(4,k) = -A(4,k)/A(4,4);
   end
end
u(4,1) = b(4,1)/A(4,4);
```

Kalau seluruh penyederhanaan ini digabung, maka scriptnya akan seperti ini

```
1 clear all
2 clc
```

```
3
4  A = [ 10 -1 2 0
5     -1 11 -1 3
6     2 -1 10 -1
7     0 3 -1 8];
8
9  b = [ 6 ; 25 ; -11 ; 15 ];
10
11 [m,n] = size(A);
12
13 J = zeros(4);
14
15 for k = 1:4
16    if (k ~= 1)
17       J(1,k) = -A(1,k)/A(1,1);
18    end
19 end
20 u(1,1) = b(1,1)/A(1,1);
21
22 for k = 1:4
23    if (k ~= 2)
24       J(2,k) = -A(2,k)/A(2,2);
25    end
26 end
27 u(2,1) = b(2,1)/A(2,2);
28
29 for k = 1:4
30    if (k ~= 3)
31       J(3,k) = -A(3,k)/A(3,3);
32    end
33 end
34 u(3,1) = b(3,1)/A(3,3);
35
36 for k = 1:4
37    if (k ~= 4)
38       J(4,k) = -A(4,k)/A(4,4);
39    end
40 end
41 u(4,1) = b(4,1)/A(4,4);
```

Selanjutnya, saya tampilkan indeks p. Perhatikan penempatannya

```
1  clear all
2  clc
3
4  A = [ 10 -1 2 0
5     -1 11 -1 3
6     2 -1 10 -1
7     0 3 -1 8];
8
9  b = [ 6 ; 25 ; -11 ; 15 ];
10
11 [m,n] = size(A);
12
13 J = zeros(4);
14
15 p = 1;
16 for k = 1:4
```

```
   if (k ~= p)
      J(p,k) = -A(p,k)/A(p,p);
   end
end
u(p,1) = b(p,1)/A(p,p);

p = 2;
for k = 1:4
   if (k ~= p)
      J(p,k) = -A(p,k)/A(p,p);
   end
end
u(p,1) = b(p,1)/A(p,p);

p = 3;
for k = 1:4
   if (k ~= p)
      J(p,k) = -A(p,k)/A(p,p);
   end
end
u(p,1) = b(p,1)/A(p,p);

p = 4;
for k = 1:4
   if (k ~= p)
      J(p,k) = -A(p,k)/A(p,p);
   end
end
u(p,1) = b(p,1)/A(p,p);
```

Selanjutnya saya buat proses looping menggunakan indeks p tersebut. Perhatikan baik-baik perubahannya

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

[m,n] = size(A);

J = zeros(4);

for p = 1:4
  for k = 1:4
    if (k ~= p)
       J(p,k) = -A(p,k)/A(p,p);
    end
  end
  u(p,1) = b(p,1)/A(p,p);
end
```

Dan akhirnya, angka 4 dapat digantikan dengan huruf n agar script tersebut tidak dibatasi oleh

matrik 4x4 saja.

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

[m,n] = size(A);

J = zeros(n);

for p = 1:n
  for k = 1:n
    if (k ~= p)
       J(p,k) = -A(p,k)/A(p,p);
    end
  end
  u(p,1) = b(p,1)/A(p,p);
end
```

Selanjutnya, vektor **xlama** diinisialisasi; dan proses iterasi pertama dimulai

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

[m,n] = size(A);

J = zeros(n);

for p = 1:n
   for k = 1:n
      if (k ~= p)
         J(p,k) = -A(p,k)/A(p,p);
      end
   end
   u(p,1) = b(p,1)/A(p,p);
end

xlama = [ 0 ; 0 ; 0 ; 0 ];  % --- inisialisasi xlama
xbaru = J*xlama + u;  % --- iterasi pertama
```

xbaru yang didapat tak lain adalah hasil iterasi pertama, yaitu $\mathbf{x}^{(1)} = (0,6000; 2,2727; -1,1000; 1,8750)^T$. Kemudian, sebelum iterasi ke-2 dilakukan, **xbaru** tersebut mesti disimpan sebagai **xlama**.

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

[m,n] = size(A);

J = zeros(n);

for p = 1:n
   for k = 1:n
      if (k ~= p)
         J(p,k) = -A(p,k)/A(p,p);
      end
   end
   u(p,1) = b(p,1)/A(p,p);
end

xlama = [ 0 ; 0 ; 0 ; 0 ];  % --- inisialisasi xlama
xbaru = J*xlama + u;  % --- iterasi pertama

xlama = xbaru;
xbaru = J*xlama + u;  % --- iterasi kedua
```

Sampai disini, **xbaru** yang didapat dari hasil iterasi ke-2 adalah $\mathbf{x}^{(2)} = (1,0473; 1,7159; -0,8052; 0,8852)^T$. Setelah itu, untuk iterasi ke-3, **xbaru** tersebut mesti disimpan sebagai **xlama** kembali,

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

[m,n] = size(A);

J = zeros(n);

for p = 1:n
   for k = 1:n
      if (k ~= p)
         J(p,k) = -A(p,k)/A(p,p);
      end
   end
   u(p,1) = b(p,1)/A(p,p);
end

xlama = [ 0 ; 0 ; 0 ; 0 ];  % --- inisialisasi xlama
```

```
xbaru = J*xlama + u;  % --- iterasi pertama

xlama = xbaru;
xbaru = J*xlama + u;  % --- iterasi kedua

xlama = xbaru;
xbaru = J*xlama + u;  % --- iterasi ketiga
```

Sampai disini, **xbaru** yang didapat adalah hasil iterasi ke-3, yaitu $\mathbf{x}^{(3)} = (0,9326; 2,0530; -1,0493; 1,1309)^T$. Kemudian, untuk iterasi ke-4, *script* di atas dimodifikasi dengan cara yang sama. Tapi konsekuensinya *script* tersebut akan semakin bertambah panjang. Guna menghindari hal itu, *script* di atas perlu dioptimasi dengan proses looping sebagai berikut

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

[m,n] = size(A);

J = zeros(n);

for p = 1:n
   for k = 1:n
      if (k ~= p)
         J(p,k) = -A(p,k)/A(p,p);
      end
   end
   u(p,1) = b(p,1)/A(p,p);
end

xlama = [ 0 ; 0 ; 0 ; 0 ];  % --- inisialisasi xlama

itermaks = 10;  % --- iteraksi maksimum sampai 10 kali
for k = 1:itermaks
   xbaru = J*xlama + u;
   xlama = xbaru;
end
```

Dalam script di atas, jumlah iterasi dibatasi hanya sampai 10 kali saja. Maka keluaran dari script di atas adalah hanya sampai hasil perhitungan iterasi yang ke-10. Hasil dari keseluruhan iterasi, mulai dari iterasi ke-1 hingga iterasi ke-10 disajikan pada Tabel 6.1.

Berdasarkan Tabel 6.1, terlihat bahwa hasil iterasi ke-1, $\mathbf{x}^{(1)} = (0,6000; 2,2727; -1,1000; 1,8852)^T$ adalah hasil yang paling jauh dari solusi, $\mathbf{x} = (1; 2; -1; 1)^T$. Coba bandingkan dengan hasil iterasi ke-2! Jelas terlihat bahwa hasil iterasi ke-2 lebih mendekati solusi. Kalau terus diurutkan, maka hasil iterasi ke-10 merupakan hasil yang paling dekat dengan solusi.

Sebelum dilanjutkan, saya ingin tuliskan script yang sudah dimodifikasi, dimana semua bagian inisialisasi saya letakkan di baris-baris awal

Tabel 6.1: Hasil akhir elemen-elemen vektor **x** hingga iterasi ke-10

k	0	1	2	3	4	...	9	10
$x_1^{(k)}$	0,0000	0,6000	1,0473	0,9326	1,0152	...	0,9997	1,0001
$x_2^{(k)}$	0,0000	2,2727	1,7159	2,0530	1,9537	...	2,0004	1,9998
$x_3^{(k)}$	0,0000	-1,1000	-0,8052	-1,0493	-0,9681	...	-1,0004	-0,9998
$x_4^{(k)}$	0,0000	1,8852	0,8852	1,1309	0,9739	...	1,0006	0,9998

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];
b = [ 6 ; 25 ; -11 ; 15 ];
xlama = [ 0 ; 0 ; 0 ; 0 ];
itermaks = 10;

[m,n] = size(A);

% --- membuat matrik J dan vektor u ---
J = zeros(n);
for p = 1:n
   for k = 1:n
      if (k ~= p)
         J(p,k) = -A(p,k)/A(p,p);
      end
   end
   u(p,1) = b(p,1)/A(p,p);
end

% --- proses iterasi jacobi ---
for k = 1:itermaks
   xbaru = J*xlama + u;
   xlama = xbaru;
end
```

6.3.2 *Stopping criteria*

Tabel 6.2 memperlihatkan perhitungan norm2-selisih antara **xbaru** dan **xlama** dari iterasi ke-1 hingga iterasi ke-10. Hasil perhitungan norm2-selisih tersebut, saya beri nama epsilon, ϵ, dimana semakin kecil nilai epsilon (ϵ), menandakan hasil iterasinya semakin dekat dengan solusi. Hasil norm2-selisih yang semakin kecil pada iterasi ke-10 menunjukan bahwa hasil iterasi ke-10 adalah hasil yang paling dekat dengan solusi yang sebenarnya.

Tabel 6.2: Hasil perhitungan norm2-selisih hingga iterasi ke-10

norm ℓ_2	$\left\|\mathbf{x}^{(2)}-\mathbf{x}^{(1)}\right\|_2$	$\left\|\mathbf{x}^{(3)}-\mathbf{x}^{(2)}\right\|_2$	$\left\|\mathbf{x}^{(4)}-\mathbf{x}^{(3)}\right\|_2$	...	$\left\|\mathbf{x}^{(10)}-\mathbf{x}^{(9)}\right\|_2$
ϵ	0,6000	0,4473	0,1146	...	0,0004

Berikut ini adalah script untuk mendapatkan nilai epsilon

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];
b = [ 6 ; 25 ; -11 ; 15 ];
xlama = [ 0 ; 0 ; 0 ; 0 ];
itermaks = 10;

[m,n] = size(A);

% --- membuat matrik J dan vektor u ---
J = zeros(n);
for p = 1:n
   for k = 1:n
      if (k ~= p)
         J(p,k) = -A(p,k)/A(p,p);
      end
   end
   u(p,1) = b(p,1)/A(p,p);
end

% --- proses iterasi jacobi ---
for k = 1:itermaks
   xbaru = J*xlama + u;
   xselisih = xbaru - xlama;
   epsilon = norm2(xselisih)
   xlama = xbaru;
end
```

Tanda titik-koma pada baris ke-29 sengaja dihilangkan agar nilai epsilon selalu ditampilkan ketika script tersebut dijalankan.

Nilai epsilon ini begitu penting untuk menentukan kapan proses iterasi harus dihentikan. Oleh karenanya, nilai epsilon difungsikan sebagai *stopping criteria*. Berdasarkan Tabel 6.2, jika nilai ϵ ditentukan sebesar 0,2 , maka proses iterasi akan berhenti pada iterasi ke-4. Atau kalau nilai ϵ ditentukan sebesar 0,0001 , maka proses iterasi akan berhenti pada iterasi ke-10. Kesimpulannya, semakin kecil nilai ϵ, semakin panjang proses iterasinya, namun hasil akhirnya semakin akurat.

Di bawah ini adalah *script* iterasi Jacobi yang memanfaatkan nilai epsilon untuk menghentikan proses iterasi

```
clear all
clc

A = [ 10 -1 2 0
   -1 11 -1 3
   2 -1 10 -1
   0 3 -1 8];
b = [ 6 ; 25 ; -11 ; 15 ];
xlama = [ 0 ; 0 ; 0 ; 0 ];
itermaks = 1000;
epsilon = 0.0001;
```

```
12
13 [m,n] = size(A);
14
15 % --- membuat matrik J dan vektor u ---
16 J = zeros(n);
17 for p = 1:n
18    for k = 1:n
19       if (k ~= p)
20          J(p,k) = -A(p,k)/A(p,p);
21       end
22    end
23    u(p,1) = b(p,1)/A(p,p);
24 end
25
26 % --- proses iterasi jacobi ---
27 for k = 1:itermaks
28    xbaru = J*xlama + u;
29    xselisih = xbaru - xlama;
30    if (norm2(xselisih) < epsilon)
31       break;
32    end
33    xlama = xbaru;
34 end
35 iterasi = k
36 x = xbaru
```

Pada baris ke-11 saya tetapkan nilai epsilon sebesar 0,0001. Sementara baris ke-10, dimana itermaks saya batasi hingga 1000 kali iterasi. Akan tetapi dengan adanya baris ke-30, maka jika norm2(xselisih) lebih kecil nilainya dari nilai epsilon yang dinyatakan pada baris ke-11, proses iterasi akan dihentikan. Sementara, statemen baris ke-35 sengaja saya tambahkan hanya untuk sekedar mengetahui berapa kali komputer kita melakukan proses iterasi. Dengan nilai epsilon 0,0001, proses iterasi akan dihentikan pada iterasi yang ke-10. Jadi, walaupun itermaks telah ditentukan yaitu 1000, komputer hanya melakukan proses iterasi sampai iterasi yang ke-10 saja.

6.3.3 Fungsi eksternal iterasi Jacobi

Fungsi eksternal metode iterasi Jacobi dapat diambil dari script yang terakhir di atas adalah

```
1  function [k,xbaru] = ijcb(A,b,xlama,itermaks,epsilon)
2
3  [m,n] = size(A);
4
5  % --- membuat matrik J dan vektor u ---
6  J = zeros(n);
7  for p = 1:n
8     for k = 1:n
9        if (k ~= p)
10          J(p,k) = -A(p,k)/A(p,p);
11       end
12    end
13    u(p,1) = b(p,1)/A(p,p);
14 end
15
```

```
% --- proses iterasi jacobi ---
for k = 1:itermaks
   xbaru = J*xlama + u;
   xselisih = xbaru - xlama;
   if (norm2(xselisih) < epsilon)
      break;
   end
   xlama = xbaru;
end
```

Dengan fungsi eksternal ini, maka untuk menyelesaikan suatu sistem persamaan linear, anda dapat menyusun program sederhana. Contohnya adalah

```
clear all
clc

A = [ 10 -1 2 0;
   -1 11 -1 3;
   2 -1 10 -1;
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

xlama = [ 0 ; 0 ; 0 ; 0 ];  %nilai awal
itermaks = 1000;   % iterasi maksimum
epsilon = 0.0001;   % stopping criteria

[k,xbaru] = iterjacobi(A,b,xlama,itermaks,epsilon);
x = xbaru
iterasi = k
```

Demikianlah penjelasan tentang metode iterasi Jacobi dilengkapi dengan cara membuat scriptnya. Sebagai catatan, metode iterasi Jacobi ini selalu sukses mencapai solusi hanya jika matrik A memiliki pola diagonal dominan dimana nilai elemen-elemen diagonal harus lebih besar dibandingkan nilai elemen setiap barisnya. Sekarang mari kita beralih ke metode iterasi Gauss-Seidel.

6.4 Iterasi Gauss-Seidel

Metode Iterasi Gauss-Seidel hampir sama dengan metode Iterasi Jacobi. Perbedaannya hanya terletak pada penggunaan nilai elemen vektor **xbaru** yang langsung digunakan pada persamaan dibawahnya. Untuk lebih jelasnya, perhatikan sistem persamaan linear berikut, yang diturunkan dari contoh terdahulu

$$\begin{aligned}
x_1^{baru} &= \frac{1}{10}x_2^{lama} - \frac{2}{10}x_3^{lama} + \frac{6}{10} \\
x_2^{baru} &= \frac{1}{11}x_1^{baru} + \frac{1}{11}x_3^{lama} - \frac{3}{11}x_4^{lama} + \frac{25}{11} \\
x_3^{baru} &= -\frac{2}{10}x_1^{baru} + \frac{1}{10}x_2^{baru} + \frac{1}{10}x_4^{lama} - \frac{11}{10} \\
x_4^{baru} &= -\frac{3}{8}x_2^{baru} + \frac{1}{8}x_3^{baru} + \frac{15}{8}
\end{aligned}$$

Pada baris pertama, x_1^{baru} dihitung berdasarkan x_2^{lama} dan x_3^{lama}. Kemudian x_1^{baru} tersebut langsung dipakai pada baris kedua untuk menghitung x_2^{baru}. Selanjutnya x_1^{baru} dan x_2^{baru} digunakan pada baris ketiga untuk mendapatkan x_3^{baru}. Begitu seterusnya hingga x_4^{baru} pun diperoleh pada baris keempat. Sistem persamaan tersebut dapat dinyatakan dalam indeks *k* seperti dibawah ini dimana *k* adalah jumlah iterasi.

$$\begin{aligned}
x_1^{(k)} &= \frac{1}{10}x_2^{(k-1)} - \frac{2}{10}x_3^{(k-1)} + \frac{6}{10} \\
x_2^{(k)} &= \frac{1}{11}x_1^{(k)} + \frac{1}{11}x_3^{(k-1)} - \frac{3}{11}x_4^{(k-1)} + \frac{25}{11} \\
x_3^{(k)} &= -\frac{2}{10}x_1^{(k)} + \frac{1}{10}x_2^{(k)} + \frac{1}{10}x_4^{(k-1)} - \frac{11}{10} \\
x_4^{(k)} &= -\frac{3}{8}x_2^{(k)} + \frac{1}{8}x_3^{(k)} + \frac{15}{8}
\end{aligned}$$

Misalnya kita tentukan nilai-nilai awal $\mathbf{x}^{(0)}$ sebagai berikut $x_1^{(0)} = 0$, $x_2^{(0)} = 0$, $x_3^{(0)} = 0$ dan $x_4^{(0)} = 0$. Atau dinyatakan seperti ini $\mathbf{x}^{(0)} = (0; 0; 0; 0)^t$. Maka pada $k = 1$ kita akan memperoleh nilai-nilai $\mathbf{x}^{(1)}$ sebagai berikut

$$\begin{aligned}
x_1^{(1)} &= 0,6000 \\
x_2^{(1)} &= 2,3272 \\
x_3^{(1)} &= -0,9873 \\
x_4^{(1)} &= 0,8789
\end{aligned}$$

Lalu proses perhitungan diulangi lagi dengan $k = 2$. Begitu seterusnya proses ini diulang-ulang lagi untuk nilai-nilai k berikutnya sampai $\mathbf{x}^{(k)}$ mendekati solusi yang sesungguhnya, yaitu

$$\mathbf{x} = (1; 2; -1; 1)^t$$

Marilah kita amati hasil seluruh iterasi. Tabel di bawah ini menampilkan hasil perhitungan hingga iterasi yang ke-5. Kita bisa saksikan bahwa dibandingkan dengan iterasi Jacobi, problem sistem persamaan linear yang sama, bisa diselesaikan oleh metode iterasi Gauss-Seidel hanya dalam 5 kali iterasi. Dari kasus ini, bisa kita simpulkan bahwa iterasi Gauss-Seidel bek-

Tabel 6.3: Hasil Iterasi Gauss-Seidel

k	0	1	2	3	4	5
$x_1^{(k)}$	0,0000	0,6000	1,030	1,0065	1,0009	1,0001
$x_2^{(k)}$	0,0000	2,3272	2,037	2,0036	2,0003	2,0000
$x_3^{(k)}$	0,0000	-0,9873	-1,014	-1,0025	-1,0003	-1,0000
$x_4^{(k)}$	0,0000	0,8789	0,9844	0,9983	0,9999	1,0000

erja lebih efektif dibandingkan iterasi Jacobi. Ya.., memang secara umum demikian, akan tetapi ternyata ditemukan kondisi yang sebaliknya pada kasus-kasus yang lain.

6.4.1 *Script* iterasi Gauss-Seidel

Pembuatan script iterasi Gauss-Seidel dimulai dari sistem persamaan linear yang telah dibahas di atas, yaitu

$$\begin{aligned}
x_1^{baru} &= \frac{1}{10}x_2^{lama} - \frac{2}{10}x_3^{lama} + \frac{6}{10} \\
x_2^{baru} &= \frac{1}{11}x_1^{baru} + \frac{1}{11}x_3^{lama} - \frac{3}{11}x_4^{lama} + \frac{25}{11} \\
x_3^{baru} &= -\frac{2}{10}x_1^{baru} + \frac{1}{10}x_2^{baru} + \frac{1}{10}x_4^{lama} - \frac{11}{10} \\
x_4^{baru} &= -\frac{3}{8}x_2^{baru} + \frac{1}{8}x_3^{baru} + \frac{15}{8}
\end{aligned}$$

Pada pembahasan iterasi Jacobi, saya telah membuat matrik J berisi konstanta yang menemani variabel x. Matrik J ini akan saya gunakan lagi untuk menyusun script metode iterasi Gauss-Seidel

$$J = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix}$$

Kemudian matrik J dipecah menjadi matrik L dan matrik U, dimana J = L + U

$$\begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{11} & 0 & 0 & 0 \\ -\frac{2}{10} & \frac{1}{10} & 0 & 0 \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} + \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ 0 & 0 & \frac{1}{11} & -\frac{3}{11} \\ 0 & 0 & 0 & \frac{1}{10} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Sampai disini saya nyatakan matrik **L**, matrik **U** dan vektor *u* sebagai berikut

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{11} & 0 & 0 & 0 \\ -\frac{2}{10} & \frac{1}{10} & 00 & \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \qquad U = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ 0 & 0 & \frac{1}{11} & -\frac{3}{11} \\ 0 & 0 & 0 & \frac{1}{10} \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad u = \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

Karena matrik L dan U berasal dari matrik J, maka pembuatan script iterasi Gauss-Seidel akan saya mulai dari script perhitungan matrik J yang telah dibuat sebelumnya. Inilah script untuk membuat matrik J,

```
1 clear all
2 clc
3
4 A = [ 10 -1 2 0;
5    -1 11 -1 3;
6    2 -1 10 -1;
7    0 3 -1 8];
8
```

```
b = [ 6 ; 25 ; -11 ; 15 ];

dim = size(A);
n = dim(1);

%---- Perhitungan matrik J dan vektor u-----
for k = 1:n
   J(k,k) = 0;
end

for p = 1:n-1
   for k = 1:n
      if k == p
         k = k+1;
      end
      J(p,k) = -A(p,k)/A(p,p);
   end
   u(p,1) = b(p,1)/A(p,p);
end

for k = 1:n-1
   J(n,k) = -A(n,k)/A(n,n);
end
u(n,1) = b(n,1)/A(n,n);
```

Untuk memperoleh matrik L, pertama-tama matrik J dicopy ke matrik L. Kemudian seluruh elemen segitiga di atas elemen diagonal diganti dengan angka nol. Proses ini dilakukan mulai dari baris ke-34 hingga ke-43.

```
clear all
clc

A = [ 10 -1 2 0;
   -1 11 -1 3;
   2 -1 10 -1;
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

dim = size(A);
n = dim(1);

%---- Perhitungan matrik J dan vektor u-----
for k = 1:n
   J(k,k) = 0;
end

for p = 1:n-1
   for k = 1:n
      if k == p
         k = k+1;
      end
      J(p,k) = -A(p,k)/A(p,p);
   end
   u(p,1) = b(p,1)/A(p,p);
end

```

```
29 for k = 1:n-1
30    J(n,k) = -A(n,k)/A(n,n);
31 end
32 u(n,1) = b(n,1)/A(n,n);
33 %----------------------------------------
34 L = J;     % matrik J dicopy ke matrik L
35 for k = 2:4
36    L(1,k) = 0;
37 end
38 for k = 3:4
39    L(2,k) = 0;
40 end
41 for k = 4:4
42    L(3,k) = 0;
43 end
```

Proses perhitungan mulai dari baris ke-35 hingga ke-43 akan disederhanakan dengan langkah-langkah berikut. Saya munculkan indeks p,

```
L = J;     % matrik J dicopy ke matrik L
p = 1;
for k = 2:4
   L(p,k) = 0;
end
p = 2;
for k = 3:4
   L(p,k) = 0;
end
p = 3;
for k = 4:4
   L(p,k) = 0;
end
```

Dengan adanya indeks p, bagian looping dapat dimodifikasi menjadi

```
L = J;     % matrik J dicopy ke matrik L
p = 1;
for k = p+1:4
   L(p,k) = 0;
end
p = 2;
for k = p+1:4
   L(p,k) = 0;
end
p = 3;
for k = p+1:4
   L(p,k) = 0;
end
```

Kemudian, berdasarkan indeks p, dibuatlah proses looping,

```
L = J;     % matrik J dicopy ke matrik L
for p = 1:3
   for k = p+1:4
      L(p,k) = 0;
   end
end
```

Selanjutnya, angka 3 dan 4 dapat diganti dengan variabel n agar bisa digabung dengan script utamanya. Perhatikan baris ke-35 dan ke-36 pada script berikut

```
1  clear all
2  clc
3
4  A = [ 10 -1 2 0;
5     -1 11 -1 3;
6     2 -1 10 -1;
7     0 3 -1 8];
8
9  b = [ 6 ; 25 ; -11 ; 15 ];
10
11 dim = size(A);
12 n = dim(1);
13
14 %---- Perhitungan matrik J dan vektor u-----
15 for k = 1:n
16    J(k,k) = 0;
17 end
18
19 for p = 1:n-1
20    for k = 1:n
21       if k == p
22          k = k+1;
23       end
24       J(p,k) = -A(p,k)/A(p,p);
25    end
26    u(p,1) = b(p,1)/A(p,p);
27 end
28
29 for k = 1:n-1
30    J(n,k) = -A(n,k)/A(n,n);
31 end
32 u(n,1) = b(n,1)/A(n,n);
33 %----------------------------------------
34 L = J;     % matrik J dicopy ke matrik L
35 for p = 1:n-1
36    for k = p+1:n
37       L(p,k) = 0;
38    end
39 end
```

OK, dengan demikian matrik L telah terbentuk dan tersimpan di memory komputer. Sekarang kita akan membentuk matrik U. Prosesnya sama seperti saat pembentukan matrik L, yaitu dimulai dengan mencopy matrik J ke dalam matrik U. Perhatikan mulai dari baris ke-41 berikut ini,

```
1  clear all
2  clc
3
4  A = [ 10 -1 2 0;
5     -1 11 -1 3;
6     2 -1 10 -1;
7     0 3 -1 8];
8
```

```
b = [ 6 ; 25 ; -11 ; 15 ];

dim = size(A);
n = dim(1);

%---- Perhitungan matrik J dan vektor u-----
for k = 1:n
   J(k,k) = 0;
end

for p = 1:n-1
   for k = 1:n
      if k == p
         k = k+1;
      end
      J(p,k) = -A(p,k)/A(p,p);
   end
   u(p,1) = b(p,1)/A(p,p);
end

for k = 1:n-1
   J(n,k) = -A(n,k)/A(n,n);
end
u(n,1) = b(n,1)/A(n,n);
%----------------------------------------
L = J;     % matrik J dicopy ke matrik L
for p = 1:n-1
   for k = p+1:n
      L(p,k) = 0;
   end
end
%----------------------------------------
U = J;     % matrik J dicopy ke matrik U
for k = 2:4
   U(k,1) = 0;
end
for k = 3:4
   U(k,2) = 0;
end
for k = 4:4
   U(k,3) = 0;
end
```

Kemudian, indeks p dimunculkan mulai diantara baris ke-42 hingga ke-50,

```
U = J;     % matrik J dicopy ke matrik U
p = 1;
for k = p+1:4
   U(k,p) = 0;
end
p = 2;
for k = p+1:4
   U(k,p) = 0;
end
p = 3;
for k = p+1:4
   U(k,p) = 0;
end
```

Selanjutnya, berdasarkan indeks p dibuatlah proses looping yang baru

```
U = J;     % matrik J dicopy ke matrik U
for p = 1:3
   for k = p+1:4
      U(k,p) = 0;
   end
end
```

Akhirnya, script ini digabungkan ke script utamanya setelah mengganti angkan 3 dan 4 dengan variabel n.

```
clear all
clc

A = [ 10 -1 2 0;
   -1 11 -1 3;
   2 -1 10 -1;
   0 3 -1 8];

b = [ 6 ; 25 ; -11 ; 15 ];

dim = size(A);
n = dim(1);

%---- Perhitungan matrik J dan vektor u-----
for k = 1:n
   J(k,k) = 0;
end

for p = 1:n-1
   for k = 1:n
      if k == p
         k = k+1;
      end
      J(p,k) = -A(p,k)/A(p,p);
   end
   u(p,1) = b(p,1)/A(p,p);
end

for k = 1:n-1
   J(n,k) = -A(n,k)/A(n,n);
end
u(n,1) = b(n,1)/A(n,n);
%----------------------------------------
L = J;     % matrik J dicopy ke matrik L
for p = 1:n-1
   for k = p+1:n
      L(p,k) = 0;
   end
end
%----------------------------------------
U = J;     % matrik J dicopy ke matrik U
for p = 1:n-1
   for k = p+1:n
      U(k,p) = 0;
   end
end
```

Secara umum, script iterasi Gauss-Seidel yang saya tuliskan disini hampir sama dengan iterasi Jacobi. Perbedaan kecil-nya terletak pada bagian *nilai update*, dimana elemen **xbaru** hasil perhitungan dilibatkan langsung untuk menghitung elemen **xbaru** selanjutnya.

```
clear all
clc

%----nilai awal-----------
xlama(1,1)=0;
xlama(2,1)=0;
xlama(3,1)=0;
xlama(4,1)=0;
xlama

n=4          %jumlah elemen vektor
itermaks=10  %jumlah iterasi maksimal
sc=0.001     %stopping-criteria

for i=1:itermaks
 %------nilai update-------------
  xbaru(1,1)=(1/10)*xlama(2,1)-(2/10)*xlama(3,1)+(6/10);
  xbaru(2,1)=(1/11)*xbaru(1,1)+(1/11)*xlama(3,1)-(3/11)*xlama(4,1)+(25/11);
  xbaru(3,1)=-(2/10)*xbaru(1,1)+(1/10)*xbaru(2,1)+(1/10)*xlama(4,1)-(11/10);
  xbaru(4,1)=-(3/8)*xbaru(2,1)+(1/8)*xbaru(3,1)+(15/8);
  xbaru

 %------norm selisih-------------
 s=0;
 for i=1:n
  s=s+(xbaru(i,1)-xlama(i,1))^2;
 end
 epsilon=sqrt(s)

 %------memeriksa stopping criteria, sc--------
 if epsilon<sc
  break
 end

 xlama=xbaru; %xbaru dijadikan xlama untuk iterasi berikutnya
end
```

Perumusan metode Iterasi Gauss-Seidel dapat dinyatakan sebagai berikut:

$$x_i^{(k)} = \frac{-\sum_{j=1}^{i-1}\left(a_{ij}x_j^{(k)}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j^{(k-1)}\right) + b_i}{a_{ii}} \tag{6.7}$$

dimana i=1,2,3,...,n.

6.4.2 Algoritma

- Langkah 1: Tentukan k=1
- Langkah 2: Ketika ($k \leq N$) lakukan Langkah 3-6

  - Langkah 3: Untuk i=1,...,n, hitunglah

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

  - Langkah 4: Jika $\|\mathbf{x} - \mathbf{XO}\| < \epsilon$, maka keluarkan OUTPUT $(x_1, ..., x_n)$ lalu STOP
  - Langkah 5: Tentukan k=k+1
  - Langkah 6: Untuk i=1,...n, tentukan $XO_i = x_i$

- Langkah 7: OUTPUT ('Iterasi maksimum telah terlampaui') lalu STOP

6.4.3 *Script* iterasi Gauss-Seidel dalam Fortran

```
      IMPLICIT NONE
      DIMENSION A(10,10),B(10),X(10),XO(10)
      REAL A,B,X,XO,EPS,NORM,S1,S2
      INTEGER N,I,J,K,ITMAX
      WRITE(*,*)
      WRITE(*,*) '==> ITERASI GAUSS-SEIDEL UNTUK SISTEM LINEAR <=='
      WRITE(*,*)
      WRITE (*,'(1X,A)') 'JUMLAH PERSAMAAN ? '
      READ (*,*) N
      WRITE (*,*) 'MASUKAN ELEMEN-ELEMEN MATRIK A DAN VEKTOR B'
      DO 52 I = 1,N
        DO 62 J = 1,N
          WRITE (*,'(1X,A,I2,A,I2,A)') 'A(',I,',',J,') = '
          READ (*,*) A(I,J)
62      CONTINUE
        WRITE (*,'(1X,A,I2,A)') 'B(',I,') ? '
        READ (*,*) B(I)
        WRITE (*,*)
52    CONTINUE
      WRITE (*,'(1X,A)') 'JUMLAH ITERASI MAKSIMUM ? '
      READ (*,*) ITMAX
      WRITE (*,'(1X,A)') 'NILAI EPSILON ATAU TOLERANSI ? '
      READ (*,*) EPS
      WRITE (*,*) 'MASUKAN NILAI AWAL UNTUK XO'
      DO 72 I = 1,N
        WRITE (*,'(1X,A,I2,A)') 'XO(',I,') ? '
        READ (*,*) XO(I)
72    CONTINUE
      WRITE (*,*)
C     MENAMPILKAN MATRIK A
      WRITE (*,'(1X,A)') 'MATRIK A:'
      DO 110 I = 1,N
        WRITE (*,6) (A(I,J),J=1,N)
110   CONTINUE
      WRITE (*,*)
C     MENAMPILKAN VEKTOR B
      WRITE (*,'(1X,A)') 'VEKTOR B:'
      DO 111 I = 1,N
        WRITE (*,6) B(I)
111   CONTINUE
      WRITE (*,*)
C     LANGKAH 1
```

```
      K = 1
C     LANGKAH 2
100   IF(K.GT.ITMAX) GOTO 200
C     LANGKAH 3
      DO 10 I = 1,N
        S1 = 0.0
        DO 20 J=I+1,N
          S1 = S1-A(I,J)*XO(J)
20      CONTINUE
        S2 = 0.0
        DO 23 J=1,I-1
          S2 = S2-A(I,J)*X(J)
23      CONTINUE
        X(I) = (S2+S1+B(I))/A(I,I)
10    CONTINUE
C     SAYA PILIH NORM-2. ANDA BOLEH PAKAI NORM YANG LAIN!
      NORM = 0.0
      DO 40 I=1,N
        NORM = NORM + (X(I)-XO(I))*(X(I)-XO(I))
40    CONTINUE
      NORM = SQRT(NORM)
      WRITE(*,'(1X,A,I3)') 'ITERASI KE-', K
      WRITE(*,'(1X,A,F14.8)') 'NORM-2 = ', NORM
      WRITE(*,'(1X,A,I3,A,F14.8)') ('X(',I,') = ', X(I),I=1,N)
      WRITE(*,*)
C     LANGKAH 4
      IF(NORM.LE.EPS) THEN
        WRITE(*,7) K,NORM
        GOTO 400
      END IF
C     LANGKAH 5
      K = K+1
C     LANGKAH 6
      DO 30 I=1,N
        XO(I) = X(I)
30    CONTINUE
      GOTO 100
C     LANGKAH 7
200   CONTINUE
      WRITE(*,9)
400   STOP

5     FORMAT(1X,I3)
6     FORMAT(1X,(6(1X,F14.8)))
7     FORMAT(1X,'KONVERGEN PADA ITERASI YANG KE- ',I3,
     *' , NORM= ',F14.8)
9     FORMAT(1X,'MELEBIHI BATAS MAKSIMUM ITERASI')
      END
```

6.5 Iterasi dengan Relaksasi

Metode Iterasi Relaksasi (*Relaxation method*) dinyatakan dengan rumus berikut:

$$x_i^{(k)} = (1-\omega)\, x_i^{(k-1)} + \frac{\omega}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right] \tag{6.8}$$

dimana i=1,2,3,...,n.

Untuk lebih jelasnya, marilah kita perhatikan contoh berikut, diketahui sistem persamaan linear $\mathbf{Ax} = \mathbf{b}$ yaitu

$$\begin{aligned} 4x_1 + 3x_2 + &= 24 \\ 3x_1 + 4x_2 - x_3 &= 30 \\ -x_2 + 4x_3 &= -24 \end{aligned}$$

memiliki solusi $(3, 4, -5)^t$. Metode Gauss-Seidel dan Relaksasi dengan $\omega = 1,25$ akan digunakan untuk menyelesaikan sistem persamaan linear di atas dengan $\mathbf{x}^{(0)} = (1, 1, 1)^t$. Untuk setiap nilai $k = 1, 2, 3, ...$, persamaan Gauss-Seidelnya adalah

$$\begin{aligned} x_1^{(k)} &= -0,75x_2^{(k-1)} + 6 \\ x_2^{(k)} &= -0,75x_1^{(k)} + 0,25x_3^{(k-1)} + 7,5 \\ x_3^{(k)} &= 0,25x_2^{(k)} - 6 \end{aligned}$$

Sedangkan persamaan untuk metode Relaksasi dengan $\omega = 1,25$ adalah

$$\begin{aligned} x_1^{(k)} &= -0,25x_1^{(k-1)} - 0,9375x_2^{(k-1)} + 7,5 \\ x_2^{(k)} &= -0,9375x_1^{(k)} - 0,25x_2^{(k-1)} + 0,3125x_3^{(k-1)} + 9,375 \\ x_3^{(k)} &= 0,3125x_2^{(k)} - 0,25x_3^{(k-1)} - 7,5 \end{aligned}$$

Tabel berikut ini menampilkan perhitungan dari masing-masing metode hingga iterasi ke-7.

Tabel 6.4: Hasil perhitungan iterasi Gauss-Seidel

k	0	1	2	3	4	5	6	7
$x_1^{(k)}$	1	5,2500	3,1406	3,0879	3,0549	3,0343	3,0215	3,0134
$x_2^{(k)}$	1	3,8125	3,8828	3,9267	3,9542	3,9714	3,9821	3,9888
$x_3^{(k)}$	1	-5,0468	-5,0293	-5,0183	-5,0114	-5,0072	-5,0044	-5,0028

Tabel 6.5: Hasil perhitungan iterasi Relaksasi dengan $\omega = 1,25$

k	0	1	2	3	4	5	6	7
$x_1^{(k)}$	1	6,3125	2,6223	3,1333	2,9570	3,0037	2,9963	3,0000
$x_2^{(k)}$	1	3,5195	3,9585	4,0102	4,0075	4,0029	4,0009	4,0002
$x_3^{(k)}$	1	-6,6501	-4,6004	-5,0967	-4,9735	-5,0057	-4,9983	-5,0003

Dari kasus ini, bisa kita simpulkan bahwa iterasi Relaksasi memerlukan proses iterasi yang lebih singkat dibandingkan iterasi Gauss-Seidel. Jadi, pada kasus ini (dan juga secara umum), Relaksasi lebih efektif dibandingkan Gauss-Seidel. Pertanyaannya sekarang, bagaimana menentukan nilai ω optimal?

Metode Relaksasi dengan pilihan nilai ω yang berkisar antara 0 dan 1 disebut metode *under-relaxation*, dimana metode ini berguna agar sistem persamaan linear bisa mencapai kondisi konvergen walaupun sistem tersebut sulit mencapai kondisi konvergen dengan metode Gauss-Seidel. Sementara bila ω nilainya lebih besar dari angka 1, maka disebut metode *successive over-relaxation (SOR)*, yang mana metode ini berguna untuk mengakselerasi atau mempercepat kondisi konvergen dibandingkan dengan Gauss-Seidel. Metode SOR ini juga sangat berguna untuk menyelesaikan sistem persamaan linear yang muncul dari persamaan diferensial-parsial tertentu.

6.5.1 Algoritma Iterasi Relaksasi

- Langkah 1: Tentukan k=1
- Langkah 2: Ketika ($k \leq N$) lakukan Langkah 3-6
  - Langkah 3: Untuk i=1,...,n, hitunglah

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

  - Langkah 4: Jika $\|\mathbf{x} - \mathbf{XO}\| < \epsilon$, maka keluarkan OUTPUT $(x_1, ..., x_n)$ lalu STOP
  - Langkah 5: Tentukan k=k+1
  - Langkah 6: Untuk i=1,...n, tentukan $XO_i = x_i$
- Langkah 7: OUTPUT ('Iterasi maksimum telah terlampaui') lalu STOP

Demikianlah catatan singkat dari saya tentang metode iterasi untuk menyelesaikan problem sistem persamaan linear. Saya cukupkan sementara sampai disini. Insya Allah akan saya sambung lagi dilain waktu. Kalau ada yang mau didiskusikan, silakan hubungi saya melalui email: supri92@gmail.com.

Bab 7

Interpolasi

Objektif :

- Mengenalkan Interpolasi Lagrange
- Mengenalkan Interpolasi Spline-cubic

7.1 Interpolasi Lagrange

Interpolasi Lagrange diterapkan untuk mendapatkan fungsi polinomial $P(x)$ berderajat tertentu yang melewati sejumlah titik data. Misalnya, kita ingin mendapatkan fungsi polinomial berderajat satu yang melewati dua buah titik yaitu (x_0, y_0) dan (x_1, y_1). Langkah pertama yang kita lakukan adalah mendefinisikan fungsi berikut

$$L_0(x) = \frac{x - x_1}{x_0 - x_1}$$

dan

$$L_1(x) = \frac{x - x_0}{x_1 - x_0}$$

kemudian kita definisikan fungsi polinomial sebagai berikut

$$P(x) = L_0(x)y_0 + L_1(x)y_1$$

Jika semua persamaan diatas kita gabungkan, maka akan didapat

$$\begin{aligned} P(x) &= L_0(x)y_0 + L_1(x)y_1 \\ P(x) &= \frac{x - x_1}{x_0 - x_1}y_0 + \frac{x - x_0}{x_1 - x_0}y_1 \end{aligned}$$

dan ketika $x = x_0$

$$P(x_0) = \frac{x_0 - x_1}{x_0 - x_1}y_0 + \frac{x_0 - x_0}{x_1 - x_0}y_1 = y_0$$

dan pada saat $x = x_1$

$$P(x_1) = \frac{x_1 - x_1}{x_0 - x_1} y_0 + \frac{x_1 - x_0}{x_1 - x_0} y_1 = y_1$$

dari contoh ini, kira-kira apa kesimpulan sementara anda? Ya.. kita bisa sepakat bahwa fungsi polinomial

$$P(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 \tag{7.1}$$

benar-benar melewati titik (x_0, y_0) dan (x_1, y_1).

Sekarang mari kita perhatikan lagi contoh lainnya. Misalnya ada tiga titik yaitu (x_0, y_0), (x_1, y_1) dan (x_2, y_2). Tentukanlah fungsi polinomial yang melewati ketiganya! Dengan pola yang sama kita bisa awali langkah pertama yaitu mendefinisikan

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$

lalu

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

dan

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

kemudian kita definisikan fungsi polinomial sebagai berikut

$$P(x) = L_0(x)y_0 + L_1(x)y_1 + L_2(x)y_2$$

Jika semua persamaan diatas kita gabungkan, maka akan didapat fungsi polinomial

$$P(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2$$

Kita uji sebentar. Ketika $x = x_0$

$$P(x_0) = \frac{(x_0 - x_1)(x_0 - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x_0 - x_0)(x_0 - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x_0 - x_0)(x_0 - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2 = y_0$$

pada saat $x = x_1$

$$P(x_1) = \frac{(x_1 - x_1)(x_1 - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x_1 - x_0)(x_1 - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x_1 - x_0)(x_1 - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2 = y_1$$

pada saat $x = x_2$

$$P(x_2) = \frac{(x_2 - x_1)(x_2 - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x_2 - x_0)(x_2 - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x_2 - x_0)(x_2 - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2 = y_2$$

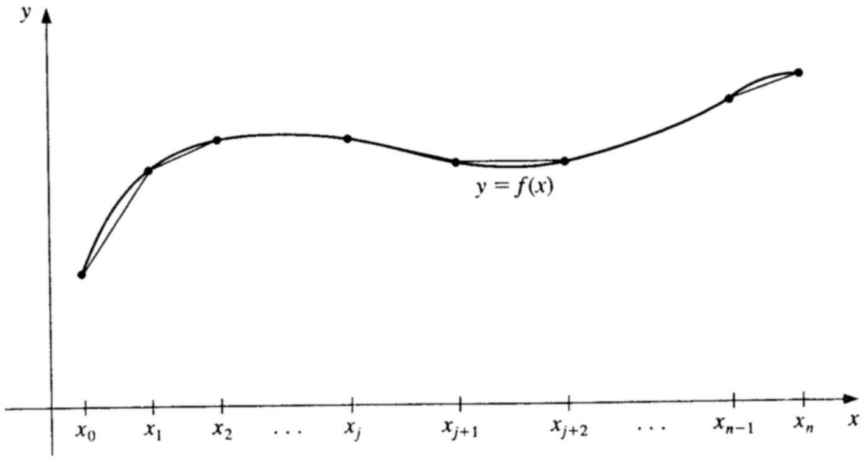
Terbukti bahwa fungsi polonomial

$$P(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2 \qquad (7.2)$$

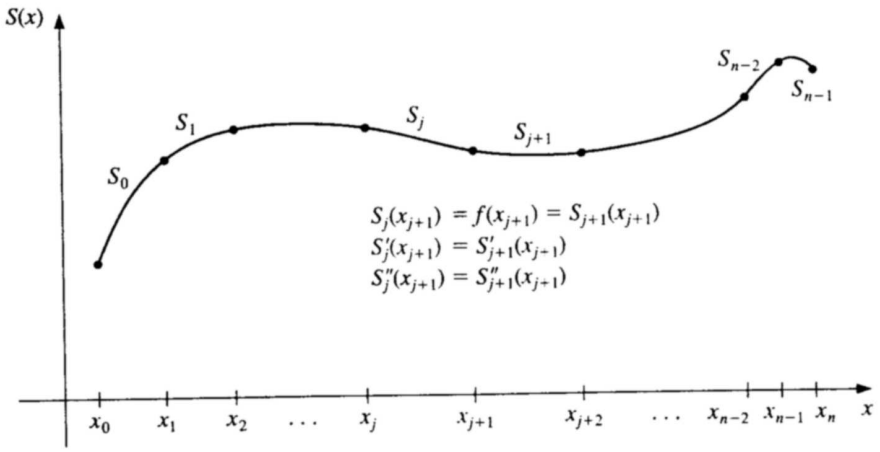
melewati ketiga titik tadi.

Kalau kita bandingkan antara persamaan (7.1) dan persamaan (7.2), terlihat bahwa derajat persamaan (7.2) lebih tinggi dibandingkan dengan derajat persamaan (7.1). Hal ini terlihat dari x^2 pada persamaan (7.2) sementara pada persamaan (7.1) hanya ada x. persamaan (7.2) disebut funsi polinomial berderajat 2, sedangkan persamaan (7.1) disebut fungsi polinomial berderajat 1.

7.2 Interpolasi Cubic Spline



Gambar 7.1: Fungsi $f(x)$ dengan sejumlah titik data



Gambar 7.2: Pendekatan dengan polinomial cubic spline

Diketahui suatu fungsi $f(x)$ (Figure 7.1) yang dibatasi oleh interval a dan b, dan memiliki sejumlah titik data $a = x_0 < x_1 < ... < x_n = b$. Interpolasi cubic spline $S(x)$ adalah sebuah potongan fungsi polinomial kecil-kecil (Figure 7.2) berderajat tiga (*cubic*) yang menghubungkan dua titik data yang bersebelahan dengan ketentuan sebagai berikut:

1. $S_j(x)$ adalah potongan fungsi yang berada pada sub-interval dari x_j hingga x_{j+1} untuk nilai $j = 0, 1, ..., n-1$;
2. $S(x_j) = f(x_j)$, artinya pada setiap titik data (x_j), nilai $f(x_j)$ bersesuaian dengan $S(x_j)$ dimana $j = 0, 1, ..., n$;
3. $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$. Perhatikan titik x_{j+1} pada Figure 7.2. Ya.. tentu saja jika fungsi itu kontinyu, maka titik x_{j+1} menjadi titik sambungan antara S_j dan S_{j+1}.
4. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$, artinya kontinyuitas menuntut turunan pertama dari S_j dan S_{j+1} pada titik x_{j+1} harus bersesuaian.
5. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$, artinya kontinyuitas menuntut turunan kedua dari S_j dan S_{j+1} pada titik x_{j+1} harus bersesuaian juga.
6. Salah satu syarat batas diantara 2 syarat batas x_0 dan x_n berikut ini mesti terpenuhi:
   - $S''(x_0) = S''(x_n) = 0$ ini disebut *natural boundary*
   - $S'(x_0) = f'(x_0)$ dan $S'(x_n) = f'(x_n)$ ini disebut *clamped boundary*

Polinomial cubic spline S (polinomial pangkat 3) untuk suatu fungsi f berdasarkan ketentuan di atas adalah

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \tag{7.3}$$

dimana $j = 0, 1, ..., n-1$. Maka ketika $x = x_j$

$$\begin{aligned} S_j(x_j) &= a_j + b_j(x_j - x_j) + c_j(x_j - x_j)^2 + d_j(x_j - x_j)^3 \\ S_j(x_j) &= a_j = f(x_j) \end{aligned}$$

Itu artinya, a_j selalu jadi pasangan titik data dari x_j. Dengan pola ini maka pasangan titik data x_{j+1} adalah a_{j+1}, konsekuensinya $S(x_{j+1}) = a_{j+1}$. Berdasarkan ketentuan (**3**), yaitu ketika $x = x_{j+1}$ dimasukan ke persamaan (12.7)

$$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3$$

dimana $j = 0, 1, ..., n-2$. Sekarang, kita nyatakan $h_j = x_{j+1} - x_j$, sehingga

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \tag{7.4}$$

Kemudian, turunan pertama dari persamaan (12.7) adalah

$$S'_j(x) = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2$$

ketika $x = x_j$,

$$S'_j(x_j) = b_j + 2c_j(x_j - x_j) + 3d_j(x_j - x_j)^2 = b_j$$

dan ketika $x = x_{j+1}$,

$$b_{j+1} = S'_j(x_{j+1}) = b_j + 2c_j(x_{j+1} - x_j) + 3d_j(x_{j+1} - x_j)^2$$

Ini dapat dinyatakan sebagai

$$b_{j+1} = b_j + 2c_j(x_{j+1} - x_j) + 3d_j(x_{j+1} - x_j)^2$$

dan dinyatakan dalam h_j

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2 \tag{7.5}$$

Berikutnya, kita hitung turunan kedua dari persamaan (12.7)

$$S''_j(x) = 2c_j + 6d_j(x - x_j) \tag{7.6}$$

tapi dengan ketentuan tambahan yaitu $S''(x)/2$, sehingga persamaan ini dimodifikasi menjadi

$$S''_j(x) = c_j + 3d_j(x - x_j)$$

dengan cara yang sama, ketika $x = x_j$

$$S''_j(x_j) = c_j + 3d_j(x_j - x_j) = c_j$$

dan ketika $x = x_{j+1}$

$$\begin{aligned} c_{j+1} = S''_j(x_{j+1}) &= c_j + 3d_j(x_{j+1} - x_j) \\ c_{j+1} &= c_j + 3d_j h_j \end{aligned} \tag{7.7}$$

dan d_j bisa dinyatakan

$$d_j = \frac{1}{3h_j}(c_{j+1} - c_j)$$

dari sini, persamaan (7.4) dapat ditulis kembali

$$\begin{aligned} a_{j+1} &= a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \\ &= a_j + b_j h_j + c_j h_j^2 + \frac{h_j^2}{3}(c_{j+1} - c_j) \\ &= a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1}) \end{aligned} \tag{7.8}$$

sementara persamaan (7.5) menjadi

$$\begin{aligned} b_{j+1} &= b_j + 2c_j h_j + 3d_j h_j^2 \\ &= b_j + 2c_j h_j + h_j(c_{j+1} - c_j) \\ &= b_j + h_j(c_j + c_{j+1}) \end{aligned} \tag{7.9}$$

Sampai sini masih bisa diikuti, bukan? Selanjutnya, kita coba mendapatkan b_j dari persamaan (7.8)

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}) \tag{7.10}$$

dan untuk b_{j-1}

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j) \tag{7.11}$$

Langkah berikutnya adalah mensubtitusikan persamaan (7.10) dan persamaan (7.11) kedalam persamaan (7.9),

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \tag{7.12}$$

dimana $j = 1, 2, ..., n-1$. Dalam sistem persamaan ini, nilai $\{h_j\}_{j=0}^{n-1}$ dan nilai $\{a_j\}_{j=0}^{n}$ sudah diketahui, sementara nilai $\{c_j\}_{j=0}^{n}$ belum diketahui dan memang nilai inilah yang akan dihitung dari persamaan ini.

Sekarang coba perhatikan ketentuan nomor (**6**), ketika $S''(x_0) = S''(x_n) = 0$, berapakah nilai c_0 dan c_n? Nah, kita bisa evaluasi persamaan (7.6)

$$S''(x_0) = 2c_0 + 6d_0(x_0 - x_0) = 0$$

jelas sekali c_0 harus berharga nol. Demikian halnya dengan c_n harganya harus nol. Jadi untuk *natural boundary*, nilai $c_0 = c_n = 0$.

Persamaan (7.12) dapat dihitung dengan operasi matrik $\mathbf{Ax} = \mathbf{b}$ dimana

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots & \dots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 & \dots & \dots & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \dots & \dots & \dots & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}$$

Sekarang kita beralih ke *clamped boundary* dimana $S'(a) = f'(a)$ dan $S'(b) = f'(b)$. Nah, kita bisa evaluasi persamaan (7.10) dengan $j = 0$, dimana $f'(a) = S'(a) = S'(x_0) = b_0$, sehingga

$$f'(a) = \frac{1}{h_0}(a_1 - a_0) - \frac{h_0}{3}(2c_0 + c_1)$$

konsekuensinya,

$$2h_0c_0 + h_0c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \tag{7.13}$$

Sementara pada $x_n = b_n$ dengan persamaan (7.9)

$$f'(b) = b_n = b_{n-1} + h_{n-1}(c_{n-1} + c_n)$$

sedangkan b_{n-1} bisa didapat dari persamaan (7.11) dengan $j = n - 1$

$$b_{n-1} = \frac{1}{h_{n-1}}(a_n - a_{n-1}) - \frac{h_{n-1}}{3}(2c_{n-1}j + c_n)$$

Jadi

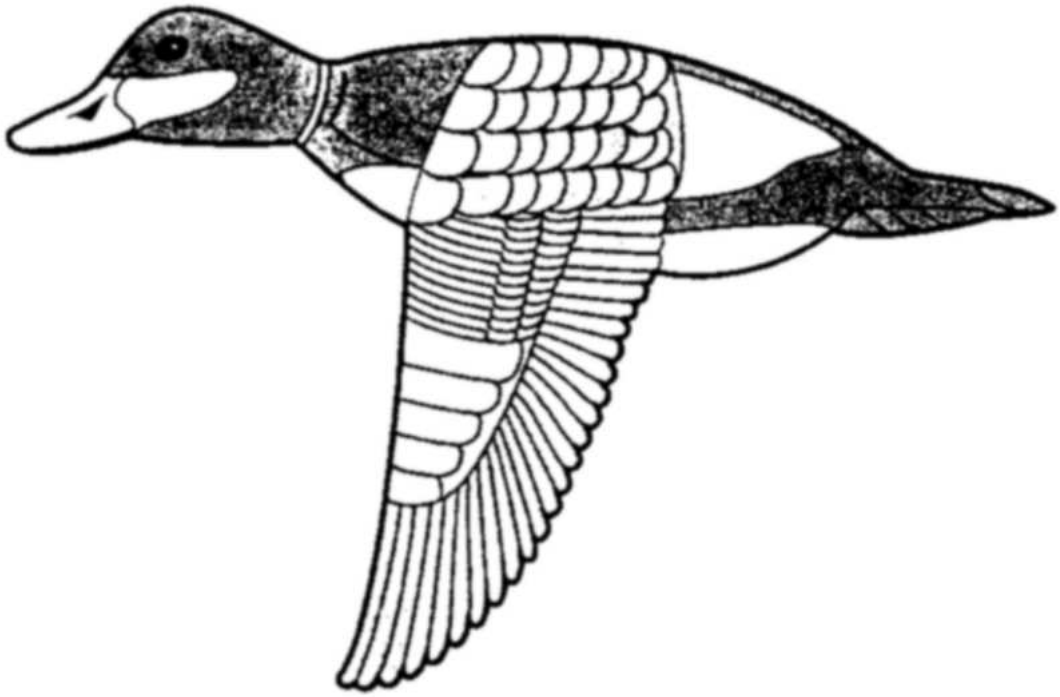
$$\begin{aligned} f'(b) &= \frac{1}{h_{n-1}}(a_n - a_{n-1}) - \frac{h_{n-1}}{3}(2c_{n-1}j + c_n) + h_{n-1}(c_{n-1} + c_n) \\ &= \frac{1}{h_{n-1}}(a_n - a_{n-1} + \frac{h_{n-1}}{3}(c_{n-1}j + 2c_n) \end{aligned}$$

dan akhirnya kita peroleh

$$h_{n-1}c_{n-1} + 2h_{n-1}C_n = 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \tag{7.14}$$

Persamaan (7.13) dan persamaan (7.14) ditambah persamaan (7.12 membentuk operasi matrik $\mathbf{Ax} = \mathbf{b}$ dimana

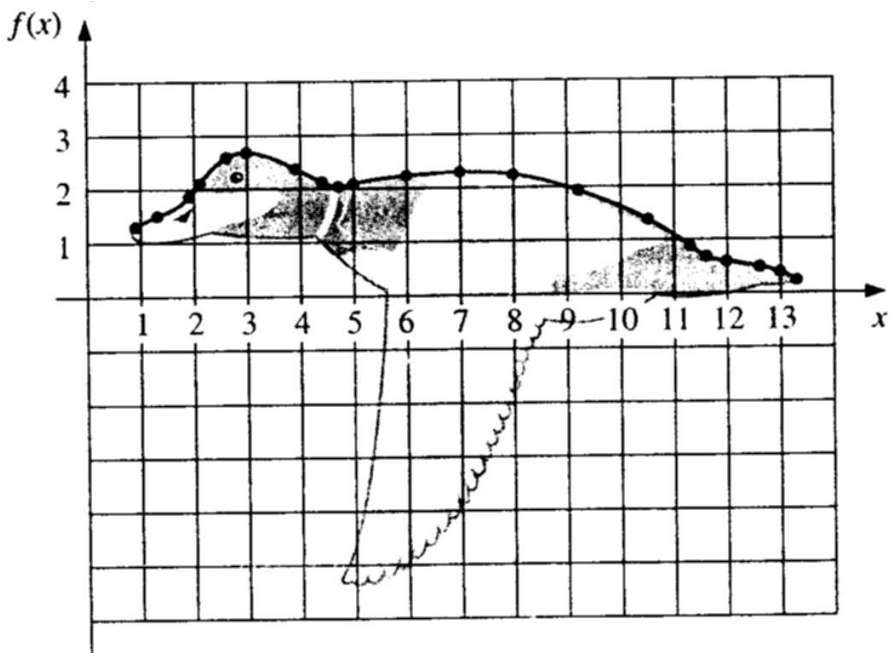
$$\mathbf{A} = \begin{bmatrix} 2h_0 & h_0 & 0 & \dots & \dots & \dots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & \dots & \dots & 0 \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \dots & \dots & \dots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$



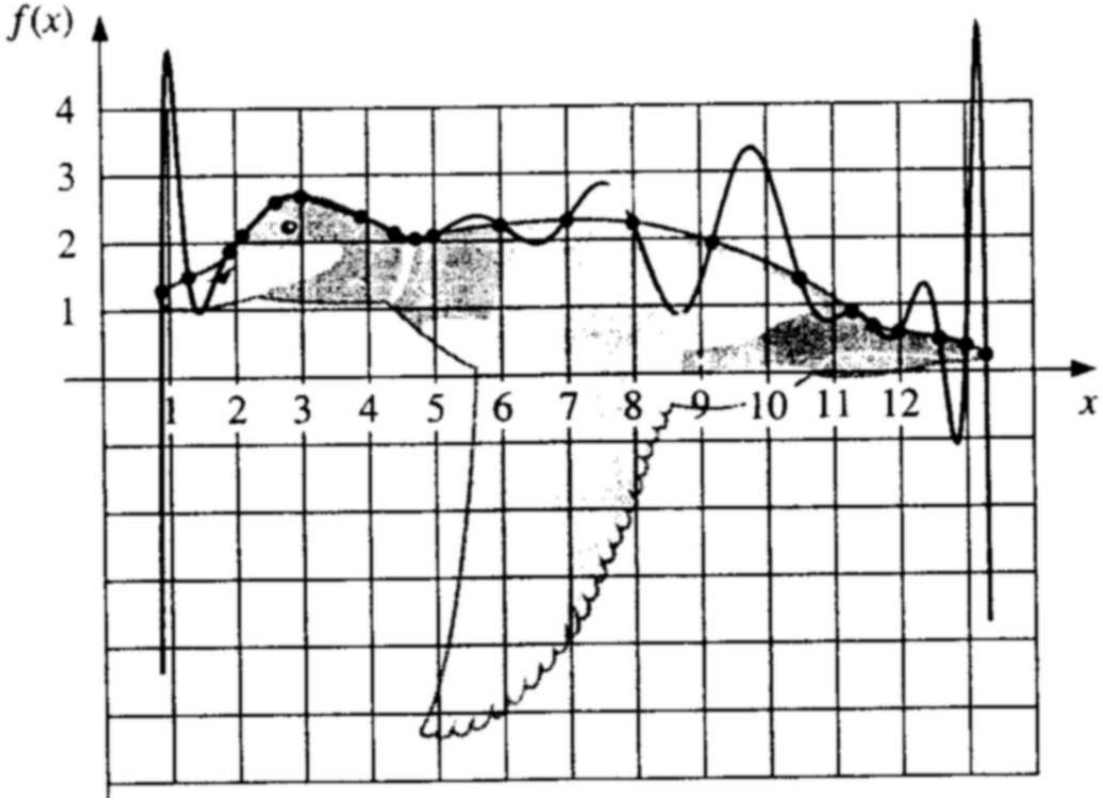
Gambar 7.3: Profil suatu object

$$\mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix}$$

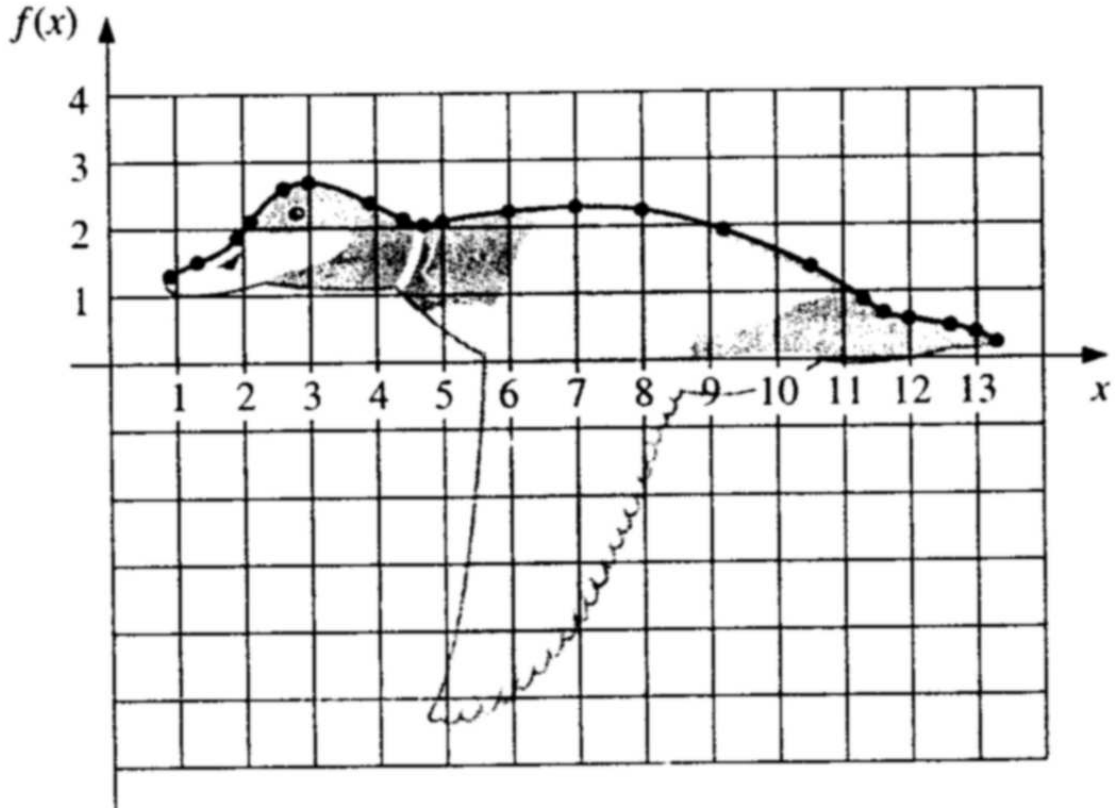


Gambar 7.4: Sampling titik data



Gambar 7.5: Hasil interpolasi cubic spline

j	x_j	a_j	b_j	c_j	d_j
0	0,9	1,3	5,4	0,00	-0,25
1	1,3	1,5	0,42	-0,30	0,95
2	1,9	1,85	1,09	1,41	-2,96
3	2,1	2,1	1,29	-0,37	-0,45
4	2,6	2,6	0,59	-1,04	0,45
5	3,0	2,7	-0,02	-0,50	0,17
6	3,9	2,4	-0,5	-0,03	0,08
7	4,4	2,15	-0,48	0,08	1,31
8	4,7	2,05	-0,07	1,27	-1,58
9	5,0	2,1	0,26	-0,16	0,04
10	6,0	2,25	0,08	-0,03	0,00
11	7,0	2,3	0,01	-0,04	-0,02
12	8,0	2,25	-0,14	-0,11	0,02
13	9,2	1,95	-0,34	-0,05	-0,01
14	10,5	1,4	-0,53	-0,1	-0,02
15	11,3	0,9	-0,73	-0,15	1,21
16	11,6	0,7	-0,49	0,94	-0,84
17	12,0	0,6	-0,14	-0,06	0,04
18	12,6	0,5	-0,18	0	-0,45
19	13,0	0,4	-0,39	-0,54	0,60
20	13,3	0,25			



Gambar 7.6: Hasil interpolasi lagrange

Bab 8

Diferensial Numerik

Objektif :

- ▷ Mengenalkan metode Euler
- ▷ Mengenalkan metode Runge Kutta orde 4
- ▷ Mengenalkan metode *Finite Difference*
- ▷ Mengenalkan Persamaan Diferensial Parsial Eliptik
- ▷ Mengenalkan Persamaan Diferensial Parsial Hiperbolik
- ▷ Mengenalkan Persamaan Diferensial Parsial Parabolik

8.1 Metode Euler

Suatu persamaan diferensial ($\frac{dy}{dt}$) dinyatakan dalam fungsi $f(t, y)$, dimana $y(t)$ adalah persamaan asalnya

$$\frac{dy}{dt} = f(t, y), \qquad a \leq t \leq b, \qquad y(a) = \alpha \tag{8.1}$$

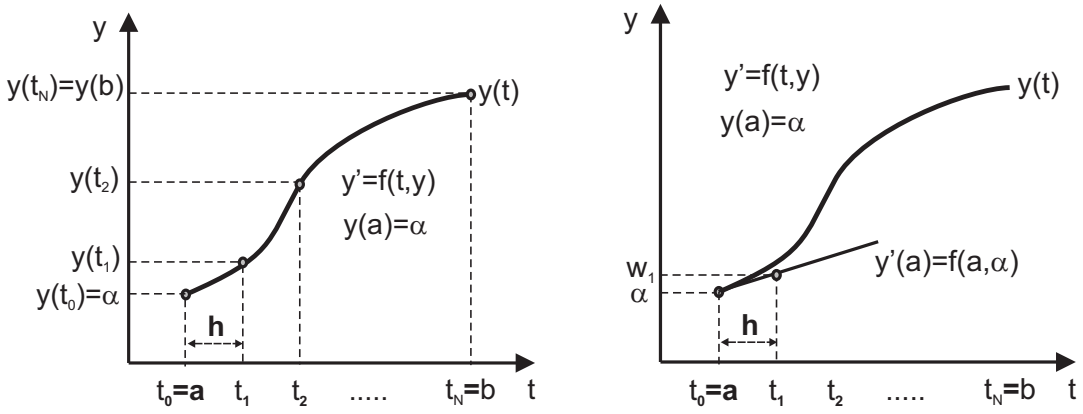
Nilai t dibatasi dari a hingga ke b. Sementara, syarat awal telah diketahui yaitu pada saat $t = a$ maka y bernilai α. Akan tetapi kita sama sekali tidak tahu bentuk formulasi persamaan asalnya $y(t)$. Gambar 8.1 memperlihatkan kurva persamaan asal $y(t)$ yang tidak diketahui bentuk formulasinya. Tantangannya adalah bagaimana kita bisa mendapatkan solusi persamaan diferensial untuk setiap nilai $y(t)$ yang t-nya terletak diantara a dan b ?

Tahap awal solusi pendekatan numerik adalah dengan menentukan point-point dalam jarak yang sama di dalam interval [a,b]. Jarak antar point dirumuskan sebagai

$$h = \frac{b - a}{N} \tag{8.2}$$

dengan N adalah bilangan *integer* positif. Nilai h ini juga dikenal dengan nama ***step size***. Selanjutnya nilai t diantara a dan b ditentukan berdasarkan

$$t_i = a + ih, \qquad i = 0, 1, 2, ..., N \tag{8.3}$$



Gambar 8.1: *Kiri*: Kurva $y(t)$ dengan pasangan titik absis dan ordinat dimana jarak titik absis sebesar h. Pasangan t_1 adalah $y(t_1)$, pasangan t_2 adalah $y(t_2)$, begitu seterusnya. *Kanan*: Garis singgung yang menyinggung kurva $y(t)$ pada t=a, kemudian berdasarkan garis singgung tersebut, ditentukan pasangan t_1 sebagai w_1. Perhatikan gambar itu sekali lagi! w_1 dan $y(t_1)$ beda tipis alias tidak sama persis.

Metode Euler diturunkan dari deret Taylor. Misalnya, fungsi $y(t)$ adalah fungsi yang kontinyu dan memiliki turunan dalam interval [a,b]. Dalam deret Taylor, fungsi $y(t)$ tersebut dirumuskan sebagai

$$y(t_{i+1}) = y(t_i) + (t_{i+1} - t_i)y'(t_i) + \frac{(t_{i+1} - t_i)^2}{2}y''(\xi_i) \tag{8.4}$$

dengan memasukkan $h = (t_{i+1} - t_i)$, maka

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi_i) \tag{8.5}$$

dan, karena $y(t)$ memenuhi persamaan diferensial (8.1), dimana $y'(t_i)$ tak lain adalah fungsi turunan $f(t_i, y(t_i))$, maka

$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2}y''(\xi_i) \tag{8.6}$$

Metode Euler dibangun dengan pendekatan bahwa suku terakhir dari persamaan (8.6), yang memuat turunan kedua, dapat diabaikan. Disamping itu, pada umumnya, notasi penulisan bagi $y(t_i)$ diganti dengan w_i. Sehingga metode Euler diformulasikan sebagai

$$w_{i+1} = w_i + hf(t_i, w_i) \qquad \text{dengan syarat awal} \quad w_0 = \alpha \tag{8.7}$$

dimana $i = 0, 1, 2, .., N - 1$.

Contoh

Diketahui persamaan diferensial

$$y' = y - t^2 + 1 \quad \text{batas interval:} \quad 0 \le t \le 2 \quad \text{syarat awal:} \quad y(0) = 0,5 \tag{8.8}$$

dimana $N = 10$. Disini terlihat bahwa batas awal interval, $a = 0$; dan batas akhir $b = 2$.

Dalam penerapan metode euler, pertama kali yang harus dilakukan adalah menghitung *step-size* (h), caranya

$$h = \frac{b-a}{N} = \frac{2-0}{10} = 0,2$$

kemudian dilanjutkan dengan menentukan posisi titik-titik t_i berdasarkan rumus

$$t_i = a + ih = 0 + i(0,2) \quad \text{sehingga} \quad t_i = 0,2i$$

serta menetapkan nilai w_0 yang diambil dari syarat awal $y(0) = 0,5$

$$w_0 = 0,5$$

Dengan demikian persamaan euler dapat dinyatakan sebagai

$$\begin{aligned} w_{i+1} &= w_i + h(w_i - t_i^2 + 1) \\ &= w_i + 0,2(w_i - 0,04i^2 + 1) \\ &= 1,2w_i - 0,008i^2 + 0,2 \end{aligned}$$

dimana $i = 0, 1, 2, ..., N-1$. Karena $N = 10$, maka $i = 0, 1, 2, ..., 9$.
Pada saat $i = 0$ dan dari syarat awal diketahui $w_0 = 0,5$, kita bisa menghitung w_1

$$w_1 = 1,2w_0 - 0,008(0)^2 + 0,2 = 0,8000000$$

Pada saat $i = 1$

$$w_2 = 1,2w_1 - 0,008(1)^2 + 0,2 = 1,1520000$$

Pada saat $i = 2$

$$w_3 = 1,2w_2 - 0,008(2)^2 + 0,2 = 1,5504000$$

Demikian seterusnya, hingga mencapai $i = 9$

$$w_{10} = 1,2w_9 - 0,008(9)^2 + 0,2 = 4,8657845$$

Berikut ini adalah *script* matlab untuk menghitung w_1, w_2, sampai w_{10}

```
clear all
clc

format long

b=2; %batas akhir interval
a=0; %batas awal  interval
N=10; % bilangan interger positif
h=(b-a)/N; % nilai step-size
w0=0.5; % nilai w awal
t0=0;   % nilai t awal

% perubahan t sesuai step-size h adalah:
t1=a+1*h;
```

```
t2=a+2*h;
t3=a+3*h;
t4=a+4*h;
t5=a+5*h;
t6=a+6*h;
t7=a+7*h;
t8=a+8*h;
t9=a+9*h;
t10=a+10*h;

% solusinya:
w1=w0+h*(w0-t0^2+1)
w2=w1+h*(w1-t1^2+1)
w3=w2+h*(w2-t2^2+1)
w4=w3+h*(w3-t3^2+1)
w5=w4+h*(w4-t4^2+1)
w6=w5+h*(w5-t5^2+1)
w7=w6+h*(w6-t6^2+1)
w8=w7+h*(w7-t7^2+1)
w9=w8+h*(w8-t8^2+1)
w10=w9+h*(w9-t9^2+1)
```

Atau bisa dipersingkat sebagai berikut

```
clear all
clc

format long

b=2; %batas akhir interval
a=0; %batas awal  interval
N=10; % bilangan interger positif
h=(b-a)/N; % nilai step-size
w0=0.5; % nilai w awal
t0=0;   % nilai t awal

% perubahan t sesuai step-size h adalah:
for i=1:N
   t(i)=a+(i*h);
end

% solusinya:
w(1)=w0+h*(w0-t0^2+1);
for i=2:N
   k=i-1;
   w(i)=w(k)+h*(w(k)-t(k)^2+1);
end
w
```

Disisi lain, solusi exact persamaan diferensial (8.8) adalah

$$y(t) = (t+1)^2 - 0,5e^t \tag{8.9}$$

Script matlab untuk mendapatkan solusi exact ini adalah:

```
clear all
clc
```

```
3
4  format long
5
6  b=2; %batas akhir interval
7  a=0; %batas awal  interval
8  N=10; % bilangan interger positif
9  h=(b-a)/N; % nilai step-size
10
11 % perubahan t sesuai step-size h adalah:
12 for i=1:N
13    t(i)=a+(i*h);
14 end
15
16 % solusi exact:
17 for i=1:N
18    y(i)=(t(i)+1)^2-0.5*exp(t(i));
19 end
20 y
```

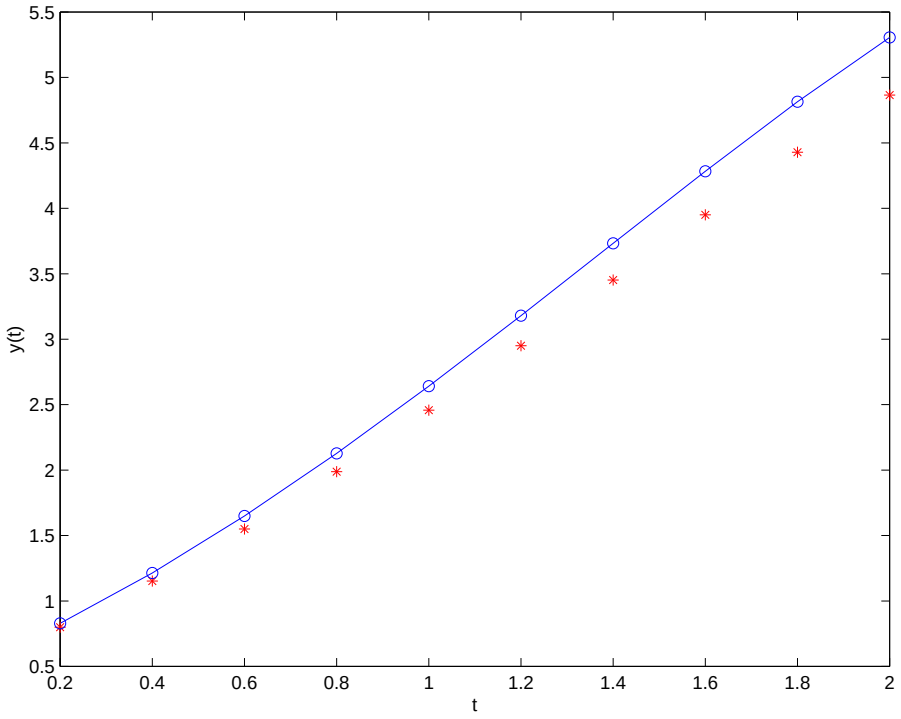
Tabel 8.1: Solusi yang ditawarkan oleh metode euler w_i dan solusi exact $y(t_i)$ serta selisih antara keduanya

i	t_i	w_i	$y_i = y(t_i)$	$\lvert w_i - y_i \rvert$
0	0,0	0,5000000	0,5000000	0,0000000
1	0,2	0,8000000	0,8292986	0,0292986
2	0,4	1,1520000	1,2140877	0,0620877
3	0,6	1,5504000	1,6489406	0,0985406
4	0,8	1,9884800	2,1272295	0,1387495
5	1,0	2,4581760	2,6408591	0,1826831
6	1,2	2,9498112	3,1799415	0,2301303
7	1,4	3,4517734	3,7324000	0,2806266
8	1,6	3,9501281	4,2834838	0,3333557
9	1,8	4,4281538	4,8151763	0,3870225
10	2,0	4,8657845	5,3054720	0,4396874

Coba anda perhatikan sejenak bagian kolom selisih $|w_i - y_i|$. Terlihat angkanya tumbuh semakin besar seiring dengan bertambahnya t_i. Artinya, ketika t_i membesar, akurasi metode euler justru berkurang. Untuk lebih jelasnya, mari kita plot hasil-hasil ini dalam suatu gambar.

Gambar (8.2) memperlihatkan sebaran titik-titik merah yang merupakan hasil perhitungan metode euler (w_i). Sementara solusi exact $y(t_i)$ diwakili oleh titik-titik biru. Tampak jelas bahwa titik-titik biru dan titik-titik merah –pada nilai t yang sama– tidak ada yang berhimpit alias ada jarak yang memisahkan mereka. Bahkan semakin ke kanan, jarak itu semakin melebar. Adanya jarak, tak lain menunjukkan keberadaan *error* (kesalahan). Hasil perhitungan metode euler yang diwakili oleh titik-titik merah ternyata menghadirkan tingkat kesalahan yang semakin membesar ketika menuju ke-N atau ketika t_i bertambah. Untuk mengatasi hal ini, salah satu pemecahannya adalah dengan menerapkan metode Runge-Kutta orde-4. Namun sebelum masuk ke pembahasan tersebut, ada baiknya kita memodifikasi *script* matlab yang terakhir tadi.

Saya kira tidak ada salahnya untuk mengantisipasi kesalahan pengetikan fungsi turunan



Gambar 8.2: Kurva biru adalah solusi exact, dimana lingkaran-lingkaran kecil warna biru pada kurva menunjukkan posisi pasangan absis t dan ordinat $y(t)$ yang dihitung oleh Persamaan (8.9). Sedangkan titik-titik merah mengacu pada hasil perhitungan metode euler, yaitu nilai w_i.

yang terdapat dalam *script* sebelumnya yaitu,

```
w(1)=w0+h*(w0-t0^2+1);
```

dan

```
w(i)=w(k)+h*(w(k)-t(k)^2+1);
```

Ketika fungsi turunan memiliki formulasi yang berbeda dengan contoh di atas, bisa jadi kita akan lupa untuk mengetikkan formulasi yang baru di kedua baris tersebut. Oleh karena itu, lebih baik fungsi turunan tersebut dipindahkan kedalam satu file terpisah. Di lingkungan matlab, file tersebut disebut file *function*. Jadi, isi file *function* untuk contoh yang sedang kita bahas ini adalah

```
function y = futur(t,w)
y = w - t^2 + 1;
```

File *function* ini mesti di-*save* dengan nama file yang sama persis dengan nama fungsinya, dalam contoh ini nama file *function* tersebut harus bernama *futur.m*. Kemudian file ini harus disimpan dalam folder yang sama dimana disana juga terdapat file untuk memproses metode euler.

Setelah itu, script metode euler dimodifikasi menjadi seperti ini

```
1 clear all
2 clc
3
```

```
4  format long
5
6  b=2;         %batas akhir interval
7  a=0;         %batas awal interval
8  N=10; % bilangan interger positif
9  h=(b-a)/N; % nilai step-size
10 w0=0.5; % nilai w awal
11 t0=0; % nilai t awal
12
13 % perubahan t sesuai step-size h adalah:
14 for i=1:N
15         t(i)=a+(i*h);
16 end
17
18 % solusinya:
19 w(1)=w0+h*futur(t0,w0);
20 for i=2:N
21         k=i-1;
22         w(i)=w(k)+h*futur(t(k),w(k));
23 end
24 w
```

Mulai dari baris ke-13 sampai dengan baris ke-24, tidak perlu diubah-ubah lagi. Artinya, jika ada perubahan formulasi fungsi turunan, maka itu cukup dilakukan pada file *futur.m* saja.

Ok. Sekarang mari kita membahas metode Runge Kutta.

8.2 Metode Runge Kutta

Pada saat membahas metode Euler untuk penyelesaian persamaan diferensial, kita telah sampai pada kesimpulan bahwa *truncation error* metode Euler terus membesar seiring dengan bertambahnya iterasi (t_i). Dikaitkan dengan hal tersebut, metode Runge-Kutta Orde-4 menawarkan penyelesaian persamaan diferensial dengan pertumbuhan *truncation error* yang jauh lebih kecil. Persamaan-persamaan yang menyusun metode Runge-Kutta Orde-4 adalah

$$\begin{aligned} w_0 &= \alpha \\ k_1 &= hf(t_i, w_i) && (8.10)\\ k_2 &= hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1) && (8.11)\\ k_3 &= hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2) && (8.12)\\ k_4 &= hf(t_{i+1}, w_i + k_3) && (8.13)\\ w_{i+1} &= w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) && (8.14) \end{aligned}$$

dimana fungsi $f(t, w)$ adalah fungsi turunan.

Contoh

Saya ambilkan contoh yang sama seperti contoh yang sudah kita bahas pada metode Euler. Diketahui persamaan diferensial

$$y' = y - t^2 + 1, \qquad 0 \le t \le 2, \qquad y(0) = 0,5$$

Jika $N = 10$, maka *step-size* bisa dihitung terlebih dahulu

$$h = \frac{b-a}{N} = \frac{2-0}{10} = 0,2$$

dan

$$t_i = a + ih = 0 + i(0,2) \qquad \rightarrow \qquad t_i = 0,2i$$

serta

$$w_0 = 0,5$$

Sekarang mari kita terapkan metode Runge-Kutta Orde-4 ini. Untuk menghitung w_1, tahap-tahap perhitungannya dimulai dari menghitung k_1

$$\begin{aligned} k_1 &= hf(t_0, w_0) \\ &= h(w_0 - t_0^2 + 1) \\ &= 0,2((0,5) - (0,0)^2 + 1) \\ &= 0,3 \end{aligned}$$

lalu menghitung k_2

$$\begin{aligned} k_2 &= hf(t_0 + \frac{h}{2}, w_0 + \frac{k_1}{2}) \\ &= h[(w_0 + \frac{k_1}{2}) - (t_0 + \frac{h}{2})^2 + 1)] \\ &= 0,2[(0,5 + \frac{0,3}{2}) - (0,0 + \frac{0,2}{2})^2 + 1)] \\ &= 0,328 \end{aligned}$$

dilanjutkan dengan k_3

$$\begin{aligned} k_3 &= hf(t_0 + \frac{h}{2}, w_0 + \frac{k_2}{2}) \\ &= h[(w_0 + \frac{k_2}{2}) - (t_0 + \frac{h}{2})^2 + 1)] \\ &= 0,2[(0,5 + \frac{0,328}{2}) - (0,0 + \frac{0,2}{2})^2 + 1)] \\ &= 0,3308 \end{aligned}$$

kemudian k_4

$$\begin{aligned} k_4 &= hf(t_1, w_0 + k_3) \\ &= h[(w_0 + k_3) - t_1^2 + 1] \\ &= 0,2[(0,5 + 0,3308) - (0,2)^2 + 1] \\ &= 0,35816 \end{aligned}$$

akhirnya diperoleh w_1

$$\begin{aligned} w_1 &= w_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ &= 0,5 + \frac{1}{6}(0,3 + 2(0,328) + 2(0,3308) + 0,35816) \\ &= 0,5 + \frac{1}{6}(0,3 + 0,656 + 0,6616 + 0,35816) \\ &= 0,8292933 \end{aligned}$$

Dengan cara yang sama, w_2, w_3, w_4 dan seterusnya dapat dihitung dengan program komputer. Script matlab-nya sebagai berikut[1]:

```
clear all
clc

format long

b=2;         %batas akhir interval
a=0;         %batas awal interval
N=10; % bilangan interger positif
h=(b-a)/N; % nilai step-size
w0=0.5; % nilai w awal
t0=0; % nilai t awal

% perubahan t sesuai step-size h adalah:
for i=1:N
        t(i)=a+(i*h);
end

% solusinya:
k1=h*futur(t0,w0);
k2=h*futur(t0+h/2,w0+k1/2);
k3=h*futur(t0+h/2,w0+k2/2);
k4=h*futur(t(1),w0+k3);
w(1)=w0+1/6*(k1+2*k2+2*k3+k4);

for i=2:N
        k=i-1;
   k1=h*futur(t(k),w(k));
        k2=h*futur(t(k)+h/2,w(k)+k1/2);
        k3=h*futur(t(k)+h/2,w(k)+k2/2);
        k4=h*futur(t(i),w(k)+k3);
        w(i)=w(k)+1/6*(k1+2*k2+2*k3+k4);
end
w
```

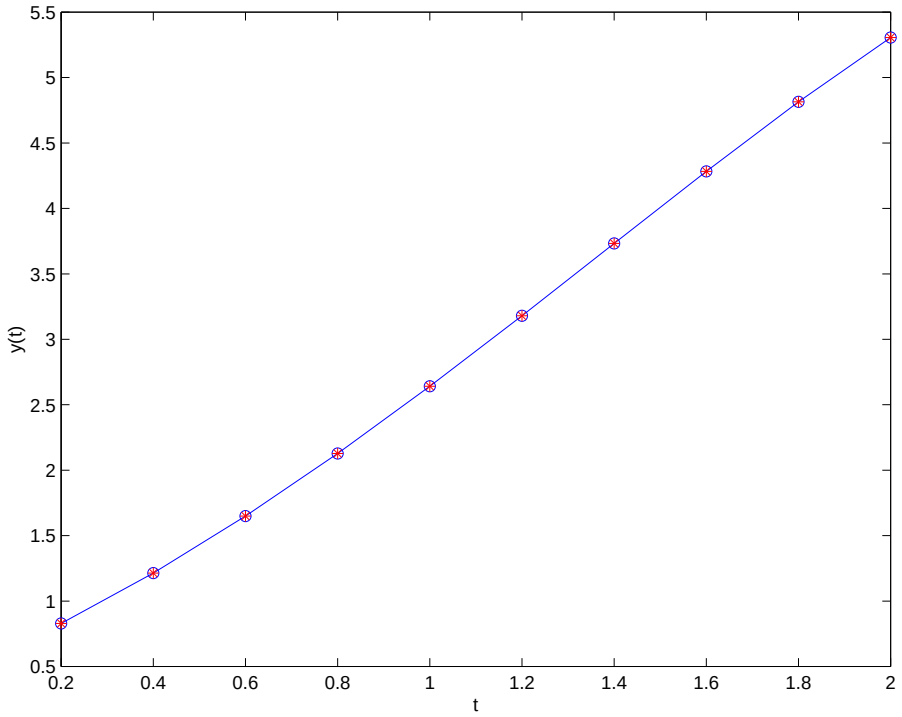
Dibandingkan dengan metode Euler, tingkat pertumbuhan *truncation error,* pada kolom $|w_i - y_i|$ (lihat Tabel 8.2), jauh lebih rendah sehingga metode Runge-Kutta Orde Empat lebih disukai untuk membantu menyelesaikan persamaan-diferensial-biasa.

Contoh tadi tampaknya dapat memberikan gambaran yang jelas bahwa metode Runge-Kutta Orde Empat dapat menyelesaikan persamaan diferensial biasa dengan tingkat akurasi

[1]Jangan lupa, file *futur.m* mesti berada dalam satu folder dengan file Runge Kutta nya!

Tabel 8.2: Solusi yang ditawarkan oleh metode Runge Kutta orde 4 (w_i) dan solusi exact $y(t_i)$ serta selisih antara keduanya

i	t_i	w_i	$y_i = y(t_i)$	$\lvert w_i - y_i \rvert$
0	0,0	0,5000000	0,5000000	0,0000000
1	0,2	0,8292933	0,8292986	0,0000053
2	0,4	1,2140762	1,2140877	0,0000114
3	0,6	1,6489220	1,6489406	0,0000186
4	0,8	2,1272027	2,1272295	0,0000269
5	1,0	2,6408227	2,6408591	0,0000364
6	1,2	3,1798942	3,1799415	0,0000474
7	1,4	3,7323401	3,7324000	0,0000599
8	1,6	4,2834095	4,2834838	0,0000743
9	1,8	4,8150857	4,8151763	0,0000906
10	2,0	5,3053630	5,3054720	0,0001089



Gambar 8.3: Kurva biru adalah solusi exact, dimana lingkaran-lingkaran kecil warna biru pada kurva menunjukkan posisi pasangan absis t dan ordinat $y(t)$ yang dihitung oleh Persamaan (8.9). Sedangkan titik-titik merah mengacu pada hasil perhitungan metode Runge Kutta orde 4, yaitu nilai w_i.

yang lebih tinggi. Namun, kalau anda jeli, ada suatu pertanyaan cukup serius yaitu apakah metode ini dapat digunakan bila pada persamaan diferensialnya tidak ada variabel t ? Misalnya pada kasus pengisian muatan pada kapasitor berikut ini.

8.2.1 Aplikasi: Pengisian muatan pada kapasitor

Sebuah kapasitor yang tidak bermuatan dihubungkan secara seri dengan sebuah resistor dan baterry (Gambar 8.4). Diketahui ϵ = 12 volt, C = 5,00 μF dan R = 8,00 $\times 10^5 \Omega$. Saat saklar

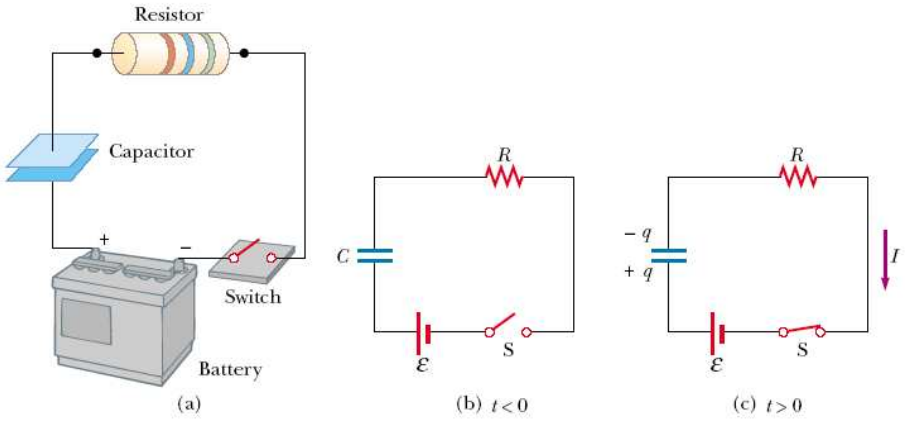
dihubungkan (t=0), muatan belum ada (q=0).

$$\frac{dq}{dt} = \frac{\epsilon}{R} - \frac{q}{RC} \tag{8.15}$$

Solusi exact persamaan (8.15) adalah

$$q_{exact} = q(t) = C\epsilon\left(1 - e^{-t/RC}\right) \tag{8.16}$$

Anda bisa lihat semua suku di ruas kanan persamaan (8.15) tidak mengandung variabel



Gambar 8.4: Rangkaian RC

t. Padahal persamaan-persamaan turunan pada contoh sebelumnya mengandung variabel t. Apakah persamaan (8.15) tidak bisa diselesaikan dengan metode Runge-Kutta? Belum tentu. Sekarang, kita coba selesaikan, pertama kita nyatakan

$$\begin{aligned} m_1 &= \frac{\epsilon}{R} = 1,5 \times 10^{-5} \\ m_2 &= \frac{1}{RC} = 0,25 \end{aligned}$$

sehingga persamaan (8.15) dimodifikasi menjadi

$$\begin{aligned} \frac{dq}{dt} = f(q_i) &= m_1 - q_i m_2 \\ t_i &= a + ih \end{aligned}$$

Jika $t_0 = 0$, maka $a = 0$, dan pada saat itu (secara fisis) diketahui $q_0 = 0,0$. Lalu jika ditetapkan $h = 0,1$ maka $t_1 = 0,1$ dan kita bisa mulai menghitung k_1 dengan menggunakan $q_0 = 0,0$, walaupun t_1 tidak dilibatkan dalam perhitungan ini

$$\begin{aligned} k_1 &= hf(q_0) \\ &= h(m_1 - q_0 m_2) \\ &= 0,1((1,5 \times 10^{-5}) - (0,0)(0,25)) \\ &= 0,150 \times 10^{-5} \end{aligned}$$

lalu menghitung k_2

$$\begin{aligned} k_2 &= hf(q_0 + \frac{k_1}{2}) \\ &= h[(m_1 - (q_0 + \frac{k_1}{2})m_2)] \\ &= 0,1[(1,5 \times 10^{-5} - ((0,0) + \frac{0,15 \times 10^{-5}}{2})(0,25)] \\ &= 0,14813 \times 10^{-5} \end{aligned}$$

dilanjutkan dengan k_3

$$\begin{aligned} k_3 &= hf(q_0 + \frac{k_2}{2}) \\ &= h[(m_1 - (q_0 + \frac{k_2}{2})m_2)] \\ &= 0,1[(1,5 \times 10^{-5} - ((0,0) + \frac{0,14813 \times 10^{-5}}{2})(0,25)] \\ &= 0,14815 \times 10^{-5} \end{aligned}$$

kemudian k_4

$$\begin{aligned} k_4 &= hf(q_0 + k_3) \\ &= h[(m_1 - (q_0 + k_3)m_2)] \\ &= 0,1[(1,5 \times 10^{-5} - ((0,0) + 0,14815 \times 10^{-5})(0,25)] \\ &= 0,14630 \times 10^{-5} \end{aligned}$$

akhirnya diperoleh q_1

$$\begin{aligned} q_1 &= q_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ &= 0,0 + \frac{1}{6}(0,150 + 2(0,14813) + 2(0,14815) + 0,14630) \times 10^{-5} \\ &= 0,14814 \times 10^{-5} \end{aligned}$$

Selanjutnya q_2 dihitung. Tentu saja pada saat t_2, dimana $t_2 = 0,2$, namun sekali lagi, t_2 tidak terlibat dalam perhitungan ini. Dimulai menghitung k_1 kembali

$$\begin{aligned} k_1 &= hf(q_1) \\ &= h(m_1 - q_1 m_2) \\ &= 0,1((1,5 \times 10^{-5}) - (0,14814 \times 10^{-5})(0,25)) \\ &= 0,14630 \times 10^{-5} \end{aligned}$$

lalu menghitung k_2

$$\begin{aligned} k_2 &= hf(q_1 + \frac{k_1}{2}) \\ &= h[(m_1 - (q_1 + \frac{k_1}{2})m_2)] \\ &= 0,1[(1,5 \times 10^{-5} - ((0,14814 \times 10^{-5}) + \frac{0,14630 \times 10^{-5}}{2})(0,25)] \\ &= 0,14447 \times 10^{-5} \end{aligned}$$

dilanjutkan dengan k_3

$$\begin{aligned} k_3 &= hf(q_1 + \frac{k_2}{2}) \\ &= h[(m_1 - (q_1 + \frac{k_2}{2})m_2)] \\ &= 0,1[(1,5 \times 10^{-5} - ((0,14814 \times 10^{-5}) + \frac{0,14447 \times 10^{-5}}{2})(0,25)] \\ &= 0,14449 \times 10^{-5} \end{aligned}$$

kemudian k_4

$$\begin{aligned} k_4 &= hf(q_1 + k_3) \\ &= h[(m_1 - (q_1 + k_3)m_2)] \\ &= 0,1[(1,5 \times 10^{-5} - ((0,14814 \times 10^{-5}) + 0,14449 \times 10^{-5})(0,25)] \\ &= 0,14268 \times 10^{-5} \end{aligned}$$

akhirnya diperoleh q_2

$$\begin{aligned} q_2 &= q_1 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ &= 0,14814 \times 10^{-5} + \frac{1}{6}(0,14630 + 2(0,14447) + 2(0,14449) + 0,14268) \times 10^{-5} \\ &= 0,29262 \times 10^{-5} \end{aligned}$$

Dengan cara yang sama, q_3, q_4, q_5 dan seterusnya dapat dihitung. Berikut ini adalah script dalam matlab yang dipakai untuk menghitung q

```
clear all
clc

format long

b=1; % batas akhir interval
a=0; % batas awal  interval
h=0.1; % interval waktu
N=(b-a)/h; % nilai step-size
q0=0.0; % muatan mula-mula
t0=0.0; % waktu awal

```

```
% perubahan t sesuai step-size h adalah:
for i=1:N
   t(i)=a+(i*h);
end

% solusinya:
k1=h*futur(q0);
k2=h*futur(q0+k1/2);
k3=h*futur(q0+k2/2);
k4=h*futur(q0+k3);
q(1)=q0+1/6*(k1+2*k2+2*k3+k4);

for i=2:N
        k=i-1;
          k1=h*futur(q(k));
        k2=h*futur(q(k)+k1/2);
        k3=h*futur(q(k)+k2/2);
        k4=h*futur(q(k)+k3);
        q(i)=q(k)+1/6*(k1+2*k2+2*k3+k4);
end
q
```

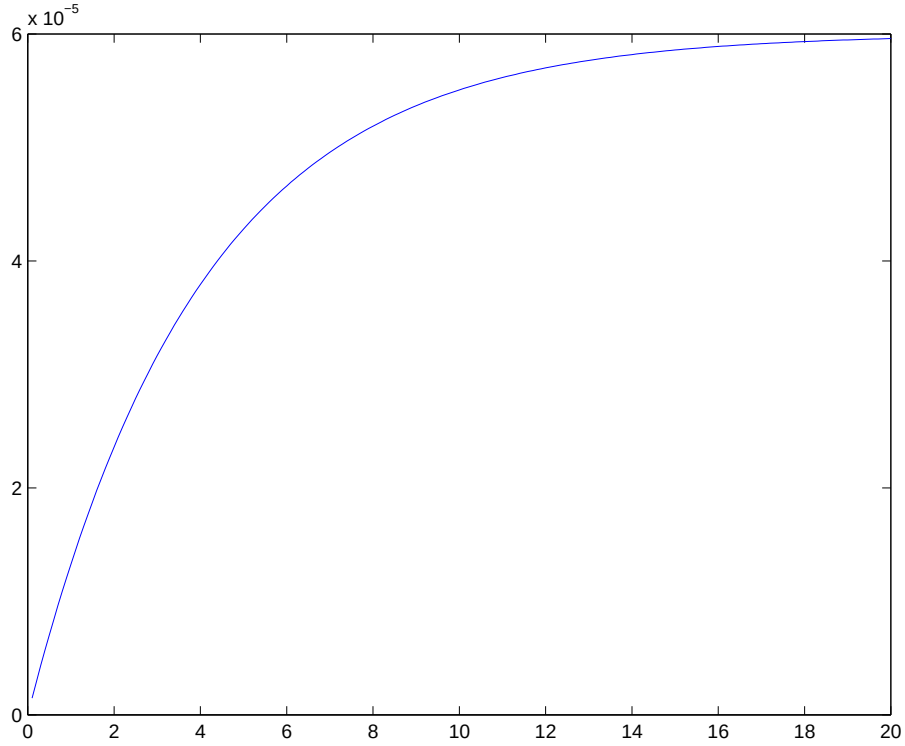
Adapun *script* fungsi turunannya (*futur.m*) adalah sebagai berikut:

```
function y=futur(q)
E=12;                          % tegangan (volt)
R=800000;        % hambatan (ohm)
C=5e-6;                % kapasitansi (farad)
m1=E/R;
m2=1/(R*C);
y=m1-(m2*q);
```

Tabel 8.3: Perbandingan antara hasil perhitungan numerik lewat metode Runge Kutta dan hasil perhitungan dari solusi exact, yaitu persamaan (8.16)

i	t_i	q_i	$q_{exact} = q(t_i)$	$\lvert q_i - q_{exact}\rvert$
0	0,0	$0{,}00000\times10^{-5}$	$0{,}00000\times10^{-5}$	0,00000
1	0,1	$0{,}14814\times10^{-5}$	$0{,}14814\times10^{-5}$	0,00000
2	0,2	$0{,}29262\times10^{-5}$	$0{,}29262\times10^{-5}$	0,00000
3	0,3	$0{,}43354\times10^{-5}$	$0{,}43354\times10^{-5}$	0,00000
4	0,4	$0{,}57098\times10^{-5}$	$0{,}57098\times10^{-5}$	0,00000
5	0,5	$0{,}70502\times10^{-5}$	$0{,}70502\times10^{-5}$	0,00000
6	0,6	$0{,}83575\times10^{-5}$	$0{,}83575\times10^{-5}$	0,00000
7	0,7	$0{,}96326\times10^{-5}$	$0{,}96326\times10^{-5}$	0,00000
8	0,8	$1{,}0876\times10^{-5}$	$1{,}0876\times10^{-5}$	0,00000
9	0,9	$1{,}2089\times10^{-5}$	$1{,}2089\times10^{-5}$	0,00000
10	1,0	$1{,}3272\times10^{-5}$	$1{,}3272\times10^{-5}$	0,00000

Luar biasa!! Tak ada *error* sama sekali. Mungkin, kalau kita buat 7 angka dibelakang koma, *error*nya akan terlihat. Tapi kalau anda cukup puas dengan 5 angka dibelakang koma, hasil ini sangat memuaskan. Gambar 8.5 memperlihatkan kurva penumpukan muatan q terhadap waktu t – dengan batas atas interval waktu dinaikkan hingga 20 –.



Gambar 8.5: Kurva pengisian muatan q (*charging*) terhadap waktu t

Sampai disini mudah-mudahan jelas dan bisa dimengerti. Silakan anda coba untuk kasus yang lain, misalnya proses pembuangan (*discharging*) q pada rangkaian yang sama, atau bisa juga anda berlatih dengan rangkaian RL dan RLC. Saya akhiri dulu uraian saya sampai disini.

8.3 Metode *Finite Difference*

Suatu persamaan diferensial dapat dinyatakan sebagai berikut:

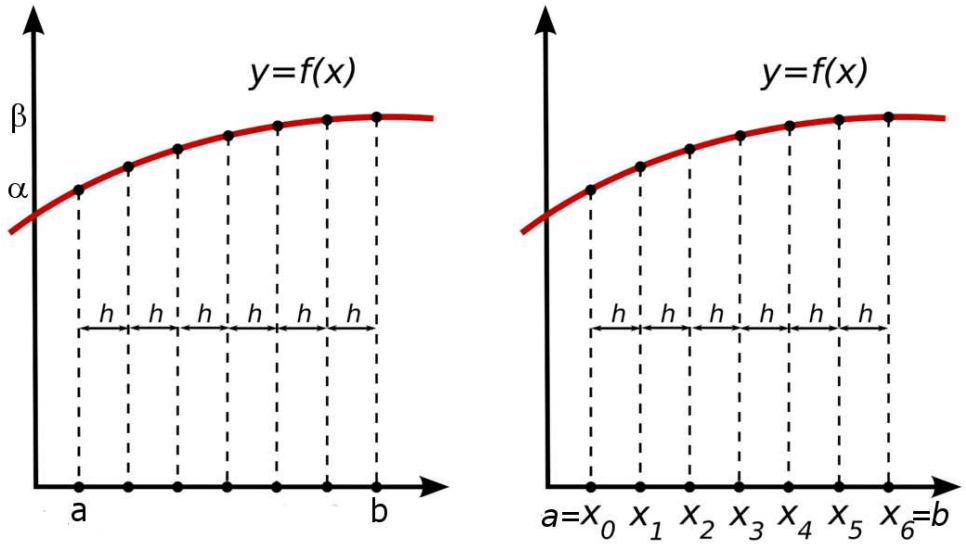
$$\frac{d^2y}{dx^2}(x) = p(x)\frac{dy}{dx}(x) + q(x)y(x) + r(x), \qquad a \leq x \leq b, \qquad y(a) = \alpha, \qquad y(b) = \beta \tag{8.17}$$

atau juga dapat dituliskan dalam bentuk lain

$$y'' = p(x)y' + q(x)y + r(x) \tag{8.18}$$

Persamaan tersebut dapat diselesaikan dengan melakukan pendekatan numerik terhadap y'' dan y'. Caranya adalah pertama, kita memilih angka integer sembarang yaitu N dimana $N > 0$ dan membagi interval $[a, b]$ dengan $(N + 1)$, hasilnya dinamakan h (lihat Gambar 8.6)

$$h = \frac{b - a}{N + 1} \tag{8.19}$$



Gambar 8.6: Kurva suatu fungsi $f(x)$ yang dibagi sama besar berjarak h. Evaluasi kurva yang dilakukan *Finite-Difference* dimulai dari batas bawah $X_0 = a$ hingga batas atas $x_6 = b$

Dengan demikian maka titik-titik x yang merupakan sub-interval antara a dan b dapat dinyatakan sebagai

$$x_i = a + ih, \qquad i = 0, 1, ..., N + 1 \tag{8.20}$$

Pencarian solusi persamaan diferensial melalui pendekatan numerik dilakukan dengan memanfaatkan polinomial Taylor untuk mengevaluasi y'' dan y' pada x_{i+1} dan x_{i-1} seperti berikut ini

$$y(x_{i+1}) = y(x_i + h) = y(x_i) + hy'(x_i) + \frac{h^2}{2}y''(x_i) \tag{8.21}$$

dan

$$y(x_{i-1}) = y(x_i - h) = y(x_i) - hy'(x_i) + \frac{h^2}{2}y''(x_i) \tag{8.22}$$

Jika kedua persamaan ini dijumlahkan

$$y(x_{i+1}) + y(x_{i-1}) = 2y(x_i) + h^2y''(x_i)$$

Dari sini y'' dapat ditentukan

$$h^2y''(x_i) = y(x_{i+1}) - 2y(x_i) + y(x_{i-1})$$

$$y''(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} \tag{8.23}$$

Dengan cara yang sama, $y'(x_i)$ dapat dicari sebagai berikut

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_{i-1})}{2h} \tag{8.24}$$

Selanjutnya persamaan (8.23) dan (8.24) disubstitusikan ke persamaan (8.18) maka

$$\begin{aligned}
\frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} &= p(x_i)\frac{y(x_{i+1}) - y(x_{i-1})}{2h} + q(x_i)y(x_i) + r(x_i) \\
\frac{-y(x_{i+1}) + 2y(x_i) - y(x_{i-1})}{h^2} &= -p(x_i)\frac{y(x_{i+1}) - y(x_{i-1})}{2h} - q(x_i)y(x_i) - r(x_i) \\
\frac{-y(x_{i+1}) + 2y(x_i) - y(x_{i-1})}{h^2} &+ p(x_i)\frac{y(x_{i+1}) - y(x_{i-1})}{2h} + q(x_i)y(x_i) = -r(x_i)
\end{aligned}$$

Sebelum dilanjut, saya nyatakan bahwa $y(x_{i+1})$=w_{i+1} dan $y(x_i)$=w_i serta $y(x_{i-1})$=w_{i-1}. Maka persamaan di atas dapat ditulis sebagai berikut

$$\begin{aligned}
\left(\frac{-w_{i+1} + 2w_i - w_{i-1}}{h^2}\right) + p(x_i)\left(\frac{w_{i+1} - w_{i-1}}{2h}\right) + q(x_i)w_i &= -r(x_i) \\
(-w_{i+1} + 2w_i - w_{i-1}) + \frac{h}{2}p(x_i)\,(w_{i+1} - w_{i-1}) + h^2 q(x_i)w_i &= -h^2 r(x_i) \\
-w_{i+1} + 2w_i - w_{i-1} + \frac{h}{2}p(x_i)w_{i+1} - \frac{h}{2}p(x_i)w_{i-1} + h^2 q(x_i)w_i &= -h^2 r(x_i) \\
-w_{i-1} - \frac{h}{2}p(x_i)w_{i-1} + 2w_i + h^2 q(x_i)w_i - w_{i+1} + \frac{h}{2}p(x_i)w_{i+1} &= -h^2 r(x_i) \\
-\left(1 + \frac{h}{2}p(x_i)\right)w_{i-1} + \left(2 + h^2 q(x_i)\right)w_i - \left((1 - \frac{h}{2}p(x_i)\right)w_{i+1} &= -h^2 r(x_i) \qquad (8.25)
\end{aligned}$$

dimana i=1,2,3...sampai N, karena yang ingin kita cari adalah w_1, w_2, w_3,..., w_N. Sementara, satu hal yang tak boleh dilupakan yaitu w_0 dan w_{N+1} biasanya selalu **sudah diketahui**. Pada persamaan (8.17), jelas-jelas sudah diketahui bahwa w_0=α dan w_{N+1}=β; keduanya dikenal sebagai **syarat batas** atau istilah asingnya adalah **boundary value**. Topik yang sedang bahas ini juga sering disebut sebagai **Masalah Syarat Batas** atau **Boundary Value Problem**.

Sampai disini kita mendapatkan sistem persamaan linear yang selanjutnya dapat dinyatakan sebagai bentuk operasi matrik

$$\mathbf{Aw} = \mathbf{b} \qquad (8.26)$$

dimana **A** adalah matrik tridiagonal dengan orde $N \times N$

$$\mathbf{A} = \begin{bmatrix}
2 + h^2 q(x_1) & -1 + \frac{h}{2}p(x_1) & 0 & \dots & \dots & \dots & 0 \\
-1 - \frac{h}{2}p(x_2) & 2 + h^2 q(x_2) & -1 + \frac{h}{2}p(x_2) & 0 & \dots & \dots & 0 \\
0 & -1 - \frac{h}{2}p(x_3) & 2 + h^2 q(x_3) & -1 + \frac{h}{2}p(x_3) & 0 & \dots & 0 \\
0 & 0 & -1 - \frac{h}{2}p(x_4) & 2 + h^2 q(x_4) & -1 + \frac{h}{2}p(x_4) & 0 & 0 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots \\
\dots & \dots & \dots & \dots & -1 - \frac{h}{2}p(x_{N-1}) & 2 + h^2 q(x_{N-1}) & -1 + \frac{h}{2}p(x_{N-1}) \\
0 & \dots & \dots & \dots & \dots & -1 - \frac{h}{2}p(x_N) & 2 + h^2 q(x_N)
\end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ \vdots \\ w_{N-1} \\ w_N \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -h^2 r(x_1) + \left(1 + \frac{h}{2} p(x_1)\right) w_0 \\ -h^2 r(x_2) \\ -h^2 r(x_3) \\ -h^2 r(x_4) \\ \vdots \\ -h^2 r(x_{N-1}) \\ -h^2 r(x_N) + \left(1 - \frac{h}{2} p(x_N)\right) w_{N+1} \end{bmatrix}$$

8.3.1 Script *Finite-Difference*

```
clear all
clc

a=1.0; %ganti angkanya sesuai data yang anda miliki
b=2.0; %ganti angkanya sesuai data yang anda miliki
n=9; %ganti angkanya sesuai data yang anda miliki
h=(b-a)/(n+1);
alpha=1; %ganti angkanya sesuai data yang anda miliki
beta=2; %ganti angkanya sesuai data yang anda miliki

%====== Mencari Elemen Matrik A ========
for i=1:n
   x=a+i*h;
   A(i,i)=2+h^2*fungsiQ(x);
end
for i=1:n-1
   x=a+i*h;
   A(i,i+1)=-1+((h/2)*fungsiP(x));
end
for i=2:n
   x=a+i*h;
   A(i,i-1)=-1-((h/2)*fungsiP(x));
end
A
%====== Mencari Elemen Vektor b ========
x=a+h;
b(1,1)=-h^2*fungsiR(x)+(1+((h/2)*fungsiP(x)))*alpha;
for i=2:8
   x=a+i*h;
   b(i,1)=-h^2*fungsiR(x);
end
xn=a+n*h
b(n,1)=-h^2*fungsiR(xn)+(1-((h/2)*fungsiP(xn)))*beta;
b
```

Pada akhirnya, elemen-elemen matrik **A** dan vektor **b** sudah diketahui. Sehingga vektor **w** dapat dihitung dengan berbagai metode pemecahan sistem persamaan linear, seperti Eliminasi Gauss, Gauss-Jourdan, Iterasi Jacobi dan Iterasi Gauss-Seidel.

Contoh
Diketahui persamaan diferensial seperti berikut ini

$$y'' = -\frac{2}{x}y' + \frac{2}{x^2}y + \frac{\sin(\ln x)}{x^2}, \qquad 1 \le x \le 2, \qquad y(1) = 1, \qquad y(2) = 2$$

memiliki solusi exact

$$y = c_1 x + \frac{c_2}{x^2} - \frac{3}{10}\sin(\ln x) - \frac{1}{10}\cos(\ln x),$$

dimana

$$c_2 = \frac{1}{70}\left[8 - 12\sin(\ln 2) - 4\cos(\ln 2)\right] \approx -0,03920701320$$

dan

$$c_1 = \frac{11}{10} - c_2 \approx 1,1392070132.$$

Dengan metode Finite-Difference, solusi pendekatan dapat diperoleh dengan membagi interval $1 \le x \le 2$ menjadi sub-interval, misalnya kita gunakan $N = 9$, sehingga spasi h diperoleh

$$h = \frac{b-a}{N+1} = \frac{2-1}{9+1} = 0,1$$

Dari persamaan diferensial tersebut juga didapat

$$\begin{aligned} p(x_i) &= -\frac{2}{x_i} \\ q(x_i) &= \frac{2}{x_i^2} \\ r(x_i) &= \frac{\sin(\ln x_i)}{x_i^2} \end{aligned}$$

Script matlab telah dibuat untuk menyelesaikan contoh soal ini. Untuk memecahkan persoalan ini, saya membuat 4 buah script, terdiri dari script utama, script fungsiP, script fungsiQ dan script fungsiR. Berikut ini adalah script fungsiP yang disimpan dengan nama file *fungsiP.m*:

```
1 function y = fungsiP(x)
2 y = -2/x;
```

lalu inilah script fungsiQ yang disimpan dengan nama file *fungsiQ.m*:

```
1 function y = fungsiQ(x)
2 y = 2/x^2;
```

kemudian ini script fungsiR yang disimpan dengan nama file *fungsiR.m*::

```
function y = fungsiR(x)
y = sin(log(x))/x^2;
```

dan terakhir, inilah script utamanya:

```
clear all
clc

a=1.0;
b=2.0;

alpha=1;
beta=2;

%=======jika diketahui n, maka h dihitung ====
n=9;
h=(b-a)/(n+1);

%=======jika diketahui h, maka n dihitung ====
%h=0.1;
%n=((b-a)/h)-1;

%====== Mencari Elemen Matrik A ========
for i=1:n
   x=a+i*h;
   A(i,i)=2+h^2*fungsiQ(x);
end
for i=1:n-1
   x=a+i*h;
   A(i,i+1)=-1+((h/2)*fungsiP(x));
end
for i=2:n
   x=a+i*h;
   A(i,i-1)=-1-((h/2)*fungsiP(x));
end
A
%====== Mencari Elemen Vektor b ========
x=a+h;
b(1,1)=-h^2*fungsiR(x)+(1+((h/2)*fungsiP(x)))*alpha;
for i=2:8
   x=a+i*h;
   b(i,1)=-h^2*fungsiR(x);
end
xn=a+n*h
b(n,1)=-h^2*fungsiR(xn)+(1-((h/2)*fungsiP(xn)))*beta;
b
```

```
%====== Menggabungkan Vektor b kedalam matrik A ========
for i=1:n
   A(i,n+1)=b(i,1);
end
A

%&&&&&& Proses Eliminasi Gauss &&&&&&&&&&&&&&&&&&&&&&&&&
%---------Proses Triangularisasi-----------
for j=1:(n-1)

%----mulai proses pivot---
        if (A(j,j)==0)
           for p=1:n+1
              u=A(j,p);
              v=A(j+1,p);
              A(j+1,p)=u;
              A(j,p)=v;
           end
        end
%----akhir proses pivot---
   jj=j+1;
   for i=jj:n
      m=A(i,j)/A(j,j);
      for k=1:(n+1)
         A(i,k)=A(i,k)-(m*A(j,k));
      end
   end
end
%----------------------------------------

%------Proses Substitusi mundur-------------
x(n,1)=A(n,n+1)/A(n,n);

for i=n-1:-1:1
   S=0;
   for j=n:-1:i+1
      S=S+A(i,j)*x(j,1);
   end
   x(i,1)=(A(i,n+1)-S)/A(i,i);
end
%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

%===== Menampilkan Vektor w =================
w=x
```

Tabel berikut ini memperlihatkan hasil perhitungan dengan pendekatan metode Finite-Difference w_i dan hasil perhitungan dari solusi exact $y(x_i)$, dilengkapi dengan selisih antara keduanya

$|w_i - y(x_i)|$. Tabel ini memperlihatkan tingkat kesalahan (error) berada pada orde 10^{-5}. Un-

x_i	w_i	$y(x_i)$	$\lvert w_i - y(x_i)\rvert$
1,0	1,00000000	1,00000000	
1,1	1,09260052	1,09262930	$2{,}88 \times 10^{-5}$
1,2	1,18704313	1,18708484	$4{,}17 \times 10^{-5}$
1,3	1,28333687	1,28338236	$4{,}55 \times 10^{-5}$
1,4	1,38140205	1,38144595	$4{,}39 \times 10^{-5}$
1,5	1,48112026	1,48115942	$3{,}92 \times 10^{-5}$
1,6	1,58235990	1,58239246	$3{,}26 \times 10^{-5}$
1,7	1,68498902	1,68501396	$2{,}49 \times 10^{-5}$
1,8	1,78888175	1,78889853	$1{,}68 \times 10^{-5}$
1,9	1,89392110	1,89392951	$8{,}41 \times 10^{-6}$
2,0	2,00000000	2,00000000	

tuk memperkecil orde kesalahan, kita bisa menggunakan polinomial Taylor berorde tinggi. Akan tetapi proses kalkulasi menjadi semakin banyak dan disisi lain penentuan syarat batas lebih kompleks dibandingkan dengan pemanfaatan polinomial Taylor yang sekarang. Untuk menghindari hal-hal yang rumit itu, salah satu jalan pintas yang cukup efektif adalah dengan menerapkan ekstrapolasi Richardson.

Contoh
Pemanfaatan ekstrapolasi Richardson pada metode Finite Difference untuk persamaan diferensial seperti berikut ini

$$y'' = -\frac{2}{x}y' + \frac{2}{x^2}y + \frac{\sin(\ln x)}{x^2}, \qquad 1 \le x \le 2, \qquad y(1) = 1, \qquad y(2) = 2,$$

dengan $h = 0,1$, $h = 0,05$, $h = 0,025$. Ekstrapolasi Richardson terdiri atas 3 tahapan, yaitu ekstrapolasi yang pertama

$$Ext_{1i} = \frac{4w_i(h = 0,05) - w_i(h = 0,1)}{3}$$

kemudian ekstrapolasi yang kedua

$$Ext_{2i} = \frac{4w_i(h = 0,025) - w_i(h = 0,05)}{3}$$

dan terakhir ekstrapolasi yang ketiga

$$Ext_{3i} = \frac{16Ext_{2i} - Ext_{1i}}{15}$$

Tabel berikut ini memperlihatkan hasil perhitungan tahapan-tahapan ekstrapolasi tersebut. Jika seluruh angka di belakang koma diikut-sertakan, maka akan terlihat selisih antara solusi exact dengan solusi pendekatan sebesar $6,3 \times 10^{-11}$. Ini benar-benar improvisasi yang luar biasa.

x_i	$w_i(h=0,1)$	$w_i(h=0,05)$	$w_i(h=0,025)$	Ext_{1i}	Ext_{2i}	Ext_{3i}
1,0	1,00000000	1,00000000	1,00000000	1,00000000	1,00000000	1,00000000
1,1	1,09260052	1,09262207	1,09262749	1,09262925	1,09262930	1,09262930
1,2	1,18704313	1,18707436	1,18708222	1,18708477	1,18708484	1,18708484
1,3	1,28333687	1,28337094	1,28337950	1,28338230	1,28338236	1,28338236
1,4	1,38140205	1,38143493	1,38144319	1,38144598	1,38144595	1,38144595
1,5	1,48112026	1,48114959	1,48115696	1,48115937	1,48115941	1,48115942
1,6	1,58235990	1,58238429	1,58239042	1,58239242	1,58239246	1,58239246
1,7	1,68498902	1,68500770	1,68501240	1,68501393	1,68501396	1,68501396
1,8	1,78888175	1,78889432	1,78889748	1,78889852	1,78889853	1,78889853
1,9	1,89392110	1,89392740	1,89392898	1,89392950	1,89392951	1,89392951
2,0	2,00000000	2,00000000	2,00000000	2,00000000	2,00000000	2,00000000

8.3.2 Aplikasi

Besar simpangan terhadap waktu ($y(t)$) suatu sistem osilator mekanik yang padanya diberikan gaya secara periodik (*forced-oscilations*) memenuhi persamaan diferensial seperti dibawah ini berikut syarat-syarat batasnya

$$\frac{d^2y}{dt^2} = \frac{dy}{dt} + 2y + cos(t), \qquad 0 \leq t \leq \frac{\pi}{2}, \qquad y(0) = -0,3, \qquad y(\frac{\pi}{2}) = -0,1$$

Dengan metode Finite-Difference, tentukanlah besar masing-masing simpangan di setiap interval $h = \pi/8$. Buatlah table untuk membandingkan hasil finite-difference dengan solusi analitik yang memenuhi $y(t) = -\frac{1}{10}\left[sin(t) + 3cos(t)\right]$.

jawab:
Secara umum, persamaan diferensial dapat dinyatakan sbb:

$$\frac{d^2y}{dx^2}(x) = p(x)\frac{dy}{dx}(x) + q(x)y(x) + r(x), \qquad a \leq x \leq b, \qquad y(a) = \alpha, \qquad y(b) = \beta$$

Dengan membandingkan kedua persamaan di atas, kita bisa definisikan

$$p(t) = 1 \qquad q(t) = 2 \qquad r(t) = \cos(t) \qquad a = 0 \qquad b = \frac{\pi}{2} \qquad \alpha = -0,3 \qquad \beta = -0,1$$

Adapun persamaan finite-difference adalah

$$-\left(1 + \frac{h}{2}p(x_i)\right)w_{i-1} + \left(2 + h^2q(x_i)\right)w_i - \left((1 - \frac{h}{2}p(x_i)\right)w_{i+1} = -h^2r(x_i)$$

Persamaan diatas dikonversi kedalam operasi matriks

$$\mathbf{Aw} = \mathbf{b} \tag{8.27}$$

dimana **A** adalah matrik tridiagonal dengan orde $N \times N$

$$\mathbf{A} = \begin{bmatrix} 2+h^2q(x_1) & -1+\frac{h}{2}p(x_1) & 0 & \dots & \dots & \dots & 0 \\ -1-\frac{h}{2}p(x_2) & 2+h^2q(x_2) & -1+\frac{h}{2}p(x_2) & 0 & \dots & \dots & 0 \\ 0 & -1-\frac{h}{2}p(x_3) & 2+h^2q(x_3) & -1+\frac{h}{2}p(x_3) & 0 & \dots & 0 \\ 0 & 0 & -1-\frac{h}{2}p(x_4) & 2+h^2q(x_4) & -1+\frac{h}{2}p(x_4) & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & -1-\frac{h}{2}p(x_{N-1}) & 2+h^2q(x_{N-1}) & -1+\frac{h}{2}p(x_{N-1}) \\ 0 & \dots & \dots & \dots & \dots & -1-\frac{h}{2}p(x_N) & 2+h^2q(x_N) \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ \vdots \\ w_{N-1} \\ w_N \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -h^2r(x_1) + \left(1+\frac{h}{2}p(x_1)\right)w_0 \\ -h^2r(x_2) \\ -h^2r(x_3) \\ -h^2r(x_4) \\ \vdots \\ -h^2r(x_{N-1}) \\ -h^2r(x_N) + \left(1-\frac{h}{2}p(x_N)\right)w_{N+1} \end{bmatrix}$$

Jumlah baris matrik ditentukan oleh bilangan n. Namun disoal hanya tersedia informasi nilai $h = \pi/8$, sehingga n harus dihitung terlebih dahulu:

$$h = \frac{b-a}{n+1} \qquad n = \frac{b-a}{h} - 1 = \frac{\frac{\pi}{2}-0}{\pi/8} - 1 = 3$$

perhitungan ini dilakukan didalam script matlab. Selanjutnya seluruh elemen matrik **A** dan vektor **b** dihitung dengan matlab

$$\begin{bmatrix} 2,3084 & -0,8037 & 0 \\ -1,1963 & 2,3084 & -0,8037 \\ 0 & -1,1963 & 2,3084 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -0,5014 \\ -0,1090 \\ -0,1394 \end{bmatrix}$$

Proses diteruskan dengan metode Eliminasi Gauss dan didapat hasil akhir berikut ini

$$w_1 = -0.3157 \qquad w_2 = -0.2829 \qquad w_3 = -0.2070$$

8.4 Persamaan Diferensial Parsial

Dalam sub-bab ini, penulisan 'persamaan diferensial parsial' akan dipersingkat menjadi PDP. PDP dapat dibagi menjadi 3 jenis, yaitu persamaan diferensial eliptik, parabolik dan hiperbolik. PDP eliptik dinyatakan sebagai berikut

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y) \tag{8.28}$$

Di bidang fisika, persamaan (8.28) dikenal sebagai **Persamaan Poisson**. Jika $f(x, y)$=0, maka diperoleh persamaan yang lebih sederhana

$$\frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = 0 \tag{8.29}$$

yang biasa disebut sebagai **Persamaan Laplace**. Contoh masalah PDP eliptik di bidang fisika adalah distribusi panas pada kondisi *steady-state* pada obyek 2-dimensi dan 3-dimensi.

Jenis PDP kedua adalah PDP parabolik yang dinyatakan sebagai berikut

$$\frac{\partial u}{\partial t}(x, t) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x, t) = 0 \tag{8.30}$$

Fenomena fisis yang bisa dijelaskan oleh persamaan ini adalah masalah aliran panas pada suatu obyek dalam fungsi waktu t.

Terakhir, PDP ketiga adalah PDP hiperbolik yang dinyatakan sebagai berikut

$$\alpha^2 \frac{\partial^2 u}{\partial^2 x}(x, t) = \frac{\partial^2 u}{\partial t^2}(x, t) \tag{8.31}$$

biasa digunakan untuk menjelaskan fenomena gelombang.

Sekarang, mari kita bahas lebih dalam satu-persatu, difokuskan pada bagaimana cara menyatakan semua PDP di atas dalam formulasi *Finite-Difference*.

8.5 PDP eliptik

Kita mulai dari persamaan aslinya

$$\frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = f(x, y) \tag{8.32}$$

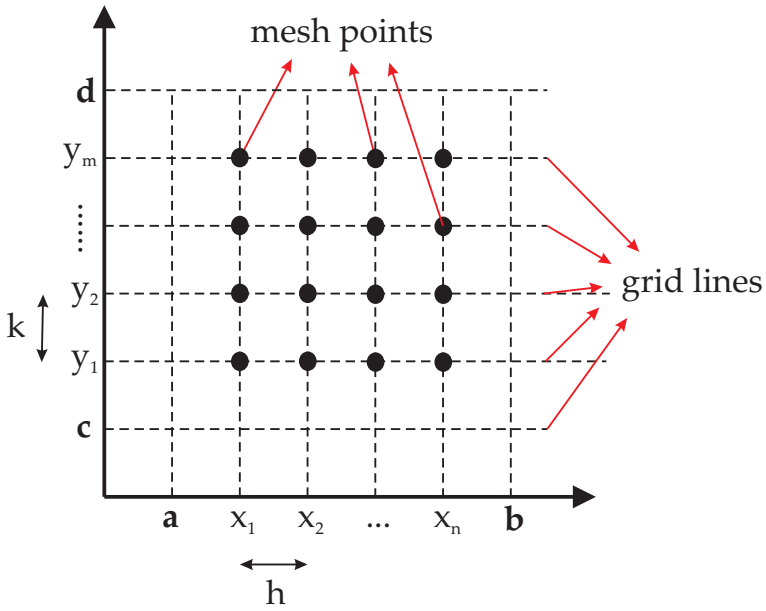
dimana $R = [(x, y)|a < x < b, c < y < d]$. Maksudnya, variasi titik-titik x berada di antara a dan b. Demikian pula dengan variasi titik-titik y, dibatasi mulai dari c sampai d (lihat Gambar 8.7). Jika h adalah jarak interval antar titik yang saling bersebelahan pada titik-titik dalam rentang horizontal a dan b, maka titik-titik variasi di antara a dan b dapat diketahui melalui rumus ini

$$x_i = a + ih, \quad \text{dimana } i = 1, 2, \ldots, n \tag{8.33}$$

dimana a adalah titik awal pada sumbu horisontal x. Demikian pula pada sumbu y. Jika k adalah jarak interval antar titik yang bersebelahan pada titik-titik dalam rentang vertikal c dan d, maka titik-titik variasi di antara c dan d dapat diketahui melalui rumus ini

$$y_j = c + jk, \quad \text{dimana } j = 1, 2, \ldots, m \tag{8.34}$$

dimana c adalah titik awal pada sumbu vertikal y. Perhatikan Gambar 8.7, garis-garis yang sejajar sumbu horisontal, $y = y_i$ dan garis-garis yang sejajar sumbu vertikal, $x = x_i$ disebut **grid lines**. Sementara titik-titik perpotongan antara garis-garis horisontal dan vertikal dinamakan



Gambar 8.7: Skema *grid lines* dan *mesh points* pada aplikasi metode *Finite-Difference*

mesh points.

Turunan kedua sebagaimana yang ada pada persamaan (8.32) dapat dinyatakan dalam rumus *centered-difference* sebagai berikut

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, y_j) \tag{8.35}$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} - \frac{k^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j) \tag{8.36}$$

Metode *Finite-Difference* biasanya mengabaikan suku yang terakhir, sehingga cukup dinyatakan sebagai

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} \tag{8.37}$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} \tag{8.38}$$

Pengabaian suku terakhir otomatis menimbulkan *error* yang dinamakan *truncation error*. Jadi, ketika suatu persamaan diferensial diolah secara numerik dengan metode *Finite-Difference*, maka solusinya pasti meleset alias keliru "sedikit", dikarenakan adanya *truncation error* tersebut. Akan tetapi, nilai *error* tersebut dapat ditolerir hingga batas-batas tertentu yang uraiannya akan dikupas pada bagian akhir bab ini.

Ok. Mari kita lanjutkan! Sekarang persamaan (8.37) dan (8.38) disubstitusi ke persamaan (8.32), hasilnya adalah

$$\frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} = f(x_i, y_j) \tag{8.39}$$

dimana $i = 1, 2, ..., n-1$ dan $j = 1, 2, ..., m-1$ dengan syarat batas sebagai berikut

$$u(x_0, y_j) = g(x_0, y_j) \qquad u(x_n, y_j) = g(x_n, y_j)$$
$$u(x_i, y_0) = g(x_i, y_0) \qquad u(x_i, y_m) = g(x_i, y_m)$$

Pengertian syarat batas disini adalah bagian tepi atau bagian pinggir dari susunan *mesh points*.

Pada metode *Finite-Difference*, persamaan (8.39) dinyatakan dalam notasi w, sebagai berikut

$$\begin{aligned}
\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} + \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{k^2} &= f(x_i, y_j) \\
w_{i+1,j} - 2w_{i,j} + w_{i-1,j} + \frac{h^2}{k^2}(w_{i,j+1} - 2w_{i,j} + w_{i,j-1}) &= h^2 f(x_i, y_j) \\
w_{i+1,j} - 2w_{i,j} + w_{i-1,j} + \frac{h^2}{k^2}w_{i,j+1} - 2\frac{h^2}{k^2}w_{i,j} + \frac{h^2}{k^2}w_{i,j-1} &= h^2 f(x_i, y_j) \\
-2[1 + \frac{h^2}{k^2}]w_{i,j} + (w_{i+1,j} + w_{i-1,j}) + \frac{h^2}{k^2}(w_{i,j+1} + w_{i,j-1}) &= h^2 f(x_i, y_j) \\
2[1 + \frac{h^2}{k^2}]w_{i,j} - (w_{i+1,j} + w_{i-1,j}) - \frac{h^2}{k^2}(w_{i,j+1} + w_{i,j-1}) &= -h^2 f(x_i, y_j) \qquad (8.40)
\end{aligned}$$

dimana $i = 1, 2, ..., n-1$ dan $j = 1, 2, ..., m-1$, dengan syarat batas sebagai berikut

$$\begin{aligned}
w_{0,j} = g(x_0, y_j) \qquad w_{n,j} = g(x_n, y_j) \qquad j &= 0, 1, ..., m-1; \\
w_{i,0} = g(x_i, y_0) \qquad w_{i,m} = g(x_i, y_m) \qquad i &= 1, 2, ..., n-1.
\end{aligned}$$

Persamaan (8.40) adalah rumusan akhir metode *Finite-Difference* untuk PDP Eliptik.

8.5.1 Contoh pertama

Misalnya kita diminta mensimulasikan distribusi panas pada lempengan logam berukuran $0,5$ m x $0,5$ m. Temperatur pada 2 sisi tepi lempengan logam dijaga pada $0°C$, sementara pada 2 sisi tepi lempengan logam yang lain, temperaturnya diatur meningkat secara linear dari $0°C$ hingga $100°C$. Problem ini memenuhi PDP Eliptik:

$$\frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = 0; \quad 0 < x < 0,5, \quad 0 < y < 0,5$$

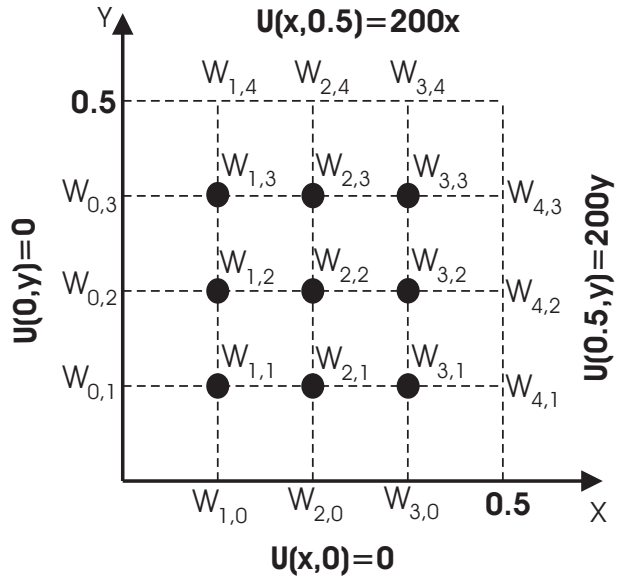
dengan syarat-syarat batas

$$u(0, y) = 0, \quad u(x, 0) = 0, \quad u(x, 0.5) = 200x, \quad u(0.5, y) = 200y$$

Jika $n = m = 4$ sedangkan ukuran lempeng logam adalah $0,5$ m x $0,5$ m, maka

$$h = \frac{0,5}{4} = 0,125 \qquad k = \frac{0,5}{4} = 0,125$$

Grid lines berikut *mesh points* dibuat berdasarkan nilai h dan k tersebut (lihat Gambar 8.8). Langkah berikutnya adalah menyusun persamaan *Finite-Difference*, dimulai dari persamaan



Gambar 8.8: Susunan *grid lines* dan *mesh points* untuk mensimulasikan distribusi temperatur pada lempeng logam sesuai contoh satu

asalnya (persamaan 8.40)

$$2[1+\frac{h^2}{k^2}]w_{i,j}-(w_{i+1,j}+w_{i-1,j})-\frac{h^2}{k^2}(w_{i,j+1}+w_{i,j-1})=-h^2f(x_i,y_j)$$

Karena $h=k=0,125$ dan $f(x_i,y_j)=0$, maka

$$4w_{i,j}-w_{i+1,j}-w_{i-1,j}-w_{i,j-1}-w_{i,j+1}=0 \tag{8.41}$$

Disisi lain, karena $n=4$, maka nilai i yang bervariasi $i=1,2,...,n-1$ akan menjadi $i=1,2,3$. Demikian hal-nya dengan j, karena $m=4$, maka variasi $j=1,2,...,m-1$ atau $j=1,2,3$. Dengan menerapkan persamaan (8.41) pada setiap *mesh point* yang belum diketahui temperaturnya, diperoleh

$$\begin{aligned}
4w_{1,3}-w_{2,3}-w_{1,2} &= w_{0,3}+w_{1,4}\\
4w_{2,3}-w_{3,3}-w_{2,2}-w_{1,3} &= w_{2,4}\\
4w_{3,3}-w_{3,2}-w_{2,3} &= w_{4,3}+w_{3,4}\\
4w_{1,2}-w_{2,2}-w_{1,1}-w_{1,3} &= w_{0,2}\\
4w_{2,2}-w_{3,2}-w_{2,1}-w_{1,2}-w_{2,3} &= 0\\
4w_{3,2}-w_{3,1}-w_{2,2}-w_{3,3} &= w_{4,2}\\
4w_{1,1}-w_{2,1}-w_{1,2} &= w_{0,1}+w_{1,0}\\
4w_{2,1}-w_{3,1}-w_{1,1}-w_{2,2} &= w_{2,0}\\
4w_{3,1}-w_{2,1}-w_{3,2} &= w_{3,0}+w_{4,1}
\end{aligned}$$

Semua notasi w yang berada diruas kanan tanda sama-dengan sudah ditentukan nilainya berdasarkan syarat batas, yaitu

$$\begin{aligned} w_{1,0} &= w_{2,0} = w_{3,0} = w_{0,1} = w_{0,2} = w_{0,3} = 0, \\ w_{1,4} &= w_{4,1} = 25, \quad w_{2,4} = w_{4,2} = 50, \quad \text{dan} \\ w_{3,4} &= w_{4,3} = 75 \end{aligned}$$

Dengan memasukkan syarat batas tersebut ke dalam sistem persamaan linear, maka

$$\begin{aligned} 4w_{1,3} - w_{2,3} - w_{1,2} &= 25 \\ 4w_{2,3} - w_{3,3} - w_{2,2} - w_{1,3} &= 50 \\ 4w_{3,3} - w_{3,2} - w_{2,3} &= 150 \\ 4w_{1,2} - w_{2,2} - w_{1,1} - w_{1,3} &= 0 \\ 4w_{2,2} - w_{3,2} - w_{2,1} - w_{1,2} - w_{2,3} &= 0 \\ 4w_{3,2} - w_{3,1} - w_{2,2} - w_{3,3} &= 50 \\ 4w_{1,1} - w_{2,1} - w_{1,2} &= 0 \\ 4w_{2,1} - w_{3,1} - w_{1,1} - w_{2,2} &= 0 \\ 4w_{3,1} - w_{2,1} - w_{3,2} &= 25 \end{aligned}$$

Kemudian dijadikan operasi perkalian matrik

$$\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} w_{1,3} \\ w_{2,3} \\ w_{3,3} \\ w_{1,2} \\ w_{2,2} \\ w_{3,2} \\ w_{1,1} \\ w_{2,1} \\ w_{3,1} \end{bmatrix} = \begin{bmatrix} 25 \\ 50 \\ 150 \\ 0 \\ 0 \\ 50 \\ 0 \\ 0 \\ 25 \end{bmatrix}$$

Mari kita perhatikan sejenak susunan elemen-elemen angka pada matrik berukuran 9x9 di atas. Terlihat jelas pada elemen diagonal selalu berisi angka 4. Ini sama sekali bukan ketidaksengajaan. Melainkan susunan itu sengaja direkayasa sedemikian rupa sehingga elemen-elemen tridiagonal terisi penuh oleh angka bukan 0 dan pada diagonal utamanya diletakkan angka yang terbesar. Metode Eliminasi Gauss dan Iterasi Gauss-Seidel telah diaplikasikan untuk menyelesaikan persamaan matrik di atas.

8.5.2 *Script* Matlab untuk PDP Elliptik

Inilah *script* Matlab yang dipakai untuk menghitung nila-nilai w menggunakan metode Eliminasi Gauss.

```
clear all
clc
n=9;
A=[ 4 -1  0 -1  0  0  0  0  0;
   -1  4 -1  0 -1  0  0  0  0;
    0 -1  4  0  0 -1  0  0  0;
   -1  0  0  4 -1  0 -1  0  0;
    0 -1  0 -1  4 -1  0 -1  0;
    0  0 -1  0 -1  4  0  0 -1;
    0  0  0 -1  0  0  4 -1  0;
    0  0  0  0 -1  0 -1  4 -1;
    0  0  0  0  0 -1  0 -1  4];

b=[25; 50; 150; 0; 0; 50; 0; 0; 25];

%&&&&&& Proses Eliminasi Gauss &&&&&&&&&&&&&&&&&&&&&&&&&&
%====== Menggabungkan Vektor b kedalam matrik A ========
%====== sehingga terbentuk matrik Augmentasi.   ========
for i=1:n
   A(i,n+1)=b(i,1);
end

%---------Proses Triangularisasi-----------
for j=1:(n-1)

%----mulai proses pivot---
        if (A(j,j)==0)
           for p=1:n+1
              u=A(j,p);
              v=A(j+1,p);
              A(j+1,p)=u;
              A(j,p)=v;
           end
        end
%----akhir proses pivot---
   jj=j+1;
   for i=jj:n
      m=A(i,j)/A(j,j);
      for k=1:(n+1)
         A(i,k)=A(i,k)-(m*A(j,k));
      end
   end
end
%----------------------------------------

%------Proses Substitusi mundur-------------
x(n,1)=A(n,n+1)/A(n,n);

for i=n-1:-1:1
   S=0;
   for j=n:-1:i+1
      S=S+A(i,j)*x(j,1);
   end
```

```
   x(i,1)=(A(i,n+1)-S)/A(i,i);
end
%&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
%===== Menampilkan Vektor w =================
w=x
```

Sementara berikut ini adalah *script* Matlab untuk menghitung nila-nilai w menggunakan metode Iterasi Gauss-Seidel.

```
clear all
clc

n=9;
A=[ 4 -1  0 -1  0  0  0  0  0;
   -1  4 -1  0 -1  0  0  0  0;
    0 -1  4  0  0 -1  0  0  0;
   -1  0  0  4 -1  0 -1  0  0;
    0 -1  0 -1  4 -1  0 -1  0;
    0  0 -1  0 -1  4  0  0 -1;
    0  0  0 -1  0  0  4 -1  0;
    0  0  0  0 -1  0 -1  4 -1;
    0  0  0  0  0 -1  0 -1  4];

b=[25; 50; 150; 0; 0; 50; 0; 0; 25];

%&&&&&&& ITERASI GAUSS-SEIDEL &&&&&&&&&&&&&&&&&&
itermax=100; %iterasi maksimum
%----nilai awal-----------
xl=[0; 0; 0; 0; 0; 0; 0; 0; 0];
xb=xl;
%----stopping criteria-----------
sc=0.001;
%----memulai iterasi-------------
for iterasi=1:itermax
   smtr1=0;
   for j=2:n
      smtr1=smtr1+A(1,j)*xl(j,1);
        end
   xb(1,1)=(-smtr1+b(1,1))/A(1,1);
   %-------------------------------------------
       for i=2:n-1
      smtr2=0;
      for j=i+1:n
                  smtr2=smtr2-A(i,j)*xl(j,1);
      end
      smtr3=0;
      for k=1:i-1
              smtr3=smtr3-A(i,k)*xb(k,1);
      end
      xb(i,1)=(smtr3+smtr2+b(i,1))/A(i,i);
        end
   %-------------------------------------------
   smtr4=0;
   for k=1:n-1
           smtr4=smtr4-A(n,k)*xb(k,1);
        end
        xb(n,1)=(smtr4+b(n,1))/A(n,n);
```

```
49    %------perhitungan norm2 -------------
50          s=0;
51          for i=1:n
52             s=s+(xb(i,1)-xl(i,1))^2;
53          end
54          epsilon=sqrt(s);
55          %-------------------------------------
56    xl=xb;
57    %------memeriksa stopping criteria--------
58    if epsilon<sc
59       w=xb
60       break
61    end
62    %----------------------------------------
63 end
```

Tabel berikut memperlihatkan hasil pemrosesan dengan metode Eliminasi Gauss (disingkat: EG) dan iterasi Gauss-Seidel (disingkat: GS)

	$w_{1,3}$	$w_{2,3}$	$w_{3,3}$	$w_{1,2}$	$w_{2,2}$	$w_{3,2}$	$w_{1,1}$	$w_{2,1}$	$w_{3,1}$
EG	18.7500	37.5000	56.2500	12.5000	25.0000	37.5000	6.2500	12.5000	18.7500
GS	18.7497	37.4997	56.2498	12.4997	24.9997	37.4998	6.2498	12.4998	18.7499

Inilah solusi yang ditawarkan oleh *Finite-Difference*. Kalau diamati dengan teliti, angka-angka distribusi temperatur pada 9 buah *mesh points* memang logis dan masuk akal. Dalam kondisi riil, mungkin kondisi seperti ini hanya bisa terjadi bila lempengan logam tersebut terbuat dari bahan yang homogen.

Hasil EG dan GS memang berbeda, walaupun perbedaannya tidak *significant*. Namun perlu saya tegaskan disini bahwa jika sistem persamaan linear yang diperoleh dari *Finite Difference* berorde 100 atau kurang dari itu, maka lebih baik memilih metode Eliminasi Gauss sebagai langkah penyelesaian akhir. Alasannya karena, *direct method* seperti eliminasi Gauss, lebih stabil dibandingkan metode iterasi. Tapi jika orde-nya lebih dari 100, disarankan memilih metode iterasi seperti iterasi Gauss-Seidel, atau menggunakan metode SOR yang terbukti lebih efisien dibanding Gauss-Seidel. Jika matrik A bersifat *positive definite*, metode *Court Factorization* adalah pilihan yg paling tepat karena metode ini sangat efisien sehingga bisa menghemat memori komputer.

8.5.3 Contoh kedua

Diketahui persamaan poisson sebagai berikut

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = xe^y, \quad 0 < x < 2, \quad 0 < y < 1,$$

dengan syarat batas

$$u(0,y) = 0, \quad u(2,y) = 2e^y, \quad 0 \le y \le 1,$$
$$u(x,0) = x, \quad u(x,1) = ex, \quad 0 \le x \le 2,$$

Solusi numerik dihitung dengan pendekatan finite-difference gauss-seidel dimana batas toleransi kesalahan ditentukan

$$\left| w_{ij}^{(l)} - w_{ij}^{(l-1)} \right| \le 10^{-10}$$

8.6 PDP parabolik

PDP parabolik yang kita pelajari disini adalah persamaan difusi

$$\frac{\partial u}{\partial t}(x,t) = \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t), \quad 0 < x < \ell, \quad t > 0, \tag{8.42}$$

yang berlaku pada kondisi

$$u(0,t) = u(\ell,t) = 0, \quad t > 0,$$

dan

$$u(x,0) = f(x), \quad 0 \le x \le \ell,$$

dimana t dalam dimensi waktu, sementara x berdimensi jarak.

8.6.1 Metode *Forward-difference*

Solusi numerik diperoleh menggunakan **forward-difference**[2] dengan langkah-langkah yang hampir mirip seperti yang telah dibahas pada PDP eliptik. Langkah pertama adalah menentukan sebuah angka $m > 0$, yang dengannya, nilai h ditentukan oleh rumus $h = \ell/m$. Langkah kedua adalah menentukan ukuran *time-step* k dimana $k > 0$. Adapun *mesh points* ditentukan oleh (x_i, t_j), dimana $x_i = ih$, dengan $i = 0, 1, 2, ..., m$, dan $t_j = jk$ dengan $j = 0, 1, ...$

Berdasarkan deret Taylor, turunan pertama persamaan (8.42) terhadap t, dengan *time step* k, adalah

$$\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_j + k) - u(x_i, t_j)}{k} - \frac{k}{2}\frac{\partial^2 u}{\partial t^2}(x_i, \mu_j) \tag{8.43}$$

Namun, sebagaimana pendekatan *finite-difference* pada umumnya, pendekatan *forward-difference* selalu mengabaikan suku terakhir, sehingga persamaan di atas ditulis seperti ini

$$\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_j + k) - u(x_i, t_j)}{k} \tag{8.44}$$

[2]Pada Bab ini ada beberapa istilah yang masing-masing menggunakan kata *difference*, yaitu *finite difference*, *forward difference*, *centered difference* dan *backward difference*. Setiap istilah punya arti yang berbeda.

Sementara itu, turunan kedua persamaan (8.42) terhadap x berdasarkan deret Taylor adalah

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) = \frac{u(x_i + h, t_J) - 2u(x_i, t_j) + u(x_i - h, t_J)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, t_j) \tag{8.45}$$

Pengabaian suku terakhir menjadikan persamaan di atas ditulis kembali sebagai berikut

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) = \frac{u(x_i + h, t_j) - 2u(x_i, t_j) + u(x_i - h, t_j)}{h^2} \tag{8.46}$$

Kemudian persamaan (8.44) dan (8.46) disubstitusi kedalam persamaan (8.42), maka diperoleh

$$\frac{u(x_i, t_j + k) - u(x_i, t_j)}{k} = \alpha^2 \frac{u(x_i + h, t_j) - 2u(x_i, t_j) + u(x_i - h, t_j)}{h^2} \tag{8.47}$$

atau dapat dinyatakan dalam notasi w

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0 \tag{8.48}$$

Dari sini diperoleh solusi untuk $w_{i,j+1}$, yaitu

$$w_{i,j+1} = \left(1 - \frac{2\alpha^2 k}{h^2}\right) w_{i,j} + \alpha^2 \frac{k}{h^2}(w_{i+1,j} + w_{i-1,j}) \tag{8.49}$$

jika

$$\lambda = \frac{\alpha^2 k}{h^2} \tag{8.50}$$

maka

$$(1 - 2\lambda) w_{i,j} + \lambda w_{i+1,j} + \lambda w_{i-1,j} = w_{i,j+1} \tag{8.51}$$

8.6.2 Contoh ketiga: *One dimensional heat equation*

Misalnya diketahui, distribusi panas satu dimensi (1D) sebagai fungsi waktu (t) pada sebatang logam memenuhi persamaan berikut

$$\frac{\partial u}{\partial t}(x,t) - \frac{\partial^2 u}{\partial x^2}(x,t) = 0, \quad 0 < x < 1 \quad 0 \le t,$$

dengan syarat batas

$$u(0,t) = u(1,t) = 0, \quad 0 < t,$$

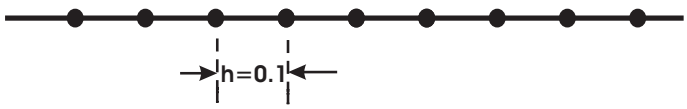
dan kondisi mula-mula

$$u(x,0) = \sin(\pi x), \quad 0 \le x \le 1,$$

Solusi analitik atas masalah ini adalah

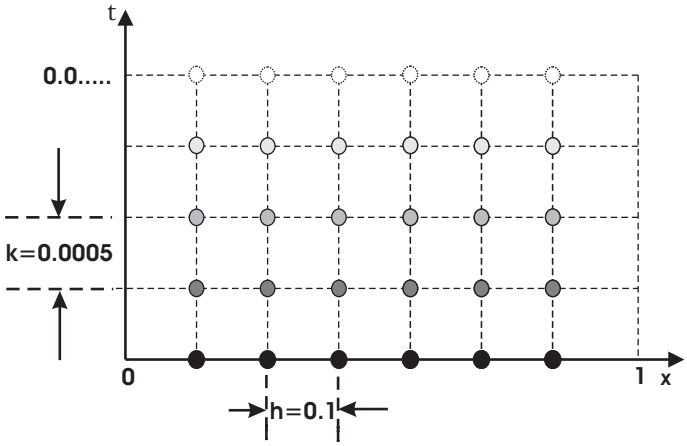
$$u(x,t) = e^{-\pi^2 t}\sin(\pi x)$$

Adapun sebaran posisi *mesh-points* dalam 1-D diperlihatkan pada Gambar 8.9. Sementara



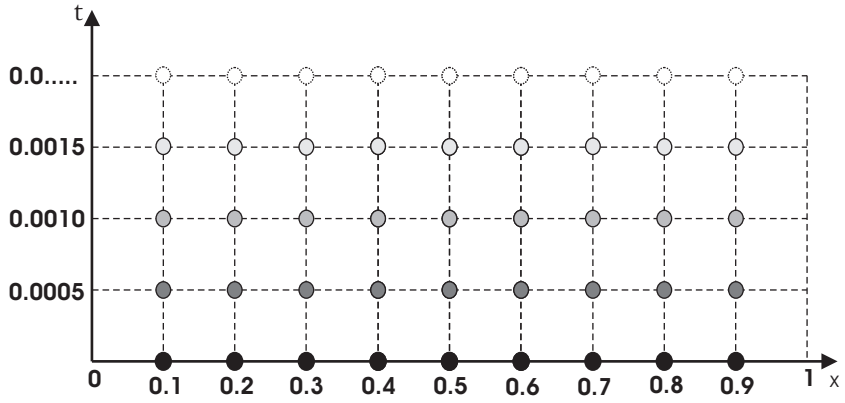
Gambar 8.9: Sebatang logam dengan posisi titik-titik simulasi (*mesh-points*) distribusi temperatur. Jarak antar titik ditentukan sebesar $h = 0,1$.

Gambar 8.10 melengkapi Gambar 8.9, dimana perubahan waktu tercatat setiap interval $k = 0,0005$. Sepintas Gambar 8.10 terlihat seolah-olah obyek yang mau disimulasikan berbentuk 2-dimensi, padahal bendanya tetap 1-dimensi yaitu hanya sebatang logam.



Gambar 8.10: Interval *mesh-points* dengan jarak $h = 0,1$ dalam interval waktu $k = 0,0005$

Selanjutnya, Gambar 8.11 memperlihatkan tepi-tepi syarat batas yaitu angka 0 di ujung kiri dan angka 1 di ujung kanan pada sumbu horisontal x. Diantara batas-batas itu terdapat sebaran titik simulasi berjarak $h = 0,1$. Sementara, sumbu vertikal menunjukan perubahan dari waktu ke waktu dengan interval $k = 0,0005$. Karena $\alpha = 1$, $h = 0,1$ dan $k = 0,0005$ maka



Gambar 8.11: Posisi *mesh-points*. Arah x menunjukkan posisi titik-titik yang dihitung dengan *forward-difference*, sedangkan arah t menunjukkan perubahan waktu yg makin meningkat

λ dapat dihitung dengan persamaan (8.50)

$$\lambda = \frac{\alpha^2 k}{h^2} = \frac{0,1}{0,0005^2} = 0,05$$

Berdasarkan persamaan (8.51), sistem persamaan linear dapat disusun sebagai berikut

$$\begin{aligned}
0,9w_{1,j} + 0,5w_{2,j} &= w_{1,j+1} - 0,5w_{0,j} \\
0,9w_{2,j} + 0,5w_{3,j} + 0,5w_{1,j} &= w_{2,j+1} \\
0,9w_{3,j} + 0,5w_{4,j} + 0,5w_{2,j} &= w_{3,j+1} \\
0,9w_{4,j} + 0,5w_{5,j} + 0,5w_{3,j} &= w_{4,j+1} \\
0,9w_{5,j} + 0,5w_{6,j} + 0,5w_{4,j} &= w_{5,j+1} \\
0,9w_{6,j} + 0,5w_{7,j} + 0,5w_{5,j} &= w_{6,j+1} \\
0,9w_{7,j} + 0,5w_{8,j} + 0,5w_{6,j} &= w_{7,j+1} \\
0,9w_{8,j} + 0,5w_{9,j} + 0,5w_{7,j} &= w_{8,j+1} \\
0,9w_{9,j} + 0,5w_{8,j} &= w_{9,j+1} - 0,5w_{10,j}
\end{aligned}$$

Syarat batas menetapkan bahwa $w_{0,j} = w_{10,j} = 0$. Lalu dinyatakan dalam bentuk operasi matrik

$$\begin{bmatrix}
0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9
\end{bmatrix}
\begin{bmatrix} w_{1,j} \\ w_{2,j} \\ w_{3,j} \\ w_{4,j} \\ w_{5,j} \\ w_{6,j} \\ w_{7,j} \\ w_{8,j} \\ w_{9,j} \end{bmatrix}
=
\begin{bmatrix} w_{1,j+1} \\ w_{2,j+1} \\ w_{3,j+1} \\ w_{4,j+1} \\ w_{5,j+1} \\ w_{6,j+1} \\ w_{7,j+1} \\ w_{8,j+1} \\ w_{9,j+1} \end{bmatrix} \tag{8.52}$$

Persamaan matriks di atas dapat direpresentasikan sebagai

$$A\mathbf{w}^{(j)} = \mathbf{w}^{(j+1)} \tag{8.53}$$

Proses perhitungan dimulai dari $j = 0$. Persamaan matrik menjadi

$$\begin{bmatrix}
0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9
\end{bmatrix}
\begin{bmatrix} w_{1,0} \\ w_{2,0} \\ w_{3,0} \\ w_{4,0} \\ w_{5,0} \\ w_{6,0} \\ w_{7,0} \\ w_{8,0} \\ w_{9,0} \end{bmatrix}
=
\begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \\ w_{4,1} \\ w_{5,1} \\ w_{6,1} \\ w_{7,1} \\ w_{8,1} \\ w_{9,1} \end{bmatrix}$$

Nilai $w_{1,0}, w_{2,0}, ..., w_{9,0}$ sudah ditentukan oleh kondisi awal, yaitu

$$u(x,0) = \sin \pi x, \quad 0 \leq x \leq 1,$$

Jika $h = 0,1$, maka $x_1 = h = 0,1$; $x_2 = 2h = 0,2$; $x_3 = 3h = 0,3$;....; $x_9 = 9h = 0,9$. Lalu masing-masing dimasukkan ke $\sin \pi x$ untuk mendapatkan nilai $u(x,0)$. Kemudian notasi $u(x,0)$ diganti dengan notasi w yang selanjutnya dinyatakan sebagai berikut: $w_{1,0} = u(x_1,0) = u(0.1,0) = \sin \pi(0.1) = 0,3090$. Dengan cara yang sama: $w_{2,0} = 0,5878$; $w_{3,0} = 0,8090$; $w_{4,0} = 0,9511$; $w_{5,0} = 1,0000$; $w_{6,0} = 0,9511$; $w_{7,0} = 0,8090$; $w_{8,0} = 0,5878$; dan $w_{9,0} = 0,3090$. Maka persamaan matriks menjadi

$$\begin{bmatrix} 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 \end{bmatrix} \begin{bmatrix} 0,3090 \\ 0,5878 \\ 0,8090 \\ 0,9511 \\ 1,0000 \\ 0,9511 \\ 0,8090 \\ 0,5878 \\ 0,3090 \end{bmatrix} = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \\ w_{4,1} \\ w_{5,1} \\ w_{6,1} \\ w_{7,1} \\ w_{8,1} \\ w_{9,1} \end{bmatrix}$$

Ini hanya perkalian matrik biasa [3]. Hasil perkalian itu adalah: $w_{1,1} = 0,3075$; $w_{2,1} = 0,5849$; $w_{3,1} = 0,8051$; $w_{4,1} = 0,9464$; $w_{5,1} = 0,9951$; $w_{6,1} = 0,9464$; $w_{7,1} = 0,8051$; $w_{8,1} = 0,5849$; dan $w_{9,1} = 0,3075$. Semua angka ini adalah nilai temperatur kawat di masing-masing *mesh points* setelah selang waktu $0,0005$ detik[4].

Selanjutnya, hasil ini diumpankan lagi ke persamaan matriks yang sama untuk mendapatkan $w_{x,2}$

$$\begin{bmatrix} 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 \end{bmatrix} \begin{bmatrix} 0,3075 \\ 0,5849 \\ 0,8051 \\ 0,9464 \\ 0,9951 \\ 0,9464 \\ 0,8051 \\ 0,5849 \\ 0,3075 \end{bmatrix} = \begin{bmatrix} w_{1,2} \\ w_{2,2} \\ w_{3,2} \\ w_{4,2} \\ w_{5,2} \\ w_{6,2} \\ w_{7,2} \\ w_{8,2} \\ w_{9,2} \end{bmatrix}$$

Perhitungan dengan cara seperti ini diulang-ulang sampai mencapai waktu maksimum. Jika waktu maksimum adalah $T = 0,5$ detik, berarti mesti dilakukan 1000 kali iterasi[5]. Untuk

[3] Topik tentang perkalian matrik sudah diulas pada Bab 1

[4] karena *step time* k-nya sudah ditentukan sebesar $0,0005$

[5] cara menghitung jumlah iterasi: $T/k = 0,5/0,0005 = 1000$

sampai 1000 kali, maka indeks j bergerak dari 1 sampai 1000. Dengan bantuan script Matlab, proses perhitungan menjadi sangat singkat.

8.6.2.1 *Script Forward-Difference*

Script matlab *Forward-Difference* untuk menyelesaikan contoh masalah ini, dimana $h = 0,1$ dan $k = 0,0005$

```
clear all
clc

n=9;
alpha=1.0;
k=0.0005;
h=0.1;
lambda=(alpha^2)*k/(h^2);

% Kondisi awal
for i=1:n
   suhu(i)=sin(pi*i*0.1);
end

%Mengcopy kondisi awal ke w
for i=1:n
   w0(i,1)=suhu(i);
end

A=[ (1-2*lambda) lambda   0  0  0  0  0  0  0;
lambda (1-2*lambda) lambda  0  0  0  0  0  0;
0 lambda (1-2*lambda) lambda  0  0  0  0  0 ;
0 0 lambda (1-2*lambda) lambda  0  0  0  0;
0 0 0 lambda (1-2*lambda) lambda  0  0  0;
0 0 0 0 lambda (1-2*lambda) lambda  0  0;
0 0 0 0 0 lambda (1-2*lambda) lambda  0 ;
0 0 0 0 0 0 lambda (1-2*lambda) lambda ;
0 0 0 0 0 0 0 lambda (1-2*lambda) ];

iterasi=1000;
for k=1:iterasi
        disp('perkalian matriks')
        %====================================
        for i=1:n
           w(i,1)=0.0;
        end

        for i=1:n
                for j=1:n
                 w(i,1)=w(i,1)+A(i,j)*w0(j,1);
                 end
        end
        %====================================
        w
        w0=w;
end
```

Tabel 8.4 memperlihatkan hasil perhitungan yang diulang-ulang hingga 1000 kali. Tabel tersebut juga menunjukkan hasil perbandingan antara pemilihan nilai interval $k = 0,0005$ dan $k = 0,01$. Tabel ini menginformasikan satu hal penting, yaitu pada saat interval $k = 0,0005$, *forward-difference* berhasil mencapai konvergensi yang sangat baik. Namun pada saat interval $k = 0.01$, dengan jumlah iterasi hanya 50 kali untuk mencapai *time* maksimum $0,5$ detik, terlihat jelas hasil *forward-difference* tidak konvergen (Bandingkan kolom ke-4 dan kolom ke-6!), dan ini dianggap bermasalah. Masalah ini bisa diatasi dengan metode *backward-difference*.

Tabel 8.4: Hasil simulasi distribusi panas bergantung waktu dalam 1-dimensi. Kolom ke-2 adalah solusi analitik/exact, kolom ke-3 dan ke-5 adalah solusi numerik *forward-difference*. Kolom ke-4 dan ke-6 adalah selisih antara solusi analitik dan numerik

x_i	$u(x_i, 0.5)$	$w_{i,1000}$ $k = 0,0005$	$\lvert u(x_i, 0.5) - w_{i,1000}\rvert$	$w_{i,50}$ $k = 0,01$	$\lvert u(x_i, 0.5) - w_{i,50}\rvert$
0,0	0	0		0	
0,1	0,00222241	0,00228652	$6,411 \times 10^{-5}$	$8,19876 \times 10^{7}$	$8,199 \times 10^{7}$
0,2	0,00422728	0,00434922	$1,219 \times 10^{-4}$	$-1,55719 \times 10^{8}$	$1,557 \times 10^{8}$
0,3	0,00581836	0,00598619	$1,678 \times 10^{-4}$	$2,13833 \times 10^{8}$	$2,138 \times 10^{8}$
0,4	0,00683989	0,00703719	$1,973 \times 10^{-4}$	$-2,50642 \times 10^{8}$	$2,506 \times 10^{8}$
0,5	0,00719188	0,00739934	$2,075 \times 10^{-4}$	$2,62685 \times 10^{8}$	$2,627 \times 10^{8}$
0,6	0,00683989	0,00703719	$1,973 \times 10^{-4}$	$-2,49015 \times 10^{8}$	$2,490 \times 10^{8}$
0,7	0,00581836	0,00598619	$1,678 \times 10^{-4}$	$2,11200 \times 10^{8}$	$2,112 \times 10^{8}$
0,8	0,00422728	0,00434922	$1,219 \times 10^{-4}$	$-1,53086 \times 10^{8}$	$1,531 \times 10^{8}$
0,9	0,00222241	0,00228652	$6,511 \times 10^{-5}$	$8,03604 \times 10^{7}$	$8,036 \times 10^{7}$
1,0	0	0		0	

8.6.3 Metode *Backward-difference*

Kalau kita ulang lagi pelajaran yang lalu tentang *forward-difference*, kita akan dapatkan formula *forward-difference* adalah sebagai berikut (lihat persamaan (8.48))

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0$$

Sekarang, dengan sedikit modifikasi, formula *backward-difference* dinyatakan sebagai

$$\frac{w_{i,j} - w_{i,j-1}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0 \tag{8.54}$$

jika ditetapkan

$$\lambda = \frac{\alpha^2 k}{h^2}$$

maka *backward-difference* disederhanakan menjadi

$$(1 + 2\lambda)\, w_{i,j} - \lambda w_{i+1,j} - \lambda w_{i-1,j} = w_{i,j-1} \tag{8.55}$$

coba sejenak anda bandingkan dengan formula *forward-difference* dalam λ sebagaimana dinyatakan oleh persamaan (8.51)

$$(1-2\lambda)\, w_{i,j} + \lambda w_{i+1,j} + \lambda w_{i-1,j} = w_{i,j+1}$$

O.K., mari kita kembali ke contoh soal kita yang tadi, dimana ada perubahan nilai k yang semula $k = 0,0005$ menjadi $k = 0,01$. Sementara α dan h nilainya tetap. Maka λ dapat dihitung dengan persamaan (8.50) kembali

$$\lambda = \frac{\alpha^2 k}{h^2} = \frac{0,1}{0,01^2} = 1$$

Berdasarkan persamaan (8.55), sistem persamaan linear mengalami sedikit perubahan

$$\begin{aligned}
3w_{1,j} - 1w_{2,j} &= w_{1,j-1} + 1w_{0,j} \\
3w_{2,j} - 1w_{3,j} - 1w_{1,j} &= w_{2,j-1} \\
3w_{3,j} - 1w_{4,j} - 1w_{2,j} &= w_{3,j-1} \\
3w_{4,j} - 1w_{5,j} - 1w_{3,j} &= w_{4,j-1} \\
3w_{5,j} - 1w_{6,j} - 1w_{4,j} &= w_{5,j-1} \\
3w_{6,j} - 1w_{7,j} - 1w_{5,j} &= w_{6,j-1} \\
3w_{7,j} - 1w_{8,j} - 1w_{6,j} &= w_{7,j-1} \\
3w_{8,j} - 1w_{9,j} - 1w_{7,j} &= w_{8,j-1} \\
3w_{9,j} - 1w_{8,j} &= w_{9,j-1} + 1w_{10,j}
\end{aligned}$$

Syarat batas masih sama, yaitu $w_{0,j} = w_{10,j} = 0$. Lalu jika dinyatakan dalam bentuk operasi matrik

$$\begin{bmatrix}
3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3
\end{bmatrix}
\begin{bmatrix} w_{1,j} \\ w_{2,j} \\ w_{3,j} \\ w_{4,j} \\ w_{5,j} \\ w_{6,j} \\ w_{7,j} \\ w_{8,j} \\ w_{9,j} \end{bmatrix}
=
\begin{bmatrix} w_{1,j-1} \\ w_{2,j-1} \\ w_{3,j-1} \\ w_{4,j-1} \\ w_{5,j-1} \\ w_{6,j-1} \\ w_{7,j-1} \\ w_{8,j-1} \\ w_{9,j-1} \end{bmatrix}$$

Persamaan matriks di atas dapat direpresentasikan sebagai

$$A\mathbf{w}^{(j)} = \mathbf{w}^{(j-1)} \tag{8.56}$$

Perhitungan dimulai dari iterasi pertama, dimana $j = 1$

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \\ w_{4,1} \\ w_{5,1} \\ w_{6,1} \\ w_{7,1} \\ w_{8,1} \\ w_{9,1} \end{bmatrix} = \begin{bmatrix} w_{1,0} \\ w_{2,0} \\ w_{3,0} \\ w_{4,0} \\ w_{5,0} \\ w_{6,0} \\ w_{7,0} \\ w_{8,0} \\ w_{9,0} \end{bmatrix}$$

Dengan memasukan kondisi awal, ruas kanan menjadi

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{3,1} \\ w_{4,1} \\ w_{5,1} \\ w_{6,1} \\ w_{7,1} \\ w_{8,1} \\ w_{9,1} \end{bmatrix} = \begin{bmatrix} 0,3090 \\ 0,5878 \\ 0,8090 \\ 0,9511 \\ 1,0000 \\ 0,9511 \\ 0,8090 \\ 0,5878 \\ 0,3090 \end{bmatrix}$$

Berbeda dengan operasi matrik *forward difference*, operasi matrik *backward difference* ini bukan perkalian matrik biasa. Operasi matrik tersebut akan dipecahkan oleh metode Eliminasi Gauss[6]. Untuk jumlah iterasi hingga $j = 50$, perhitungannya dilakukan dalam *script* Matlab.

8.6.3.1 *Script Backward-Difference* dengan Eliminasi Gauss

```
clear all
clc

n=9;
alpha=1.0;
k=0.01;
h=0.1;
lambda=(alpha^2)*k/(h^2);

%Kondisi awal
for i=1:n
   suhu(i)=sin(pi*i*0.1);
end

%Mengcopy kondisi awal ke w
for i=1:n
```

[6]Uraian tentang metode Eliminasi Gauss tersedia di Bab 2

```
   w0(i,1)=suhu(i);
end

AA=[ (1+2*lambda) -lambda   0  0  0  0  0  0  0;
           -lambda (1+2*lambda) -lambda  0  0  0  0  0  0;
    0  -lambda (1+2*lambda) -lambda  0  0  0  0  0 ;
0 0 -lambda (1+2*lambda) -lambda  0  0  0  0;
0 0 0 -lambda (1+2*lambda) -lambda  0  0  0;
0 0 0 0 -lambda (1+2*lambda) -lambda  0  0;
0 0 0 0 0 -lambda (1+2*lambda) -lambda  0 ;
0 0 0 0 0 0 -lambda (1+2*lambda) -lambda ;
0 0 0 0 0 0 0 -lambda (1+2*lambda) ];

iterasi=50;
for i=1:iterasi
   %&&&&&& Proses Eliminasi Gauss &&&&&&&&&&&&&&&&&&&&&&&&&
   A=AA; %Matriks Backward Difference dicopy supaya fix

   for i=1:n
           A(i,n+1)=w0(i,1);
        end

        %---------Proses Triangularisasi-----------
        for j=1:(n-1)

                %----mulai proses pivot---
                if (A(j,j)==0)
                   for p=1:n+1
                      u=A(j,p);
                      v=A(j+1,p);
                      A(j+1,p)=u;
                      A(j,p)=v;
                   end
      end
                %----akhir proses pivot---
           jj=j+1;
           for i=jj:n
              m=A(i,j)/A(j,j);
              for k=1:(n+1)
                 A(i,k)=A(i,k)-(m*A(j,k));
              end
           end
        end
        %-----------------------------------------

        %------Proses Substitusi mundur-------------
        w(n,1)=A(n,n+1)/A(n,n);

        for i=n-1:-1:1
           S=0;
           for j=n:-1:i+1
              S=S+A(i,j)*w(j,1);
           end
           w(i,1)=(A(i,n+1)-S)/A(i,i);
        end
        %&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
   w0=w;
end
w
```

Hasilnya menunjukkan bahwa kinerja metode *backward-difference* lebih baik dibanding metode *forward-difference*, ini ditunjukkan dari selisih yang relatif kecil antara solusi numerik dan solusi analitik, sebagaimana bisa terlihat dari kolom ke-4 pada tabel berikut

Tabel 8.5: Hasil simulasi distribusi panas bergantung waktu dalam 1-dimensi dengan metode *backward-difference* dimana $k = 0,01$

x_i	$u(x_i, 0.5)$	$w_{i,50}$	$\lvert u(x_i, 0.5) - w_{i,50} \rvert$
0,0	0	0	
0,1	0,00222241	0,00289802	$6,756 \times 10^{-4}$
0,2	0,00422728	0,00551236	$1,285 \times 10^{-3}$
0,3	0,00581836	0,00758711	$1,769 \times 10^{-3}$
0,4	0,00683989	0,00891918	$2,079 \times 10^{-3}$
0,5	0,00719188	0,00937818	$2,186 \times 10^{-3}$
0,6	0,00683989	0,00891918	$2,079 \times 10^{-3}$
0,7	0,00581836	0,00758711	$1,769 \times 10^{-3}$
0,8	0,00422728	0,00551236	$1,285 \times 10^{-3}$
0,9	0,00222241	0,00289802	$6,756 \times 10^{-4}$
1,0	0	0	

8.6.4 Metode Crank-Nicolson

Metode ini dimunculkan disini karena metode ini memiliki performa yang lebih unggul dari dua metode sebelumnya. Namun begitu pondasi metode Crank-Nicolson terdiri atas metode *Forward-Difference* dan metode *Backward-Difference*. Mari kita ulang lagi pelajaran yang sudah kita lewati. Formula *Forward-Difference* adalah

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0$$

sedangkan *Backward-Difference* adalah

$$\frac{w_{i,j} - w_{i,j-1}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0$$

Ketika *Backward-Difference* berada pada iterasi ke $j + 1$, maka

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j+1} - 2w_{i,j+1} + w_{i-1,j+1}}{h^2} = 0 \tag{8.57}$$

Jika formula ini dijumlahkan dengan formula *forward-difference*, kemudian hasilnya dibagi 2, maka akan diperoleh

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \frac{\alpha^2}{2}\left[\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} + \frac{w_{i+1,j+1} - 2w_{i,j+1} + w_{i-1,j+1}}{h^2}\right] = 0 \tag{8.58}$$

inilah formula Crank-Nicolson. Adapun λ tetap dinyatakan sebagai

$$\lambda = \frac{\alpha^2 k}{h^2}$$

maka

$$w_{i,j+1} - w_{i,j} - \frac{\lambda}{2}\left[w_{i+1,j} - 2w_{i,j} + w_{i-1,j} + w_{i+1,j+1} - 2w_{i,j+1} + w_{i-1,j+1}\right] = 0$$

$$w_{i,j+1} - w_{i,j} - \frac{\lambda}{2}w_{i+1,j} + \lambda w_{i,j} - \frac{\lambda}{2}w_{i-1,j} - \frac{\lambda}{2}w_{i+1,j+1} + \lambda w_{i,j+1} - \frac{\lambda}{2}w_{i-1,j+1} = 0$$

$$-\frac{\lambda}{2}w_{i-1,j+1} + w_{i,j+1} + \lambda w_{i,j+1} - \frac{\lambda}{2}w_{i+1,j+1} - \frac{\lambda}{2}w_{i-1,j} - w_{i,j} + \lambda w_{i,j} - \frac{\lambda}{2}w_{i+1,j} = 0$$

$$-\frac{\lambda}{2}w_{i-1,j+1} + w_{i,j+1} + \lambda w_{i,j+1} - \frac{\lambda}{2}w_{i+1,j+1} = \frac{\lambda}{2}w_{i-1,j} + w_{i,j} - \lambda w_{i,j} + \frac{\lambda}{2}w_{i+1,j}$$

dan akhirnya

$$-\frac{\lambda}{2}w_{i-1,j+1} + (1+\lambda)w_{i,j+1} - \frac{\lambda}{2}w_{i+1,j+1} = \frac{\lambda}{2}w_{i-1,j} + (1-\lambda)w_{i,j} + \frac{\lambda}{2}w_{i+1,j} \tag{8.59}$$

Dalam bentuk persamaan matrik dinyatakan sebagai

$$A\mathbf{w}^{(j+1)} = B\mathbf{w}^{(j)}, \qquad \text{untuk } j = 0, 1, 2, ... \tag{8.60}$$

Dengan menggunakan contoh soal yang sama, yang sebelumnya telah diselesaikan dengan metode *Forward-Difference* dan *Backward-Difference*, maka penyelesaian soal tersebut dengan metode Crank-Nicolson juga akan didemonstrasikan di sini. Dengan nilai $k = 0,01$; $h = 0,1$; $\lambda = 1$ dan berdasarkan persamaan (8.59) diperoleh

$$-0,5w_{i-1,j+1} + 2w_{i,j+1} - 0,5w_{i+1,j+1} = 0,5w_{i-1,j} + 0w_{i,j} + 0,5w_{i+1,j}$$

Script Matlab untuk menyelesaikan persamaan ini adalah

```
clear all
clc

n=9;
iterasi=50;
alpha=1.0;
k=0.01;
h=0.1;
lambda=(alpha^2)*k/(h^2);

%Kondisi awal
for i=1:n
   suhu(i)=sin(pi*i*0.1);
end

%Mengcopy kondisi awal ke w
for i=1:n
   w0(i,1)=suhu(i);
```

```
end

AA=[(1+lambda) -lambda/2 0 0 0 0 0 0;
    -lambda/2 (1+lambda) -lambda/2 0 0 0 0 0;
    0 -lambda/2 (1+lambda) -lambda/2 0 0 0 0;
    0 0 -lambda/2 (1+lambda) -lambda/2 0 0 0;
    0 0 0 -lambda/2 (1+lambda) -lambda/2 0 0;
    0 0 0 0 -lambda/2 (1+lambda) -lambda/2 0;
    0 0 0 0 0 -lambda/2 (1+lambda) -lambda/2;
    0 0 0 0 0 0 -lambda/2 (1+lambda)];

B=[(1-lambda) lambda/2 0 0 0 0 0 0;
    lambda/2 (1-lambda) lambda/2 0 0 0 0 0;
    0 lambda/2 (1-lambda) lambda/2 0 0 0 0;
    0 0 lambda/2 (1-lambda) lambda/2 0 0 0;
    0 0 0 lambda/2 (1-lambda) lambda/2 0 0;
    0 0 0 0 lambda/2 (1-lambda) lambda/2 0;
    0 0 0 0 0 lambda/2 (1-lambda) lambda/2;
    0 0 0 0 0 0 lambda/2 (1-lambda)];

 iterasi=50;
 for iter=1:iterasi

   %===perkalian matriks===================
        for i=1:n
           b(i,1)=0.0;
        end
   for i=1:n
      for j=1:n
         b(i,1)=b(i,1)+B(i,j)*w0(j,1);
                end
   end
   %=======================================

   %&&&&&& Proses Eliminasi Gauss &&&&&&&&&&&&&&&&&&&&&&&&&
   A=AA; %Matriks Backward Difference dicopy supaya fix

   for i=1:n
           A(i,n+1)=b(i,1);
        end

        %---------Proses Triangularisasi-----------
        for j=1:(n-1)

                %----mulai proses pivot---
                if (A(j,j)==0)
                   for p=1:n+1
                      u=A(j,p);
                      v=A(j+1,p);
                      A(j+1,p)=u;
                      A(j,p)=v;
                   end
      end
                %----akhir proses pivot---
           jj=j+1;
           for i=jj:n
              m=A(i,j)/A(j,j);
```

```
78                for k=1:(n+1)
79                   A(i,k)=A(i,k)-(m*A(j,k));
80                end
81             end
82          end
83          %----------------------------------------
84
85          %------Proses Substitusi mundur-------------
86          w(n,1)=A(n,n+1)/A(n,n);
87
88          for i=n-1:-1:1
89             S=0;
90             for j=n:-1:i+1
91                S=S+A(i,j)*w(j,1);
92             end
93             w(i,1)=(A(i,n+1)-S)/A(i,i);
94          end
95          %&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
96       w0=w;
97    end
98    iter
99    w
```

Tabel 8.6: Hasil simulasi distribusi panas bergantung waktu (t) dalam 1-dimensi dengan metode *backward-difference* dan Crank-Nicolson

		BD	CN	*Backward-Diff*	Crank-Nicolson
x_i	$u(x_i, 0.5)$	$w_{i,50}$	$w_{i,50}$	$\lvert u(x_i, 0.5) - w_{i,50}\rvert$	$\lvert u(x_i, 0.5) - w_{i,50}\rvert$
0,0	0	0	0		
0,1	0,00222241	0,00289802	0,00230512	$6,756 \times 10^{-4}$	$8,271 \times 10^{-5}$
0,2	0,00422728	0,00551236	0,00438461	$1,285 \times 10^{-3}$	$1,573 \times 10^{-4}$
0,3	0,00581836	0,00758711	0,00603489	$1,769 \times 10^{-3}$	$2,165 \times 10^{-4}$
0,4	0,00683989	0,00891918	0,00709444	$2,079 \times 10^{-3}$	$2,546 \times 10^{-4}$
0,5	0,00719188	0,00937818	0,00745954	$2,186 \times 10^{-3}$	$2,677 \times 10^{-4}$
0,6	0,00683989	0,00891918	0,00709444	$2,079 \times 10^{-3}$	$2,546 \times 10^{-4}$
0,7	0,00581836	0,00758711	0,00603489	$1,769 \times 10^{-3}$	$2,165 \times 10^{-4}$
0,8	0,00422728	0,00551236	0,00438461	$1,285 \times 10^{-3}$	$1,573 \times 10^{-4}$
0,9	0,00222241	0,00289802	0,00230512	$6,756 \times 10^{-4}$	$8,271 \times 10^{-5}$
1,0	0	0	0		

Terlihat disini bahwa orde kesalahan metode Crank-Nicolson (kolom ke-6) sedikit lebih kecil dibandingkan metode *Backward-Difference* (kolom ke-5). Ini menunjukkan tingkat akurasi Crank-Nicolson lebih tinggi dibandingkan *Backward-Difference*.

8.7 PDP Hiperbolik

Pada bagian ini, kita akan membahas solusi numerik untuk **persamaan gelombang** yang merupakan salah satu contoh PDP hiperbolik. Persamaan gelombang dinyatakan dalam persamaan diferensial sebagai berikut

$$\frac{\partial^2 u}{\partial t^2}(x,t) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t) = 0, \quad 0 < x < \ell, \quad t > 0 \tag{8.61}$$

dengan suatu kondisi

$$u(0,t) = u(\ell,t) = 0, \quad \text{untuk} \quad t > 0,$$

$$u(x,0) = f(x), \quad \text{dan} \quad \frac{\partial u}{\partial t}(x,0) = g(x), \quad \text{untuk} \quad 0 \le x \le \ell$$

dimana α adalah konstanta. Kita tentukan ukuran *time-step* sebesar k, jarak tiap *mesh point* adalah h.

$$x_i = ih \quad \text{dan} \quad t_j = jk$$

dengan $i = 0, 1, ..., m$ dan $j = 0, 1, ...$ Pada bagian interior, posisi *mesh points* ditentukan oleh koordinat (x_i, t_j), karenanya persamaan gelombang ditulis menjadi

$$\frac{\partial^2 u}{\partial t^2}(x_i,t_j) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x_i,t_j) = 0 \tag{8.62}$$

Formula *centered-difference* digunakan sebagai pendekatan numerik persamaan gelombang pada tiap-tiap suku. Untuk turunan kedua terhadap t

$$\frac{\partial^2 u}{\partial t^2}(x_i,t_j) = \frac{u(x_i,t_{j+1}) - 2u(x_i,t_j) + u(x_i,t_{j-1})}{k^2}$$

dan turunan kedua terhadap x

$$\frac{\partial^2 u}{\partial x^2}(x_i,t_j) = \frac{u(x_{i+1},t_j) - 2u(x_i,t_j) + u(x_{i-1},t_j)}{h^2}$$

Dengan mensubtitusikan kedua persamaan di atas kedalam persamaan (8.62)

$$\frac{u(x_i,t_{j+1}) - 2u(x_i,t_j) + u(x_i,t_{j-1})}{k^2} - \alpha^2 \frac{u(x_{i+1},t_j) - 2u(x_i,t_j) + u(x_{i-1},t_j)}{h^2} = 0$$

maka dapat diturunkan formula finite-difference untuk PDP hiperbolik sebagai berikut

$$\frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{k^2} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0 \tag{8.63}$$

Jika $\lambda = \alpha k/h$, maka persamaan ini dapat ditulis kembali

$$w_{i,j+1} - 2w_{i,j} + w_{i,j-1} - \lambda^2 w_{i+1,j} + 2\lambda^2 w_{i,j} - \lambda^2 w_{i-1,j} = 0$$

sehingga $w_{i,j+1}$ selaku solusi numerik dapat dihitung dengan merubah sedikit suku-suku pada formula di atas

$$w_{i,j+1} = 2\left(1 - \lambda^2\right) w_{i,j} + \lambda^2 (w_{i+1,j} + w_{i-1,j}) - w_{i,j-1} \tag{8.64}$$

dengan $i = 1, 2, ..., m-1$ dan $j = 1, 2, ...$ Kondisi syarat batas ditentukan sebagai berikut

$$w_{0,j} = w_{m,j} = 0, \quad \text{untuk} \quad j = 1, 2, 3, ... \tag{8.65}$$

sementara kondisi awal dinyatakan

$$w_{i,0} = f(x_i), \quad \text{untuk} \quad i = 1, 2, ..., m-1 \tag{8.66}$$

Berbeda dengan PDP eliptik dan PDP parabolik, pada PDP hiperbolik, untuk menghitung *mesh point* $(j+1)$, diperlukan informasi mesh point (j) dan $(j-1)$. Hal ini sedikit menimbulkan masalah pada langkah/iterasi pertama karena nilai untuk $j = 0$ sudah ditentukan oleh persamaan (8.66) sementara nilai untuk $j = 1$ untuk menghitung $w_{i,2}$, harus diperoleh lewat kondisi kecepatan awal

$$\frac{\partial u}{\partial t}(x, 0) = g(x), \quad 0 \le x \le \ell \tag{8.67}$$

Salah satu cara pemecahan dengan pendekatan forward-difference adalah

$$\frac{\partial u}{\partial t}(x_i, 0) = \frac{u(x_i, t_1) - u(x_i, 0)}{k} \tag{8.68}$$

$$\begin{aligned} u(x_i, t_1) &= u(x_i, 0) + k\frac{\partial u}{\partial t}(x_i, 0) \\ &= u(x_i, 0) + kg(x_i) \end{aligned}$$

konsekuensinya

$$w_{i,1} = w_{i,0} + kg(x_i), \quad \text{untuk} \quad i = 1, 2, ..., m-1 \tag{8.69}$$

8.7.1 Contoh

Tentukan solusi dari persamaan gelombang berikut ini

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0, \quad 0 < x < 1, \quad 0 < t$$

dengan syarat batas

$$u(0, t) = u(\ell, t) = 0, \quad \text{untuk} \quad 0 < t,$$

dan kondisi mula-mula

$$u(x, 0) = \sin \pi x, \quad 0 \le x \le 1$$

$$\frac{\partial u}{\partial t} = 0, \quad 0 \le x \le 1$$

menggunakan metode *finite-difference*, dengan $m = 4$, $N = 4$, dan $T = 1,0$. Bandingkan hasil yang diperoleh dengan solusi analitik $u(x, t) = \cos \pi t \sin \pi x$.

Jika persamaan gelombang pada contoh soal ini dibandingkan dengan persamaan (8.61), maka diketahui nilai $\alpha = 1$ dan $\ell = 1$. Dari sini, nilai h dapat dihitung, yaitu $h = \ell/m = 1/4 = 0,25$. Sementara nilai k diperoleh dari $k = T/N = 1,0/4 = 0,25$. Dengan diketahuinya nilai α, h, dan k, maka λ dapat dihitung, yaitu $\lambda = \alpha k/h = 1$. Selanjutnya, nilai λ ini dimasukkan ke

persamaan (8.64)

$$\begin{aligned} w_{i,j+1} &= 2\left(1-\lambda^2\right)w_{i,j} + \lambda^2\left(w_{i+1,j}+w_{i-1,j}\right) - w_{i,j-1} \\ w_{i,j+1} &= 2\left(1-1^2\right)w_{i,j} + 1^2\left(w_{i+1,j}+w_{i-1,j}\right) - w_{i,j-1} \\ w_{i,j+1} &= 0w_{i,j} + \left(w_{i+1,j}+w_{i-1,j}\right) - w_{i,j-1} \end{aligned}$$

dimana i bergerak dari 0 sampai m, atau $i = 0, 1, 2, 3, 4$. Sementara j, bergerak dari 0 sampai $T/k = 4$, atau $j = 0, 1, 2, 3, 4$.

Catatan kuliah baru sampai sini!!

8.8 Latihan

1. Carilah solusi persamaan differensial elliptik berikut ini dengan pendekatan numerik menggunakan metode *Finite Difference*

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = (x^2+y^2)e^{xy}, \qquad 0 < x < 2, \qquad 0 < y < 1;$$

gunakan $h = 0,2$ dan $k = 0,1$

$$\begin{aligned} u(0,y) &= 1, \qquad & u(2,y) &= e^{2y}, \qquad & 0 \le y \le 1 \\ u(x,0) &= 1, \qquad & u(x,1) &= e^{x}, \qquad & 0 \le x \le 2 \end{aligned}$$

Bandingkan hasilnya dengan solusi analitik $u(x,t) = e^{xy}$.

2. Carilah solusi persamaan differensial parabolik berikut ini dengan pendekatan numerik menggunakan metode *Finite Difference* Backward-Difference

$$\frac{\partial u}{\partial t} - \frac{1}{16}\frac{\partial^2 u}{\partial x^2} = 0, \qquad 0 < x < 1, \qquad 0 < t;$$

$$\begin{aligned} u(0,t) = u(1,t) &= 0, \qquad 0 < t; \\ u(x,0) &= 2\sin 2\pi x, \qquad 0 \le x \le 1; \end{aligned}$$

gunakan $m = 3$, $T = 0,1$, dan $N = 2$. Bandingkan hasilnya dengan solusi analitik

$$u(x,t) = 2e^{-(\pi^2/4)t}\sin 2\pi x$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + k\frac{\partial u}{\partial t}\left(x_i, 0\right) + \frac{k^2}{2}\frac{\partial^2 u}{\partial t^2}\left(x_i, 0\right) + \frac{k^3}{6}\frac{\partial^3 u}{\partial t^3}\left(x_i, \hat{\mu}_i\right) \tag{8.70}$$

$$\frac{\partial^2 u}{\partial t^2}\left(x_i, 0\right) = \alpha^2\frac{\partial^2 u}{\partial x^2}\left(x_i, 0\right) = \alpha^2\frac{d^f}{dx^2}\left(x_i\right) = \alpha^2 f\text{"}\left(x_i\right) \tag{8.71}$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + kg\left(x_i\right) + \frac{\alpha^2 k^2}{2} f"\left(x_i\right) + \frac{k^3}{6}\frac{\partial^3 u}{\partial t^3}\left(x_i, \hat{\mu}_i\right) \tag{8.72}$$

$$w_{i1} = w_{i0} + kg\left(x_i\right) + \frac{\alpha^2 k^2}{2} f"\left(x_i\right) \tag{8.73}$$

$$f"\left(x_i\right) = \frac{f\left(x_{i+1}\right) - 2f\left(x_i\right) + f\left(x_{i-1}\right)}{h^2} - \frac{h^2}{12} f^{(4)}\left(\tilde{\xi}\right) \tag{8.74}$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + kg\left(x_i\right) + \frac{k^2\alpha^2}{2h^2}\left[f\left(x_{i+1}\right) - 2f\left(x_i\right) + f\left(x_{i-1}\right)h^2\right] + O\left(k^3 + h^2k^2\right) \tag{8.75}$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + kg\left(x_i\right) + \frac{\lambda^2}{2}\left[f\left(x_{i+1}\right) - 2f\left(x_i\right) + f\left(x_{i-1}\right)h^2\right] + O\left(k^3 + h^2k^2\right) \tag{8.76}$$

$$= \left(1 - \lambda^2\right) f\left(x_i\right) + \frac{\lambda^2}{2} f\left(x_{i+1}\right) + \frac{\lambda^2}{2} f\left(x_{i-1}\right) + kg\left(x_i\right) + O\left(k^3 + h^2k^2\right) \tag{8.77}$$

$$w_{i,1} = \left(1 - \lambda^2\right) f\left(x_i\right) + \frac{\lambda^2}{2} f\left(x_{i+1}\right) + \frac{\lambda^2}{2} f\left(x_{i-1}\right) + kg\left(x_i\right) \tag{8.78}$$

Bab 9

Integral Numerik

Objektif :

- ▷ Mengenalkan metode Trapezoida
- ▷ Mengenalkan metode Simpson
- ▷ Mengenalkan metode Composite-Simpson
- ▷ Mengenalkan metode Adaptive Quardrature
- ▷ Mengenalkan metode Gaussian Quadrature

9.1 Metode Trapezoida

Integral terhadap suatu fungsi, *f(x)*, yang dievaluasi dari *a* hingga *b* dapat dinyatakan oleh rumus berikut ini

$$\int_a^b f(x)dx \tag{9.1}$$

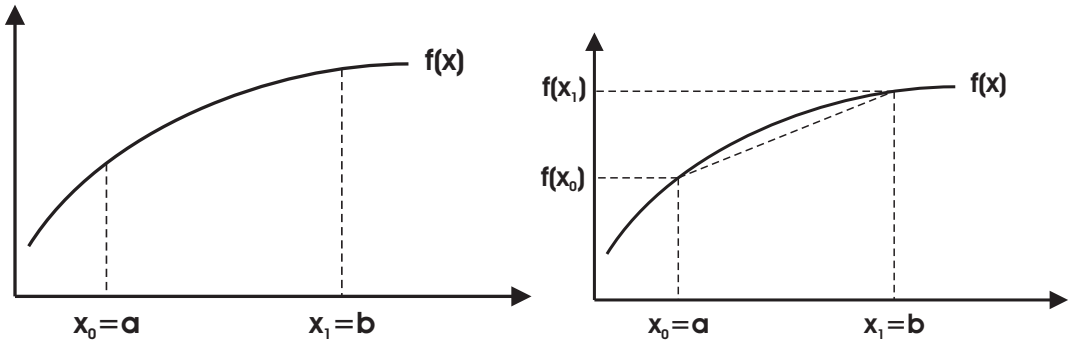
Pendekatan numerik yang paling dasar dalam memecahkan masalah integral adalah metode Trapezoida, yang dirumuskan sebagai berikut

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{h^3}{12}f''(\xi) \tag{9.2}$$

dimana $x_0 = a$, $x_1 = b$ dan $h = b - a$. Akan tetapi, suku yang terakhir pada ruas kanan dimana terdapat faktor turunan ke-2, f'', seringkali diabaikan dengan tujuan agar persamaan (9.2) menjadi lebih sederhana.

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(x_0) + f(x_1)\right] \tag{9.3}$$

Akibatnya pendekatan Trapezoida hanya bekerja efektif pada fungsi-fungsi yang turunan keduanya bernilai nol ($f'' = 0$). Gambar (9.1) memperlihatkan prinsip metode trapezoida dalam bentuk grafik. Sementara, script berikut ini dibuat berdasarkan persamaan (9.3).



Gambar 9.1: Metode Trapezoida. Gambar sebelah kiri menunjukkan kurva fungsi $f(x)$ dengan batas bawah integral adalah a dan batas atas b. Gambar sebelah kanan menunjukan cara metode Trapesoida menghitung integral dengan cara menghitung luas area integrasi, dimana luas area integrasi sama dengan luas trapesium di bawah kurva $f(x)$ dalam batas-batas a dan b. Jika anda perhatikan dengan teliti, ada area kecil dibawah garis kurva dan diatas garis miring yang berada diluar bidang trapesium. Metode Trapesoida tidak menghitung luas area kecil tersebut. Disinilah letak kelemahan metode trapezoida.

```
clear all
clc

a = ...  %batas bawah integral;
b = ...  %batas atas integral;

x0 = a;
x1 = b;
h = b-a;

% -- metode trapezoida --
Int_trapezoida = h/2*(f(x0)+f(x1))
```

Dengan fungsi eksternal fungsi *f(x)* adalah

```
function y = f(x)
y = ... % rumus fungsi yang di-integralkan;
```

9.2 Metode Simpson

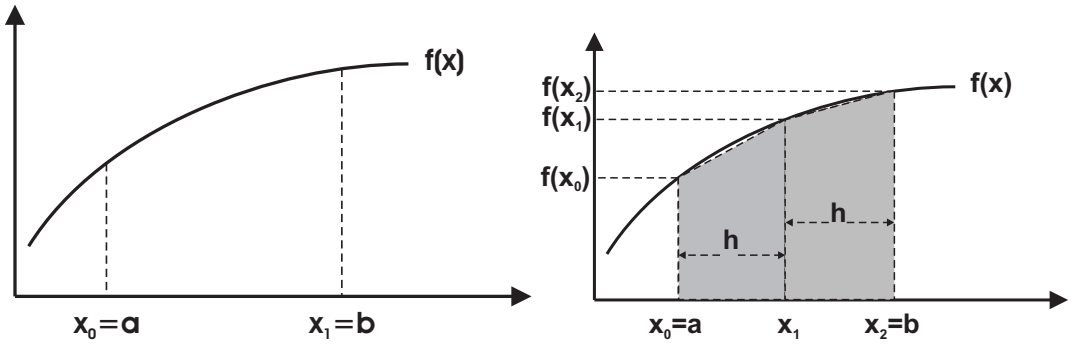
Metode pendekatan yang lebih baik dibanding metode Trapezoida dalam integral numerik adalah metode Simpson yang diformulasikan sebagai berikut

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90}f^4(\xi) \tag{9.4}$$

dengan $x_0 = a$, $x_2 = b$, dan $x_1 = a + h$ dimana $h = (b - a)/2$. Jika suku terakhir diabaikan, maka

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] \tag{9.5}$$

Gambar (9.2) memperlihatkan prinsip metode trapezoida dalam bentuk grafik. Sementara, script berikut ini dibuat berdasarkan persamaan (9.5).



Gambar 9.2: Metode Simpson. Gambar sebelah kiri menunjukkan kurva fungsi $f(x)$ dengan batas bawah integral adalah a dan batas atas b. Gambar sebelah kanan menunjukan cara metode Simpson menghitung luas area integrasi, dimana area integrasi di bawah kurva $f(x)$ dibagi 2 dalam batas interval $a - x_1$ dan $x_1 - b$ dengan lebar masing-masing adalah h

```
clc
clear all

a = ... %batas bawah integrasi ;
b = ... %batas atas integrasi ;

x0 = a;
x2 = b;
h = (b-a)/2;
x1 = a + h;

% -- metode simpson --
Int_simpson = h/3*(f(x0)+4*f(x1)+f(x2))
```

Contoh

Metode Trapezoida untuk fungsi f pada interval [0,2] adalah

$$\int_0^2 f(x)dx \approx f(0) + f(2)$$

dimana $x_0 = 0$, $x_1 = 2$ dan $h = 2 - 0 = 2$. Sedangkan metode Simpson untuk fungsi f pada interval [0,2] adalah

$$\int_0^2 f(x)dx \approx \frac{1}{3}\left[f(0) + 4f(1) + f(2)\right]$$

dengan $x_0 = 0$, $x_2 = 2$, dan $x_1 = a + h = 1$ dimana $h = (b - a)/2 = 1$.

Tabel berikut ini memperlihatkan evaluasi integral numerik terhadap beberapa fungsi dalam interval [0,2] beserta solusi exact-nya. Jelas terlihat, metode Simpson lebih baik dibanding Trapezoida. Karena hasil intergral numerik metode Simpson lebih mendekati nilai exact

$f(x)$	x^2	x^4	$1/(x+1)$	$\sqrt{1+x^2}$	$\sin x$	e^x
Nilai exact	2,667	6,400	1,099	2,958	1,416	6,389
Trapezoida	4,000	16,000	1,333	3,326	0,909	8,389
Simpson	2,667	6,667	1,111	2,964	1,425	6,421

9.3 Peran faktor pembagi, n

Kalau diamati lebih teliti, akan kita dapatkan bahwa interval [0,2] telah dibagi 2 pada metode Simpson, sementara pada metode Trapesoida tidak dibagi sama sekali. Sebenarnya dengan membagi interval lebih kecil lagi, maka *error*-nya akan semakin kecil. Misalnya, banyaknya pembagian interval dinyatakan dengan n
ketika $n = 1$: Trapesioda

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{h^3}{12}f''(\xi) \tag{9.6}$$

ketika $n = 2$: Simpson

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90}f^4(\xi) \tag{9.7}$$

ketika $n = 3$: Simpson tiga-per-delapan

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] - \frac{3h^5}{80}f^4(\xi) \tag{9.8}$$

ketika $n = 4$:

$$\int_{x_0}^{x_4} f(x)dx = \frac{2h}{45}\left[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)\right] - \frac{8h^7}{945}f^6(\xi) \tag{9.9}$$

9.3.1 Source code metode integrasi

Source code untuk persamaan (9.8) disajikan sebagai berikut

```
clc
clear all

% -- batas integrasi --
a = 0;
b = 2;

x0 = a;
x3 = b;
h = (b-a)/3;
x1 = a + h;
x2 = a + 2*h;
% ---------------------

% -- metode simpson 3/8 --
Int_38 = 3*h/8*(f(x0)+3*f(x1)+3*f(x2)+f(x3))
```

Sementara, source code untuk persamaan (9.9) disajikan sebagai berikut

```
clc
clear all

```

```
% -- batas integrasi --
a = 0;
b = 2;

x0 = a;
x4 = b;
h = (b-a)/4;
x1 = a + h;
x2 = a + 2*h;
x3 = a + 3*h;
% --------------------

% -- metode simpson n=4 --
Int_n4 = 2*h/45*(7*f(x0)+32*f(x1)+12*f(x2)+32*f(x3)+7*f(x4))
```

Perbandingan tingkat akurasi hasil perhitungan seluruh metode integral numerik yang sudah dibahas adalah sebagai berikut

$f(x)$	x^2	x^4	$1/(x+1)$	$\sqrt{1+x^2}$	$\sin x$	e^x
Nilai exact	2,667	6,400	1,099	2,958	1,416	6,389
Trapezoida	4,000	16,000	1,333	3,326	0,909	8,389
Simpson n=2	2,667	6,667	1,111	2,964	1,425	6,421
Simpson n=3	2,667	6,519	1,105	2,960	1,420	6,403
Simpson n=4	2,667	6,400	1,099	2,958	1,416	6,389

Keempat bentuk persamaan integral numerik di atas dikenal dengan **closed Newton-Cotes formulas**. Keterbatasan metode Newton-Cotes terlihat dari jumlah pembagian interval. Di atas tadi pembagian interval baru sampai pada $n = 4$. Bagaimana bila interval evaluasinya dipersempit supaya solusi numeriknya lebih mendekati solusi exact? Atau dengan kata lain $n > 4$.

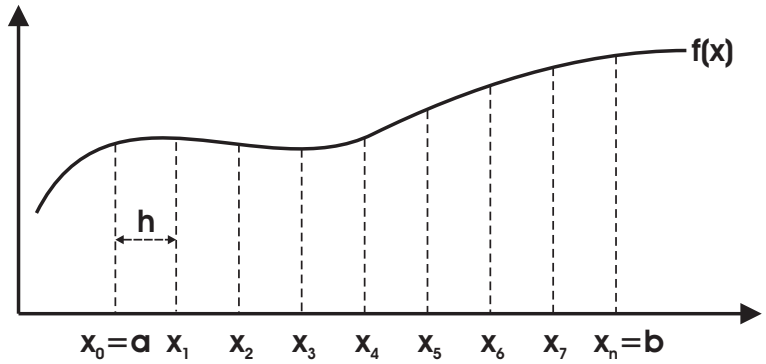
9.4 Metode Composite-Simpson

Persamaan (9.9) terlihat lebih rumit dibandingkan persamaan-persamaan sebelumnya. Bisakah anda bayangkan bentuk formulasi untuk $n = 5$ atau $n = 6$ dan seterusnya? Pasti akan lebih kompleks dibandingkan persamaan (9.9).

Metode Composite Simpson menawarkan cara mudah menghitung intergal numerik ketika nilai $n > 4$. Perhatikan contoh berikut, tentukan solusi numerik dari $\int_0^4 e^x dx$. Metode Simpson dengan $h = 2$ (atau interval evaluasi integral dibagi 2 , $n = 2$) memberikan hasil

$$\int_0^4 e^x dx \approx \frac{2}{3}\left(e^0 + 4e^2 + e^4\right) = 56,76958$$

Padahal solusi exact dari integral tersebut adalah $e^4 - e^0 = 53,59815$, artinya terdapat *error* sebesar 3,17143 yang dinilai masih terlampau besar untuk ditolerir. Bandingkan dengan



Gambar 9.3: Metode Composite Simpson. Kurva fungsi $f(x)$ dengan batas bawah integral adalah a dan batas atas b. Luas area integrasi dipecah menjadi 8 area kecil dengan lebar masing-masing adalah h.

metode yang sama namun dengan $h = 1$ (atau interval evaluasi integral dibagi 4 , $n = 4$)

$$\begin{aligned}\int_0^4 e^x dx &= \int_0^2 e^x dx + \int_2^4 e^x dx \\ &\approx \frac{1}{3}\left(e^0 + 4e + e^2\right) + \frac{1}{3}\left(e^2 + 4e^3 + e^4\right) \\ &= \frac{1}{3}\left(e^0 + 4e + 2e^2 + 4e^3 + e^4\right) \\ &= 53,86385\end{aligned}$$

Hasil ini memperlihatkan *error* yang makin kecil, yaitu menjadi 0,26570. Jadi dengan memperkecil h, *error* menjadi semakin kecil dan itu artinya solusi integral numerik semakin mendekati solusi exact. Sekarang kita coba kecilkan lagi nilai h menjadi $h = \frac{1}{2}$ (atau interval evaluasi integral dibagi 8 , $n = 8$),

$$\begin{aligned}\int_0^4 e^x dx &= \int_0^1 e^x dx + \int_1^2 e^x dx + \int_2^3 e^x dx + \int_3^4 e^x dx \\ &\approx \frac{1}{6}\left(e^0 + 4e^{1/2} + e\right) + \frac{1}{6}\left(e + 4e^{3/2} + e^2\right) + \\ &\quad \frac{1}{6}\left(e^2 + 4e^{5/2} + e^3\right) + \frac{1}{6}\left(e^3 + 4e^{7/2} + e^4\right) \\ &= \frac{1}{6}\left(e^0 + 4e^{1/2} + 2e + 4e^{3/2} + 2e^2 + 4e^{5/2} + 2e^3 + 4e^{7/2} + e^4\right) \\ &= 53,61622\end{aligned}$$

dan seperti yang sudah kita duga, *error*-nya semakin kecil menjadi 0,01807.

Prosedur ini dapat digeneralisir menjadi suatu formula sebagai berikut

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)dx \\ &= \sum_{j=1}^{n/2} \left\{ \frac{h}{3}\left[f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})\right] - \frac{h^5}{90} f^{(4)}(\xi_j) \right\} \qquad (9.10)\end{aligned}$$

dimana $h = (b-a)/n$ dan $x_j = a+jh$, untuk $j = 1, ..., n/2$, dengan $x_0 = a$ dan $x_n = b$. Formula ini dapat direduksi menjadi

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right] - \frac{h^5}{90}\sum_{j=1}^{n/2} f^{(4)}(\xi_j) \quad (9.11)$$

Formula ini dikenal sebagai metode **Composite Simpson**.

9.5 Adaptive Quardrature

Metode composite mensyaratkan luas area integrasi dibagi menjadi sejumlah *region* dengan jarak interval yang seragam yaitu sebesar nilai h. Akibatnya, bila metode composite diterapkan pada fungsi yang memiliki variasi yang tinggi dan rendah sekaligus, maka interval h yang kecil menjadi kurang efektif, sementara interval h yang besar mengundang $error$ yang besar pula. Metode *Adaptive Quadrature* muncul untuk mendapatkan langkah yang paling efektif dimana nilai interval h tidak dibuat seragam, melainkan mampu beradaptasi sesuai dengan tingkat variasi kurva fungsinya.

Misalnya kita bermaksud mencari solusi numerik dari integral $\int_a^b f(x)dx$ dengan toleransi $\epsilon > 0$. Sebagai langkah awal adalah menerapkan metode Simpson dimana step size $h = (b-a)/2$

$$\int_a^b f(x)dx = S(a,b) - \frac{h^5}{90}f^{(4)}(\mu) \quad (9.12)$$

dengan

$$S(a,b) = \frac{h}{3}\left[f(a) + 4f(a+h) + f(b)\right]$$

Langkah berikutnya adalah men

$$\begin{aligned}\int_a^b f(x)dx &= \frac{h}{6}\left[f(a) + 4f\left(a+\frac{h}{2}\right) + 2f\left(a+h\right) + 4f\left(a+\frac{3h}{2}\right) + f(b)\right] \\ &- \left(\frac{h}{2}\right)^4 \frac{(b-a)}{180} f^{(4)}(\tilde{\mu})\end{aligned} \quad (9.13)$$

9.6 Gaussian Quadrature

Suatu integral dapat ditransformasi kedalam bentuk Gaussian quadrature melalui formulasi berikut

$$\int_a^b f(x)dx = \int_{-1}^1 f\left(\frac{(b-a)t + (b+a)}{2}\right)\frac{(b-a)}{2}dt \quad (9.14)$$

dimana perubahan variabel memenuhi

$$t = \frac{2x-a-b}{b-a} \Leftrightarrow x = \frac{1}{2}\left[(b-a)t + a + b\right] \quad (9.15)$$

Berikut adalah table polinomial Legendre untuk penyelesaian Gaussian quadrature

Tabel 9.1: Polinomial Legendre untuk n=2,3,4 dan 5

n	Akar $r_{n,i}$	Koefisien $c_{n,i}$
2	0,5773502692	1,0000000000
	-0,5773502692	1,0000000000
3	0,7745966692	0,5555555556
	0,0000000000	0,8888888889
	-0,7745966692	0,5555555556
4	0,8611363116	0,3478548451
	0,3399810436	0,6521451549
	-0,3399810436	0,6521451549
	-0,8611363116	0,3478548451
5	0,9061798459	0,2369268850
	0,5384693101	0,4786286705
	0,0000000000	0,5688888889
	-0,5384693101	0,4786286705
	-0,9061798459	0,2369268850

9.6.1 Contoh

Selesaikan integrasi berikut ini

$$\int_1^{1,5} e^{-x^2} dx \tag{9.16}$$

(Solusi exact integral diatas adalah: 0.1093643)

jawab:
Pertama, integral tersebut ditransformasikan kedalam Gaussian quadrature melalui persamaan (9.14)

$$\int_1^{1,5} e^{-x^2} dx = \frac{1}{4} \int_{-1}^{1} e^{\frac{-(t+5)^2}{16}} dt \tag{9.17}$$

Kedua, Gaussian quadrature dihitung menggunakan konstanta-konstanta yang tercantum pada tabel polinomial Legendre. Untuk $n = 2$

$$\int_1^{1,5} e^{-x^2} dx \approx \frac{1}{4} \left[e^{(-(0,5773502692+5)^2/16)} + e^{(-(-0,5773502692+5)^2/16)} \right] = 0,1094003$$

Untuk $n = 3$

$$\begin{aligned} \int_1^{1,5} e^{-x^2} dx &\approx \frac{1}{4}[(0,5555555556) e^{(-(0,7745966692+5)^2/16)} + (0,8888888889) e^{(-(5)^2/16)} \\ &+ (0,5555555556) e^{(-(-0,7745966692+5)^2/16)}] = 0,1093642 \end{aligned}$$

9.6.2 Latihan

Selesaikan integrasi berikut ini

$$\int_0^{0,35} \frac{2}{x^2 - 4} dx$$

Selesaikan integrasi berikut ini

$$\int_3^{3,5} \frac{x}{\sqrt{x^2-4}} dx$$

Latihan

1. Hitunglah integral-integral berikut ini dengan metode Composite Simpson!

$$a. \quad \int_1^2 x \ln x dx, \quad n = 4$$

$$b. \quad \int_0^2 \frac{2}{x^2+4} dx, \quad n = 6$$

$$c. \quad \int_1^3 \frac{x}{x^2+4} dx, \quad n = 8$$

$$d. \quad \int_{-2}^2 x^3 e^x dx, \quad n = 4$$

$$e. \quad \int_0^{3\pi/8} \tan x dx, \quad n = 8$$

$$f. \quad \int_3^5 \frac{1}{\sqrt{x^2-4}} dx, \quad n = 8$$

2. Tentukan nilai n dan h untuk mengevaluasi

$$\int_0^2 e^{2x} \sin 3x dx$$

dengan metode Composite Simpson, bila *error* yang ditolerir harus lebih kecil dari 10^{-4}.

3. Dalam durasi 84 detik, kecepatan sebuah mobil balap formula 1 yang sedang melaju di arena *grandprix* dicatat dalam selang interval 6 detik:

$time(dt)$	0	6	12	18	24	30	36	42	48	54	60	66	72	78	84
$speed(ft/dt)$	124	134	148	156	147	133	121	109	99	85	78	89	104	116	123

Gunakan metode integral numerik untuk menghitung panjang lintasan yang telah dilalui mobil tersebut selama pencatatan waktu di atas!

Bab 10

Mencari Akar

Objektif :

▷ Mencari akar

10.1 Metode Newton

Metode Newton sangat populer dan *powerfull* untuk mencari akar suatu fungsi yang kontinyu. Ada banyak jalan untuk memperkenalkan metode ini. Salah satunya bisa didahului mulai dari deret Taylor atau polinomial Taylor. Suatu fungsi yang kontinyu dapat dinyatakan dalam deret Taylor sebagai berikut

$$f(x) = f(\bar{x}) + (x - \bar{x})f'(\bar{x}) + \frac{(x - \bar{x})^2}{2}f''(\xi(x))$$

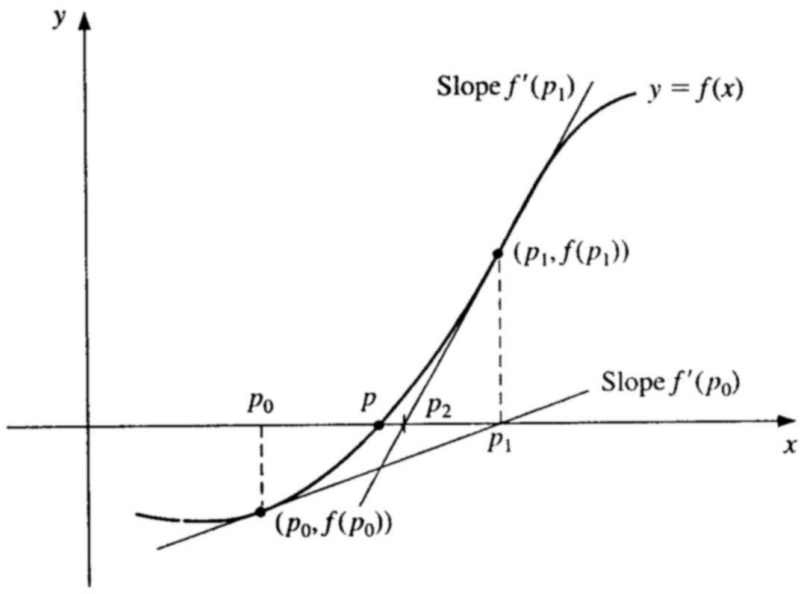
$$0 = f(\bar{x}) + (p - \bar{x})f'(\bar{x}) + \frac{(p - \bar{x})^2}{2}f''(\xi(p))$$

$$0 = f(\bar{x}) + (p - \bar{x})f'(\bar{x})$$

$$p - \bar{x} = -\frac{f(x)}{f'(\bar{x})}$$

$$p \approx \bar{x} - \frac{f(x)}{f'(\bar{x})}$$

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \quad , \qquad n \geq 1$$



Gambar 10.1: Metode Newton

Bab 11

Metode Monte Carlo

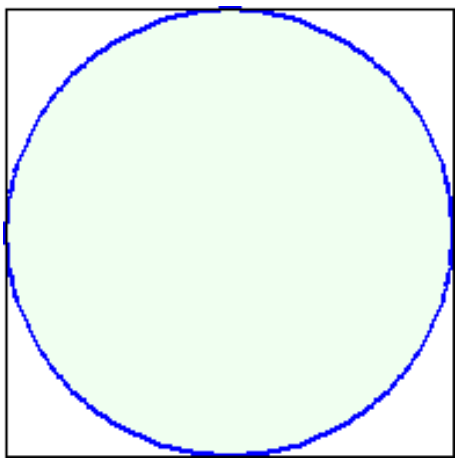
Objektif :

- Mengenalkan metode Monte Carlo

11.1 Penyederhanaan

Kita awali pembahasan metode Monte Carlo dengan mengetengahkan contoh yang sangat terkenal yaitu menghitung luas suatu lingkaran. Fugure 1 memperlihatkan lingkaran dengan radius $r = 1$ berada di dalam kotak bujursangkar. Luas lingkaran adalah $\pi r^2 = \pi(1)^2 = \pi$ sementara luas bujursangkar adalah $(2)^2 = 4$. Rasio antara luas lingkaran dan luas bola adalah

$$\rho = \frac{luas \quad lingkaran}{luas \quad bujursangkar} = \frac{\pi}{4} = 0,7853981633974483 \tag{11.1}$$



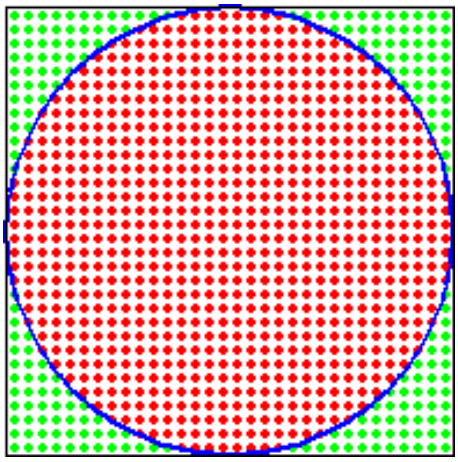
Gambar 11.1: Lingkaran dan bujursangkar

Jadi, dengan mengetahui nilai ρ, maka kita bisa menghitung luas lingkaran dengan cara

$$luas \quad lingkaran = \rho \times luas \quad bujursangkar \tag{11.2}$$

Bayangkan anda punya satu set permainan dart. Anda lemparkan sejumlah dart ke arah lingkaran tadi. Misalnya, total dart yang menancap di papan dart ada 1024 buah. Sebanyak 812 dart berada di dalam lingkaran, dan yang lainnya di luar lingkaran. Rasio antara keduanya

$$\rho = \frac{dart \quad di \quad dalam \quad lingkaran}{total \quad dart \quad di \quad dalam \quad bujursangkar} = \frac{812}{1024} = 0,79296875 \tag{11.3}$$



Gambar 11.2: Dart yang menancap pada bidang lingkaran dan bujursangkar

Dengan pendekatan ke persamaan (11.2) maka luas lingkaran adalah

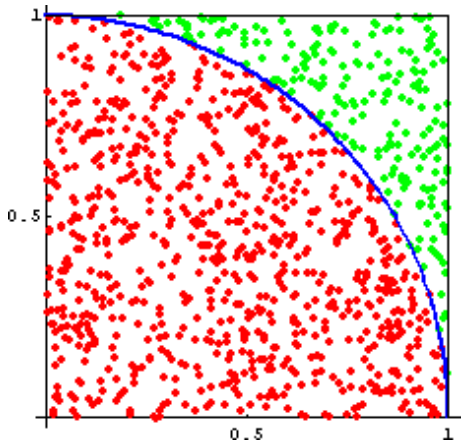
$$\begin{aligned} luas \quad lingkaran &= \rho \times luas \quad bujursangkar \\ &= 0,79296875 \times 4 \\ &= 3,171875 \end{aligned}$$

Apakah angka ini *make sense*? Mungkin anda masih ragu. Sekarang mari kita coba hitung nilai π dengan mengacu pada rumus di atas. Kita sepakati saja bahwa dart yang berada di dalam lingkaran mesti memenuhi $x_i^2 + y_i^2 \leq 1$. Dalam perhitungan, semua dart diganti dengan bilangan acak (*random number*). Dari 1000 dart, yang masuk lingkaran ada 787 buah, sehingga, mengacu persamaan (11.3)

$$\rho = \frac{787}{1000} = 0,787$$

maka berdasarkan persamaan (11.1)

$$\pi = \rho \times 4 = 0,787 \times 4 = 3,148$$



Gambar 11.3: Dart yang menancap pada bidang 1/4 lingkaran dan bujursangkar

Lumayan akurat bukan? Semakin banyak jumlah dart, semakin akurat nilai π yang anda peroleh.

Sekarang mari kita kembangkan metode Monte Carlo ini untuk menghitung luas suatu area yang terletak di bawah garis kurva suatu fungsi $f(x)$. Atau sebut saja menghitung integral suatu fungsi $f(x)$ yang dievaluasi antara batas a dan b. Luas kotak **R** yang melingkupi luas bidang integral **A** adalah

$$\mathbf{R} = \{(x, y) : a \leq x \leq b \quad dan \quad 0 \leq y \leq d\} \tag{11.4}$$

dimana

$$d = maksimum \quad f(x) \quad , \quad a \leq x \leq b \tag{11.5}$$

Bab 12

Inversi

Objektif :

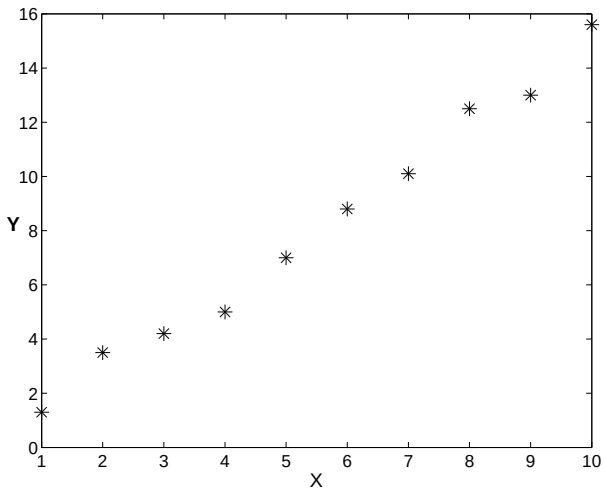
- ▷ Mengenalkan inversi linear
- ▷ Mengenalkan inversi non-linear

12.1 Inversi Linear

Diketahui data eksperimen tersaji dalam tabel berikut ini

x_i	y_i	x_i	y_i
1	1,3	6	8,8
2	3,5	7	10,1
3	4,2	8	12,5
4	5,0	9	13,0
5	7,0	10	15,6

Lalu data tersebut di-plot dalam sumbu x dan y. Sekilas, kita bisa melihat bahwa data yang



telah di-plot tersebut dapat didekati dengan sebuah persamaan garis, yaitu $a_1 x_i + a_0$. Artinya,

kita melakukan pendekatan secara linear, dimana fungsi pendekatan-nya adalah

$$P(x_i) = a_1 x_i + a_0 \tag{12.1}$$

Problemnya adalah berapakah nilai konstanta a_1 dan a_0 yang sedemikian rupa, sehingga posisi garis tersebut paling mendekati atau bahkan melalui titik-titik data yang telah di-plot di atas? Dengan kata lain, sebisa mungkin y_i sama dengan $P(x_i)$ atau dapat diformulasikan sebagai

$$\sum_{i=1}^{m} y_i - P(x_i) = 0 \tag{12.2}$$

$$\sum_{i=1}^{m} y_i - (a_1 x_i + a_0) = 0 \tag{12.3}$$

dimana jumlah data, $m = 10$. Suku yang berada disebelah kiri dinamakan fungsi error (*error function*), yaitu

$$E(a_0, a_1) = \sum_{i=1}^{m} y_i - (a_1 x_i + a_0) \tag{12.4}$$

Semua data yang diperoleh melalui eksperimen, fungsi error-nya tidak pernah bernilai nol. Jadi, tidak pernah didapatkan garis yang berhimpit dengan semua titik data ekperimen. Namun demikian, kita masih bisa berharap agar fungsi error menghasilkan suatu nilai, dimana nilai tersebut adalah nilai yang paling minimum atau paling mendekati nol. Harapan tersebut diwujudkan oleh metode **least square** dengan sedikit modifikasi pada fungsi error-nya sehingga menjadi

$$E(a_0, a_1) = \sum_{i=1}^{m} [y_i - (a_1 x_i + a_0)]^2 \tag{12.5}$$

Agar fungsi error bisa mencapai nilai minimum, maka syarat yang harus dipenuhi adalah:

$$\frac{\partial E(a_0, a_1)}{\partial a_i} = 0 \tag{12.6}$$

dimana $i = 0$ dan 1, karena dalam kasus ini memang cuma ada a_0 dan a_1. Maka mesti ada dua buah turunan yaitu:

$$\begin{aligned} \frac{\partial E(a_0, a_1)}{\partial a_0} = \frac{\partial}{\partial a_0} \sum_{i=1}^{m} [y_i - (a_1 x_i + a_0)]^2 &= 0 \\ 2 \sum_{i=1}^{m} (y_i - a_1 x_i - a_0)(-1) &= 0 \\ a_0 . m + a_1 \sum_{i=1}^{m} x_i &= \sum_{i=1}^{m} y_i \end{aligned} \tag{12.7}$$

dan

$$\begin{aligned}\frac{\partial E(a_0,a_1)}{\partial a_1} = \frac{\partial}{\partial a_1}\sum_{i=1}^{m}[y_i-(a_1x_i+a_0)]^2 &= 0\\ 2\sum_{i=1}^{m}(y_i-a_1x_i-a_0)(-x_i) &= 0\\ a_0\sum_{i=1}^{m}x_i+a_1\sum_{i=1}^{m}x_i^2 &= \sum_{i=1}^{m}x_iy_i \end{aligned} \tag{12.8}$$

Akhirnya persamaan (12.7) dan (12.8) dapat dicari solusinya berikut ini:

$$a_0 = \frac{\sum_{i=1}^{m}x_i^2\sum_{i=1}^{m}y_i-\sum_{i=1}^{m}x_iy_i\sum_{i=1}^{m}x_i}{m\left(\sum_{i=1}^{m}x_i^2\right)-\left(\sum_{i=1}^{m}x_i\right)^2} \tag{12.9}$$

dan

$$a_1 = \frac{m\sum_{i=1}^{m}x_iy_i-\sum_{i=1}^{m}x_i\sum_{i=1}^{m}y_i}{m\left(\sum_{i=1}^{m}x_i^2\right)-\left(\sum_{i=1}^{m}x_i\right)^2} \tag{12.10}$$

Coba anda bandingkan kedua hasil di atas dengan rumus least square yang terdapat pada buku **Praktikum Fisika Dasar** keluaran Departemen Fisika-UI. Mudah-mudahan sama persis. OK, berdasarkan data ekperimen yang ditampilkan pada tabel diawal catatan ini, maka didapat:

$$a_0 = \frac{385(81)-55(572,4)}{10(385)-(55)^2} = -0,360 \tag{12.11}$$

dan

$$a_1 = \frac{10(572,4)-55(81)}{10(385)-(55)^2} = 1,538 \tag{12.12}$$

Jadi, fungsi pendekatan-nya, $P(x_i)$, adalah

$$P(x_i) = 1,538x_i - 0,360 \tag{12.13}$$

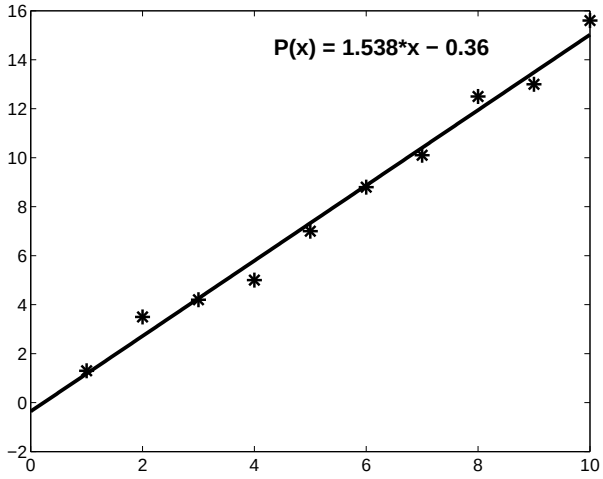
Solusi least square dengan pendekatan persamaan garis seperti ini juga dikenal dengan nama lain yaitu **regresi linear**. Sedangkan nilai a_0 dan a_1 disebut **koefisien regresi**. Gambar di bawah ini menampilkan solusi regresi linear tersebut berikut semua titik datanya

Tentu saja anda sudah bisa menduga bahwa selain regresi linear, mungkin saja terdapat regresi parabola atau quadratik dimana fungsi pendekatannya berupa persamaan parabola, yaitu:

$$P(x_i) = a_2x_i^2 + a_1x_i + a_0 \tag{12.14}$$

dimana koefisien regresinya ada tiga yaitu a_0, a_1 dan a_2. Kalau anda menduga demikian, maka dugaan anda benar! Bahkan sebenarnya tidak terbatas sampai disitu. Secara umum, fungsi pendekatan, $P(x_i)$, bisa dinyatakan dalam aljabar polinomial berikut ini:

$$P(x_i) = a_nx_i^n + a_{n-1}x_i^{n-1} + ... + a_2x_i^2 + a_1x_i + a_0 \tag{12.15}$$



Namun untuk saat ini, saya tidak ingin memperluas pembahasan hingga regresi parabola, dan polinomial. Saya masih ingin melibatkan peranan metode eliminasi gauss dalam menyelesaikan problem least square seperti yang selalu saya singgung pada catatan-catatan kuliah saya yang terdahulu. Nah, kalau metode eliminasi gauss hendak digunakan untuk mencari solusi regresi linear, kita bisa mulai dari persamaan (12.7) dan (12.8), yaitu:

$$\begin{aligned} a_0.m + a_1 \sum_{i=1}^{m} x_i &= \sum_{i=1}^{m} y_i \\ a_0 \sum_{i=1}^{m} x_i + a_1 \sum_{i=1}^{m} x_i^2 &= \sum_{i=1}^{m} x_i y_i \end{aligned}$$

Keduanya bisa dinyatakan dalam operasi matrik:

$$\begin{bmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{bmatrix} \tag{12.16}$$

Kalau anda mengikuti catatan-catatan terdahulu, pasti anda tidak asing lagi dengan dengan semua elemen-elemen matrik di atas. Semua sudah saya ulas pada catatan yang berjudul **Aplikasi Elimininasi Gauss: Model Garis**. Silakan anda lanjutkan perhitungan matrik tersebut hingga diperoleh koefisien regresi a_0 dan a_1. Selamat mencoba!

12.2 Inversi Non-Linear

Persamaan least squares linear adalah sebagai berikut:

$$[\mathbf{G}^t\mathbf{G}]\delta m = \mathbf{G}^t\delta d \tag{12.17}$$

Persamaan least squares non-linear dapat dinyatakan sebagai berikut:

$$[\mathbf{G}^t\mathbf{G} + \lambda\mathbf{I}]\delta m = \mathbf{G}^t\delta d \tag{12.18}$$

dimana **G** adalah matrik **kernel**, namun dia juga biasa dikenal dengan sebutan matrik **Jacobian**, sementara λ adalah faktor pengali Lagrange, dan **I** adalah matrik identitas yang ordenya disesuaikan dengan $\mathbf{G}^t\mathbf{G}$. Adapun definisi δm dan δd akan dijelaskan pada bagian akhir catatan ini.
Langkah-langkah untuk menyelesaikan problem least squares non-linear adalah:

1. Menentukan model, misal $f(x) = x^m$

2. Menghitung jacobian, **G**. Caranya adalah menghitung turunan pertama dari model terhadap model-parameter, m. Sesuai permisalan pada point 1, didapat

$$G = \frac{\partial f(x)}{\partial m} = x^m ln(x) \tag{12.19}$$

3. Membuat perhitungan simulasi, misalnya ditentukan $m = 2$. Nilai m adalah nilai yang hendak dicari. Dalam simulasi, nilai m dianggap sudah diketahui bahkan ditentukan. Lalu hitunglah $f(x) = x^m$ dengan x bergerak dari $x = 1, 2, 3.., 10$. Jadi, nanti akan didapat 10 buah $f(x)$. Mau lebih dari 10 juga boleh, terserah saja. Hasil hitungannya dikasih nama d, jadi $d = f(x)$. Karena dalam simulasi ini x-nya bergerak hanya sampai 10, maka hasilnya mesti ada 10 d, yaitu $d_1, d_2, .., d_{10}$.

4. Buatlah perhitungan untuk m sembarang, misal mula-mula dipilih $m = 5$. Ini adalah nilai awal dari m yang akan diiterasikan sedemikian rupa hingga nantinya m akan menuju 2 sesuai dengan nilai m pada simulasi (point 3). Bagusnya dibedakan penulisannya, atau tulis saja $m^0 = 5$, dimana m^0 maksudnya adalah m mula-mula. Lalu hitung lagi nilai $f(x) = x^{m^0}$. Sekarang dinamakan $d^c = f(x)$. Jangan lupa bahwa saat perhitungan, nilai x bergerak dari 1 sampai 10. Jadi, nanti didapat 10 d^c.

5. Hitunglah δd, dimana $\delta d = d^c - d$. Sebelumnya sudah dinyatakan bahwa d^c ada 10 buah, demikian juga d ada 10 buah, maka δd harus ada 10 buah juga.

6. Selanjutnya hitung $||\delta d||$ yang rumusnya seperti ini

$$||\delta d|| = \frac{1}{N}\Sigma(d^c - d)^2 = \frac{1}{N}\Sigma\delta d^2 \tag{12.20}$$

   dimana $N = 10$ karena δd-nya ada 10. Rumus ini tidak mutlak harus demikian, anda bisa juga menggunakan norm 2, ℓ_2.

7. Tentukan nilai epsilon, ϵ, misal $\epsilon = 0.000001$. Lalu lakukan evaluasi sederhana. Cek, apakah $||\delta d|| < \epsilon$? Pasti awalnya $||\delta d|| > \epsilon$, kenapa? Karena $m \neq m^0$. Kalau begini situasinya, δd yang ada 10 biji itu dimasukan kedalam proses berikutnya.

8. Hitunglah operasi matriks berikut ini untuk mendapatkan δm

$$[\mathbf{G}^t\mathbf{G} + \lambda\mathbf{I}]\delta m = \mathbf{G}^t\delta d \tag{12.21}$$

dengan λ-nya dikasih nilai sembarang antara 0 dan 1, misalnya $\lambda = 0.005$. Perhitungan ini bisa diselesaikan dengan metode eliminasi gauss.

9. Ganti nilai m^0 menjadi m^1 sesuai dengan rumus

$$m^1 = m^0 + \delta m \tag{12.22}$$

Nah, m^1 ini dimasukan ke proses yang dijelaskan pada point 4 kemudian proses diulangi hingga point 9, begitu seterusnya. Dari sinilah dimulai proses iterasi. Iterasi akan berhenti bila $||\delta d|| < \epsilon$. Pada saat itu, nilai m^k akan mendekati $m = 2$ sesuai dengan m simulasi.

Selamat mencoba! Saya juga telah menulis beberapa persamaan non-linear sebagai bahan latihan. Lihat saja di Latihan 1. Tapi tolong diperiksa lagi, apakah jacobiannya sudah benar atau ada kekeliruan. Selanjutnya, kalau ada pertanyaan atau komentar, silakan kirim ke supri92@gmail.com

Daftar Pustaka

[1] Burden, R.L. and Faires, J.D., (2001), *Numerical Analysis, Seventh Edition*, Brooks/Cole, Thomson Learning Academic Resource Center.

[2] Haliday and Resnick, (2001), *Fundamental of Physics*, Brooks/Cole, Thomson Learning Academic Resource Center.

Indeks