Komputasi untuk Sains dan Teknik

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

Edisi III
Revisi terakhir tgl: 30 Agustus 2009

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*)

Trust and friendship are more valuable than your skills

(*Supriyanto, 2008*)

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 penulis terima dari mahasiswa UNPAD, UDAYANA, UNESA dan UNSRI serta UI sendiri, ataupun sekedar pertanyaan seputar pekerjaan rumah 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 akan menyajikan sebagian besar yang masih kurang lengkap itu secara bertahap.

Ibarat pohon yang akan terus tumbuh semakin besar, buku ini pun memiliki tabiat pertumbuhan sebagaimana pohon itu. Mulai ditulis pada tahun 2005 dengan isi yang seadanya, pokoknya asal tercatat. Kemudian di tahun 2006-akhir, ia menjadi catatan perkuliahan Komputasi Fisika untuk pertama kalinya di Departemen Fisika, FMIPA-UI. Pengayaan isi terus berlangsung hingga akhir 2007. Lalu di awal tahun 2008 diisi dengan tambahan materi perkuliahan Analisis Numerik. Itulah yang saya maksud dengan tabiat pertumbuhan dari buku ini. Jika saya ditugaskan untuk mengajar mata kuliah Komputasi Fisika lagi pada awal September 2008, saya bertekad akan menurunkan seluruh isi buku ini kepada mahasiswa yang akan mengambil kuliah tersebut. Jadi materi Komputasi Fisika tahun 2007 dan materi Analisis Numerik 2008, digabung jadi satu kedalam satu semester dengan nama mata kuliah Komputasi Fisika. Kepada rekan-rekan mahasiswa yang akan *ngambil* mata kuliah tersebut, saya sampaikan permohonan maaf jika rencana ini akan membuat anda kurang tidur karena *bakal* semakin lama berada di depan komputer, menyelesaikan tugas dan *report*.

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 mencoba membiasakan diri menulis *script* dalam lingkungan **Python** dan **Octave**. Padahal, dalam edisi ke-2 yang lalu, *script* numerik disalin ke dalam 2 bahasa pemrograman, yaitu **Fortran77** dan **Matlab**. Namun mayoritas ditulis dalam **Matlab**. Saya ingin ganti ke **Python**, lantaran dengan **Python** ataupun **Octave**, saya dan juga mahasiswa saya tidak perlu menginstal **Matlab** bajakan ke dalam komputer kami masing-masing.

Buku yang sedang anda baca ini masih jauh dari sempurna. Keterkaitan antar Bab berikut isi-nya masih perlu perbaikan. Kondisi ini berpotensi membuat anda bingung, atau setidaknya menjadi kurang fokus. Oleh karena itu saya menghimbau kepada pembaca untuk menfokuskan diri melalui penjelasan singkat berikut ini:

- Bab 1 berisi pengenalan matrik, operasi matrik, inisialisasi matrik pada Matlab dan Fortran. Saran saya, setiap pembaca yang masih pemula di dunia pemrograman, harus menguasai Bab I terlebih dahulu. Disamping itu penjelasan lebih terperinci tentang bagaimana menentukan indeks i, j dan k dalam proses *looping* disajikan pada Bab I, untuk memberi pondasi yang kokoh bagi berdirinya bangunan pemahaman akan teknik-teknik numerik selanjutnya.

- Untuk mempelajari metode Finite-Difference, dianjurkan mulai dari Bab 1, Bab 2, Bab 4, Bab 7, dan Bab 8.

- Untuk mempelajari dasar-dasar inversi, dianjurkan mulai dari Bab 1, Bab 2, dan Bab 3.

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 sepatutnya berterima kasih kepada seluruh rekan diskusi yaitu para mahasiswa yang telah mengambil mata kuliah Komputasi Fisika PTA 2006/2007 di Departemen Fisika, FMIPA, Universitas Indonesia. Kepada seluruh mahasiswa dari berbagai universitas di Timur dan di Barat Indonesia juga perlu saya tulis disini sebagai ungkapan terima kasih atas pertanyaan-pertanyaan mereka yang turut memperkaya isi buku ini.

Walaupun buku ini masih jauh dari sempurna, namun semoga ia dapat menyumbangkan kontribusi yang berarti bagi terciptanya gelombang kebangkitan ilmu pengetahuan pada diri anak bangsa Indonesia yang saat ini sedang terpuruk. Saya wariskan ilmu ini untuk siswa dan mahasiswa Indonesia dimanapun mereka berada. Kalian berhak memanfaatkan buku ini. Saya izinkan kalian 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

Bahasa Python

✍ **Objektif :**

- ▷ Mengenalkan Bahasa Python
- ▷ Dasar-dasar Pemrograman Python

1.1 Berkenalan dengan python

Python merupakan bahasa pemrograman yang bersifat *object-oriented* dan sangat efisien untuk men-*develop* software aplikasi di bidang sains dan teknik. *Script* Python tidak perlu di-*compile* kedalam suatu kode mesin apapun, karena ia dapat dijalankan cukup dengan bantuan *interpreter*. Keuntungan dari *script* yang bisa dijalankan dengan interpreter adalah ia dapat diuji dan di*debug* dengan cepat, sehingga programmer bisa lebih berkonsentrasi pada algoritma dibalik *script* yang sedang dibangunnya. Keuntungan ini juga menjadikan aplikasi Python lebih cepat dibangun dibandingkan dengan aplikasi yang sama jika dibangun dengan bahasa C maupun Fortran. Di sisi lain, kekurangan *script* Python selaku *interpreted program* adalah ia tidak dapat di-*compile* menjadi program aplikasi yang bersifat *stand-alone*. Sehingga suatu *script* Python hanya bisa dieksekusi jika pada komputer tersebut sudah terinstall program Python.

1.2 Dasar-dasar pemrograman Python

1.2.1 Variabel

Pada sebagian besar bahasa pemrograman, nama suatu variabel merepresentasikan suatu nilai dengan tipe data tertentu; dan menempati alamat memory yang pasti. Nilai variabel tersebut dapat diganti-ganti, namun tipe data selalu tetap[1]. Tidak demikian dengan Python dimana tipe datanya dapat diubah-ubah secara dinamis. Berikut ini adalah contohnya:

```
>>> b = 2          # b bilangan bertipe integer
>>> print b
```

[1] *Integer* adalah tipe data untuk bilangan bulat; sementara *float* adalah tipe data untuk bilangan pecahan

```
2
>>> b = b * 2.1   # Sekarang b bilangan bertipe float
>>> print b
4.2
```

Tulisan $b = 2$ artinya variabel b diisi dengan angka 2 yang bertipe *integer*. Statemen berikutnya adalah operasi perkalian $b * 2.1$, lalu hasilnya disimpan pada variabel yang sama yaitu variabel b. Dengan demikian nilai b yang lama akan diganti dengan nilai yang baru, yaitu hasil operasi perkalian. Akibatnya, sekarang variabel b memiliki tipe data *float*, suatu tipe yang merepresentasikan bilangan pecahan atau desimal. Nilai variabel b menjadi 4.2.

Tanda pagar (#) menyatakan awal dari suatu komentar. Komentar adalah bagian dari *script* Python yang tidak akan dieksekusi oleh interpreter.

1.2.2 Bilangan *integer* dan *float*

Seperti telah disinggung bahwa Python mengenal bilangan bertipe *integer* dan *float*. Perbedaan tipe bilangan ini berpotensi menimbulkan *bug* (masalah). Ini contohnya

```
>>> 1/2           # bilangan integer dibagi bilangan integer
0                 # tentu saja ini keliru, mestinya 0.5
>>> 1/2.0         # bilangan integer dibagi bilangan float
0.5               # kali ini hasilnya tepat
```

Untuk menghindari kesalahan tersebut, diperlukan sebuah statemen tambahan

```
>>> from __future__ import division
>>> 1/2
0.5
```

Nah, sekarang hasilnya sudah tepat.

1.2.3 Lists

List adalah sejumlah object yang dipisahkan oleh tanda koma (,) dan diapit oleh kurung siku ([ ]). Begini contohnya:

```
>>> a = [1.0, 2.0, 3.0]   # cara membuat list
>>> a.append(4.0)         # tambahkan 4.0 kedalam list
>>> print a
[1.0, 2.0, 3.0, 4.0]
>>> a.insert(0,0.0)       # sisipkan 0.0 pada posisi 0
>>> print a
[0.0, 1.0, 2.0, 3.0, 4.0]
>>> print len(a)          # menentukan panjang list
5
```

Jika kita memberikan statemen $b = a$, maka itu tidak berarti bahwa variabel b terpisah dengan variabel a. Di python, statemen seperti itu diartikan hanya sebagai pemberian nama lain (*alias*) kepada variabel a. Artinya, perubahan yang terjadi baik itu di a ataupun di b, maka hasil akhir mereka berdua akan sama saja. Setiap perubahan yang terjadi di b akan berdampak di a. Untuk meng-copy a secara independen, gunakan statemen $c = a[:]$, sebagaimana dicontohkan berikut ini

```
>>> a = [1.0, 2.0, 3.0]
>>> b = a         # b adalah alias dari a
>>> b[0] = 5.0    # isi elemen b diubah
>>> print a
[5.0, 2.0, 3.0]   # perubahan di b tercermin di a
>>> c = a[:]      # c kopian dari a
>>> c[0] = 1.0    # isi elemen c diubah
>>> print a
[5.0, 2.0, 3.0]   # a tidak dipengaruhi c
```

Matrik dapat dideklarasikan oleh sejumlah list yang disusun berbaris. Berikut adalah matrik 3×3 dalam bentuk list:

```
>>> a = [[1, 2, 3], \
         [4, 5, 6], \
         [7, 8, 9]]
>>> print a[1]     # Print baris kedua (elemen 1)
[4, 5, 6]
>>> print a[1][2]  # Print elemen ketiga dari baris kedua
6
```

Tanda *backslash* adalah karakter yang menandakan bahwa statemen belum selesai. Perlu dicatat disini pula bahwa python menganut *zero offset*, sehingga $a[0]$ merepresentasikan baris pertama, $a[1]$ baris kedua, dst.

1.2.4 Module

Walaupun suatu matrik dapat dideklarasikan oleh sejumlah list, namun kita akan menggunakan cara yang lain untuk mendeklarasikan suatu matrik. Python telah menyediakan suatu *module* untuk mendeklarasikan suatu matrik. Bahkan module tersebut juga menyediakan berbagai operasi matrik. Berikut ini contohnya:

```
>>> from numpy import array
>>> a = array([[2.0, -1.0],[-1.0, 3.0]])
>>> print a
[[ 2. -1.]
[-1. 3.]]
```

Kata *numpy* pada statemen pertama di atas adalah nama sebuah *module* yang dikenali oleh python. Module *numpy* berisi fungsi-fungsi khusus untuk mengolah matrik. Pada statemen pertama, *array* adalah sebuah fungsi khusus yang tersimpan di *module numpy*. Masih banyak fungsi-fungsi lain yang tersimpan di *module numpy* tersebut. Silakan perhatikan baik-baik contoh di bawah ini:

```
>>> from numpy import arange,zeros,ones,float
>>> a = arange(2,10,2)
>>> print a
[2 4 6 8]
>>> b = arange(2.0,10.0,2.0)
>>> print b
[ 2. 4. 6. 8.]
>>> z = zeros((4))
>>> print z
[0 0 0 0]
>>> y = ones((3,3),float)
>>> print y
[[ 1. 1. 1.]
[ 1. 1. 1.]
[ 1. 1. 1.]]
```

1.2.5 Function

Sekarang coba anda buka *python Shell,* lalu hitung akar dua dari 9 dengan mengetikan statemen berikut

```
>>> sqrt(9)
```

Jawaban yang muncul harusnya adalah angka 3.0. Tapi kenapa dilayar monitor malah tampil tulisan seperti ini?

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    sqrt(9)
NameError: name 'sqrt' is not defined
```

Lewat *error message* itu, Python ingin bilang ke kita bahwa fungsi $sqrt()$ tidak dikenal atau tidak terdefinisikan. Bagaimana cara mengatasinya? Coba anda tambahkan module $math$ sebelum menghitung $sqrt$.

```
>>> from math import *
>>> sqrt(9)
3.0
```

Nah, sekarang sudah terpecahkan!

Pada contoh tadi, statemen $sqrt(..angka..)$ adalah fungsi (*function*) yang bertugas untuk mencari akar dua dari suatu angka. Secara umum yang dimaksud dengan *function* adalah statemen yang dieksekusi. Seringkali parameter masukan (*input*) diperlukan oleh *function* untuk dihitung, namun tidak selalu begitu. Contoh-contoh *function* yang lain adalah

```
>>> sin(pi/2)
1.0
>>> tan(pi/4)
0.99999999999999989
>>> exp(10)
22026.465794806718
>>> log(100)
4.6051701859880918
>>> log10(100)
2.0
>>> pow(2,4)
16.0
>>> asin(0.5)
0.52359877559829893
>>> pi/6
0.52359877559829882
```

Function apa saja yang disediakan oleh *math*? Ketikan statemen berikut, anda akan tahu jawabannya.

```
>>> import math
>>> dir(math)
['__doc__', '__name__', 'acos', 'asin', 'atan', 'atan2', 'ceil',
'cos', 'cosh', 'degrees', 'e', 'exp', 'fabs', 'floor', 'fmod',
'frexp', 'hypot', 'ldexp', 'log', 'log10', 'modf', 'pi', 'pow',
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh']
```

1.2.6 Membuat matrik

Sekarang kita fokus membahas cara mendeklarasikan suatu matrik dengan bahasa python. Daripada repot-repot, kita *download* aja semua *function* yang tersimpan di *numpy*.

```
>>> from numpy import *     # tanda * artinya download semua function
>>> A = zeros((3,3),float)  # bikin matrik A berukuran 3x3 diisi nol
>>> print A
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

Langkah pertama, matrik dibuat menggunakan *function zeros* lalu diikuti dengan ukuran matrik, misalnya 3×3. Kemudian elemen-elemen matrik diisi satu persatu dengan angka/bilangan.

```
>>> a[0] = [2.0, 3.1, 1.8] # cara mengisi elemen baris sekaligus
>>> a[1,1] = 5.2           # cara mengisi elemen satu-persatu
>>> print A
[[ 2. 3.1 1.8]
[ 0. 5.2 0. ]
[ 0. 0. 0. ]]
```

Ada cara lain lagi yang lebih sederhana untuk mendeklarasikan sebuah matrik, yaitu cukup dengan dua baris statemen seperti berikut ini

```
>>> from numpy import *
>>> A = array([[1.,2.,3.],[4.,5.,6.],[7.,8.,9.]])
>>> print A
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
```

Atau kalau mau benar-benar mirip dengan format matrik, cara menuliskan statemennya adalah seperti ini

```
>>> from numpy import array  # hanya function array yang dipanggil
>>> A = array([[1.,2.,3.],\
              [4.,5.,6.],\  # posisi kurung siku dibuat rata atas-bawah
              [7.,8.,9.]])
>>> print A
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
```

1.2.7 Cara meng*copy* matrik

Berbeda dengan bahasa C, bahasa Fortran, maupun matlab, sebuah matrik di python tidak bisa di*copy* hanya dengan tanda *sama-dengan*. Bagi python, tanda sama-dengan berfungsi untuk memberi nama **alias** semata terhadap obyek yang sama. Perhatikan contoh berikut

```
>>> A = array([[1.,2.],\
              [3.,4.]])
>>> B = A
>>> B[1,1]=8
>>> A
array([[ 1.,  2.],
       [ 3.,  8.]])
```

```
>>> B
array([[ 1.,  2.],
       [ 3.,  8.]])
```

Contoh di atas memperlihatkan bahwa matrik berukuran 2x2 tersebut memiliki 2 nama yaitu A dan B. Buktinya, jika elemen $B[1,1]$ diganti dengan angka 8, maka perubahan itu bisa dilihat di A, yang mana elemen $A[1,1]$ nya pun akan menyimpan angka 8 juga. Sekarang perhatikan contoh berikut ini

```
>>> A = array([[1.,2.],\
              [3.,4.]])
>>> B = A.copy()
>>> B[1,1]=8
>>> A
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> B
array([[ 1.,  2.],
       [ 3.,  8.]])
```

Dengan perintah $B = A.copy()$, maka matrik A menjadi benar-benar berbeda obyek dengan matrik B. Buktinya, perubahan elemen $B[1,1]$ tidak berefek apa-apa terhadap elemen $A[1,1]$.

1.3 Python Editor

Sebenarnya, penulisan *script* atau *source code* berbahasa python dapat dilakukan menggunakan berbagai editor, misalnya dengan *notepad*-nya *windows*. Akan tetapi demi kenyamanan programmer, sebaiknya penulisan sorce-code menggunakan python editor yang sudah tersedia di dalam *pyhton shell*. Cara memanggil python editor, klik *File* dipojok kiri atas, lalu klik *New Window* atau cukup dengan *Ctrl-N*. Maka sebuah window baru akan terbuka, yaitu window *python editor*. Sebagai bahan uji coba, silakan ketikkan statemen-statemen berikut

```
1 from numpy import array  # hanya function array yang dipanggil
2 A = array([[1.,2.,3.],\
3            [4.,5.,6.],\
4            [7.,8.,9.]])
5 print A
```

Simpanlah (*Save*) file *source-code* di atas dengan nama *matrik01.py*. Untuk mengeksekusi atau menjalankan file tersebut, pilihlah menu *Run*, dilanjutkan dengan *Run Module*. Atau bisa lebih singkat dengan cukup hanya menekan tombol *F5*. Di layar monitor akan tampil hasilnya.

1.4 Looping

Looping artinya adalah pengulangan. Misalnya anda mendapat tugas untuk menghitung akar bilangan-bilangan dari 1 sampai 10. Ada 2 cara untuk menyelesaikan tugas tersebut, pertama, salinlah *source-code* berikut pada *python editor* lalu diberi nama *looping01.py*

```
from numpy import sqrt  # hanya function sqrt yang dipanggil
print sqrt(1)
print sqrt(2)
print sqrt(3)
print sqrt(4)
print sqrt(5)
print sqrt(6)
print sqrt(7)
print sqrt(8)
print sqrt(9)
print sqrt(10)
```

Jalankan *source-code* di atas dengan menekan tombol *F5*, maka akan muncul hasil sebagai berikut

```
1.0
1.41421356237
1.73205080757
2.0
2.2360679775
2.44948974278
2.64575131106
2.82842712475
3.0
3.16227766017
```

Cara kedua dengan teknik looping, yaitu

```
from numpy import sqrt
for i in range(1,10+1):
    print sqrt(i)
```

Simpanlah *source-code* ini dengan nama *looping02.py*, lalu jalankan dengan F5, akan nampak hasil yang sama yaitu

```
1.0
1.41421356237
1.73205080757
2.0
2.2360679775
2.44948974278
2.64575131106
2.82842712475
3.0
3.16227766017
```

Mari sejenak kita bandingkan antara *looping01.py* dan *looping02.py*. Kedua *source-code* itu memiliki tujuan yang sama yaitu menghitung akar bilangan dari 1 sampai 10. Perbedaannya,

looping01.py berisi 11 baris statemen, sedangkan *looping02.py* hanya 3 baris statemen. Coba cek ukuran file-nya! Ukuran file *looping01.py* (di laptop saya) adalah 179 byte, sementara ukuran *looping02.py* adalah 72 byte. Dengan demikian dapat disimpulkan bahwa *looping02.py* lebih efisien dibanding *looping01.py*.

1.5 Optimasi

Dalam konteks *progamming*, upaya-upaya untuk memperkecil jumlah statemen ataupun upaya-upaya untuk memperkecil ukuran file disebut **optimasi**.

1.6 Latihan

1. Buatlah 3 buah source-code untuk menyatakan masing-masing matrik **A**, matrik **B** dan matrik **x** sebagai berikut

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

2. Salinlah source-code ini, lalu jalankan! Jika ada error, anda harus men-debug-nya sampai selesai, lalu setelah itu lakukan langkah optimasi sehingga ia menjadi source-code yang efisien!

```
1 from numpy import sqrt
2 print 1/log10(10)
3 print 2/log10(20)
4 print 3/log10(30)
5 print 4/log10(40)
6 print 5/log10(50)
7 print 6/log10(60)
8 print 7/log10(70)
9 print 8/log10(80)
```

Bab 2

Matrik dan Komputasi

✍ **Objektif :**

▷ Mengenalkan matrik dan jenis-jenis matrik.

▷ Mengenalkan operasi penjumlahan dan perkalian matrik.

▷ Mendeklarasikan elemen-elemen matrik ke dalam memori komputer.

▷ Membuat *source-code* operasi matrik.

2.1 Pengenalan 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 dari elemen-elemen yang dinyatakan dengan huruf kecil diikuti angka-angka indeks, misalnya a_{ij}, dimana indeks *i* menunjukan posisi baris ke-i dan indeks *j* menentukan posisi kolom ke-j.

$$\mathbf{A} = (a_{ij}) = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \tag{2.1}$$

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 $b_{32} = 4$.

2.2 Deklarasi matrik di Python

Deklarasi matrik menggunakan python dilakukan dengan cara yang sudah dibahas pada Bab sebelumnya, maka untuk menginisialisasi matrik A caranya adalah

```
1 from numpy import array
2 A = array([[3,8,5],\
3            [6,4,7]])
```

sedangkan untuk matrik B

```
1 from numpy import array
2 B = array([[1,3],\
3            [5,9],\
4            [2,4]])
```

Ada catatan tambahan yang harus diperhatikan saat kita menginisialisasi matrik menggunakan python, yaitu python memulai indeks elemen matrik dari pasangan (0,0), bukan (1,1). Jadi angka 1 pada matrik B di atas berada pada indeks (0,0). Sementara angka 3 memiliki indeks (0,1). Untuk angka 5 indeks-nya (1,0). Sedangkan angka 9 punya indeks (1,1). Begitu seterusnya. Peta indeks i dan j pada matrik B adalah:

$$\mathbf{B} = (b_{ij}) = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \end{bmatrix}$$

Statemen berikut ini bisa memastikan angka berapa saja yang menempati posisi indeks tertentu pada matriks B

```
>>> B[0,0]
1
>>> B[0,1]
3
>>> B[1,0]
5
>>> B[1,1]
9
```

Dari pemahaman ini, saya ajak anda untuk melihat cara lain menginisialisasi matrik menggunakan python. Perhatikan source-code berikut

```
1 from numpy import zeros # memanggil function zeros
2 B = zeros((3,2))        # mula-mula matrik B(3x2) diberi nilai 0 (nol)
3 B[0,0]=1               # inisialilasi elemen B(0,0) dg angka 1
```

```
B[0,1]=3              # inisialilasi elemen B(0,0) dg angka 3
B[1,0]=5              # inisialilasi elemen B(0,0) dg angka 5
B[1,1]=9              # inisialilasi elemen B(0,0) dg angka 9
B[2,0]=2              # inisialilasi elemen B(0,0) dg angka 2
B[2,1]=4              # inisialilasi elemen B(0,0) dg angka 4
```

Coba anda *Run* source-code ini lalu di-*print*, maka akan didapat

```
>>> print B
[[ 1.  3.]
 [ 5.  9.]
 [ 2.  4.]]
```

2.3 Macam-macam matrik

2.3.1 Matrik transpose

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

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

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

2.3.1.1 Python

Berikut adalah contoh source-code untuk melakukan transpose matrik menggunakan python

```
from numpy import array, zeros
A = array([[3.,8.,5.],\
           [6.,4.,7.]])   # A berukuran 2x3
print A
AT = zeros((3,2))         # inisialisasi A-transpose berukuran 3x2, diberi nilai 0
AT[0,0]=A[0,0]          # tukar posisi antara baris dan kolom
AT[0,1]=A[1,0]          # tukar posisi antara baris dan kolom
AT[1,0]=A[0,1]          # tukar posisi antara baris dan kolom
AT[1,1]=A[1,1]          # tukar posisi antara baris dan kolom
AT[2,0]=A[0,2]          # tukar posisi antara baris dan kolom
AT[2,1]=A[1,2]          # tukar posisi antara baris dan kolom
print AT
```

2.3.2 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

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

2.3.3 Matrik simetrik

Matrik simetrik adalah matrik bujursangkar yang elemen-elemen matrik **A** bernilai sama dengan matrik transpose-nya ($\mathbf{A}^T$).

Contoh 5: Matrik simetrik

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

2.3.4 Matrik diagonal

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

Contoh 6: Matrik diagonal orde 3

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

2.3.5 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

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

Setidaknya, berdasarkan apa-apa yang telah kita pelajari sebelumnya, ada 2 cara untuk menginisialisasi matrik identitas, pertama

```
1 from numpy import array
2 I = array([[1,0,0],\
3            [0,1,0],\
4            [0,0,1]])
```

kedua

```
from numpy import zeros
I = zeros((3,3))
B[0,0]=1
B[0,1]=0
B[0,2]=0
B[1,0]=0
B[1,1]=1
B[1,2]=0
B[2,0]=0
B[2,1]=0
B[2,2]=1
```

Berikut ini saya ketengahkan cara yang ketiga, yang paling mudah dan praktis

```
from numpy import identity  # memanggil function identity
I = identity(3)             # inisialisasi matrik identitas berukuran 3x3
print I
```

2.3.6 Matrik upper-triangular

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

Contoh 8: Matrik upper-triangular

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

2.3.7 Matrik lower-triangular

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

Contoh 9: Matrik lower-triangular

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

2.3.8 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

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

2.3.9 Matrik diagonal dominan

Matrik diagonal dominan adalah matrik bujursangkar yang memenuhi

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \tag{2.2}$$

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.

2.3.10 Matrik *positive-definite*

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

$$\mathbf{x}^T \mathbf{A}\mathbf{x} > 0 \tag{2.3}$$

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.

2.3.11 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

$$\mathbf{a} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & \dots & a_m \end{bmatrix} \tag{2.4}$$

Di dalam Python menjadi

$$\mathbf{a} = \begin{bmatrix} a_{00} & a_{01} & \dots & a_{0m} \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & \dots & a_m \end{bmatrix} \tag{2.5}$$

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

$$\mathbf{a} = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{2.6}$$

Di dalam Python menjadi

$$\mathbf{a} = \begin{bmatrix} a_{00} \\ a_{10} \\ \vdots \\ a_{n0} \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{2.7}$$

2.4 Operasi matematika

2.4.1 Tukar posisi

Operasi matematika yang pertama diperkenalkan adalah tukar posisi (*pivoting*). Misalnya, ada sistem persamaan linear yang tersusun dari 4 persamaan linear P_1 sampai P_4 seperti ini

$$\begin{array}{lrcrcrcrcl} P_1: & x_0 & + & x_1 & & & + & 3x_3 & = & 4 \\ P_2: & 2x_0 & + & x_1 & - & x_2 & + & x_3 & = & 1 \\ P_3: & 3x_0 & - & x_1 & - & x_2 & + & 2x_3 & = & \text{-3} \\ P_4: & -x_0 & + & 2x_1 & + & 3x_2 & - & x_3 & = & 4 \end{array}$$

Jika P_2 ditukar dengan P_3, maka susunannya menjadi

$$\begin{array}{lrcrcrcrcl} P_1: & x_0 & + & x_1 & & & + & 3x_3 & = & 4 \\ P_3: & 3x_0 & - & x_1 & - & x_2 & + & 2x_3 & = & \text{-3} \\ P_2: & 2x_0 & + & x_1 & - & x_2 & + & x_3 & = & 1 \\ P_4: & -x_0 & + & 2x_1 & + & 3x_2 & - & x_3 & = & 4 \end{array}$$

Salah satu *source code* untuk melakukan tukar posisi adalah sebagai berikut

```
from numpy import array
A = array([[1.,1.,0.,3.,4],\
           [2.,1.,-1.,1.,1],\
           [3.,-1.,-1.,2.,-3],\
           [-1.,2.,3.,-1,4]])  # angka yang paling kanan adalah angka
                               # disebelah kanan tanda sama dengan
print A

#--- operasi tukar posisi ---
for i in range(0,5):
    v=A[1,i]
    u=A[2,i]
    A[1,i]=u
    A[2,i]=v

print A
```

2.4.2 Penjumlahan matrik

Operasi penjumlahan pada dua buah matrik hanya bisa dilakukan bila kedua matrik tersebut berukuran sama. Misalnya matrik $\mathbf{C}_{2\times 3}$

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

dijumlahkan dengan matrik $\mathbf{A}_{2\times 3}$, lalu hasilnya (misalnya) dinamakan matrik $\mathbf{D}_{2\times 3}$

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

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

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

$$\begin{bmatrix}d_{00} & d_{01} & d_{02}\\ d_{10} & d_{11} & d_{12}\end{bmatrix} = \begin{bmatrix}a_{00}+c_{00} & a_{01}+c_{01} & a_{02}+c_{02}\\ a_{10}+c_{10} & a_{11}+c_{11} & a_{12}+c_{12}\end{bmatrix}$$

Jika diuraikan satu persatu, dapat dinyatakan sebagai berikut

$$\begin{aligned}d_{00} &= a_{00} + c_{00}\\ d_{01} &= a_{01} + c_{01}\\ d_{02} &= a_{02} + c_{02}\\ d_{10} &= a_{10} + c_{10}\\ d_{11} &= a_{11} + c_{11}\\ d_{12} &= a_{12} + c_{12}\end{aligned} \tag{2.8}$$

Dari sini dapat diturunkan sebuah rumus umum penjumlahan dua buah matrik

$$d_{ij} = a_{ij} + c_{ij} \tag{2.9}$$

dimana i=0,1 dan j=0,1,2. Berdasarkan persamaan (2.8), dengan mudah anda bisa melihat bahwa indeks i hanya mengalami dua kali perubahan, yaitu 0 dan 1. Sementara indeks j berubah dari 0 menjadi 1 kemudian 2.

2.4.3 Komputasi penjumlahan matrik

Berdasarkan contoh operasi penjumlahan di atas, indeks j lebih cepat berubah dibanding indeks i sebagaimana ditulis pada 3 baris pertama persamaan (2.8),

$$\begin{aligned}d_{00} &= a_{00} + c_{00}\\ d_{01} &= a_{01} + c_{01}\\ d_{02} &= a_{02} + c_{02}\end{aligned}$$

Jelas terlihat, saat indeks i masih bernilai 0, indeks j sudah berubah dari nilai 0 sampai 2. Hal ini membawa konsekuensi pada *script* pemrograman, dimana *looping* untuk indeks j harus diletakkan didalam *looping* indeks i. **Pokoknya yang *looping*-nya paling cepat harus dile-**

takkan paling dalam; sebaliknya, *looping* paling luar adalah *looping* yang indeksnya paling jarang berubah.

Di Python, angka indeks terkecil dimulai dari 0 (nol), bukan dari 1 (satu). Pada source-code ini, walaupun batas atas i tertulis sampai angka 2, namun Python hanya mengolahnya sampai angka 1 saja. Demikian pula dengan indeks j, ia hanya sampai angka 2 saja

```
for i in range(0,2):
    for j in range(0,3):
        D[i,j]=A[i,j]+C[i,j]
```

Perhatikan source-code di atas! Penulisan indeks i harus didahulukan daripada indeks j, karena dalam contoh uraian diatas, indeks j lebih cepat berubah dibanding indeks i.

Perlu dicatat bahwa ukuran matrik tidak terbatas hanya 2x3. Tentu saja anda bisa mengubah ukurannya sesuai dengan keperluan atau kebutuhan anda. Jika ukuran matrik dinyatakan secara umum sebagai n x m, dimana n adalah jumlah baris dan m adalah jumlah kolom, maka bentuk pernyataan komputasinya menjadi

```
for i in range(0,n):
    for j in range(0,m):
        D[i,j]=A[i,j]+C[i,j]
```

Sekarang, mari kita lengkapi dengan contoh sebagai berikut: diketahui matrik $\mathbf{A}_{2\times 3}$

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

dan matrik $\mathbf{C}_{2\times 3}$

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

Program untuk menjumlahkan kedua matrik tersebut adalah:

```
from numpy import array, zeros
A = array([[3.,8.,5.],\
           [6.,4.,7.]])   # A berukuran 2x3
C = array([[9.,5.,3.],\
           [7.,2.,1.]])   # C berukuran 2x3
n=2
m=3
D = zeros((n,m))
for i in range(0,n):
    for j in range(0,m):
        D[i,j]=A[i,j]+C[i,j]
print D
```

2.4.4 Perkalian matrik

Sekarang kita beralih ke operasi 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 $\mathbf{A}_{2\times3}$ dikalikan dengan matrik $\mathbf{B}_{3\times2}$, lalu hasilnya (misalnya) dinamakan matrik $\mathbf{E}_{2\times2}$

$$\mathbf{E}_{2\times2} = \mathbf{A}_{2\times3}.\mathbf{B}_{3\times2}$$

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

Tanpa mempedulikan nilai elemen-elemen pada masing-masing matrik, operasi perkalian antara matrik $\mathbf{A}_{2\times3}$ dan $\mathbf{B}_{3\times2}$, dapat dinyatakan dalam indeks masing-masing matrik tersebut, yaitu

$$\begin{bmatrix} e_{00} & e_{01} \\ e_{10} & e_{11} \end{bmatrix} = \begin{bmatrix} a_{00}.b_{00} + a_{01}.b_{10} + a_{02}.b_{20} & a_{00}.b_{01} + a_{01}.b_{11} + a_{02}.b_{21} \\ a_{10}.b_{00} + a_{11}.b_{10} + a_{12}.b_{20} & a_{10}.b_{01} + a_{11}.b_{11} + a_{12}.b_{21} \end{bmatrix}$$

Bila dijabarkan, maka elemen-elemen matrik $\mathbf{E}_{2\times2}$ adalah

$$\begin{aligned}
e_{00} &= a_{00}.b_{00} + a_{01}.b_{10} + a_{02}.b_{20} \\
e_{01} &= a_{00}.b_{01} + a_{01}.b_{11} + a_{02}.b_{21} \\
e_{10} &= a_{10}.b_{00} + a_{11}.b_{10} + a_{12}.b_{20} \\
e_{11} &= a_{10}.b_{01} + a_{11}.b_{11} + a_{12}.b_{21}
\end{aligned} \tag{2.10}$$

Sejenak, mari kita amati perubahan pasangan angka-angka indeks yang mengiringi elemen e, a dan b. Perhatikan perubahan angka pertama pada indeks elemen e seperti berikut ini

$$\begin{aligned}
e_{0..} &= .. \\
e_{0..} &= .. \\
e_{1..} &= .. \\
e_{1..} &= ..
\end{aligned}$$

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

$$\begin{aligned}
e_{0..} &= a_{0..}.b_{...} + a_{0..}.b_{...} + a_{0..}.b_{...} \\
e_{0..} &= a_{0..}.b_{...} + a_{0..}.b_{...} + a_{0..}.b_{...} \\
e_{1..} &= a_{1..}.b_{...} + a_{1..}.b_{...} + a_{1..}.b_{...} \\
e_{1..} &= a_{1..}.b_{...} + a_{1..}.b_{...} + a_{1..}.b_{...}
\end{aligned}$$

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

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

dimana i bergerak mulai dari angka 0 hingga angka 1, atau kita nyatakan i=0,1. Selanjutnya, masih dari persamaan (2.10), marilah kita perhatikan perubahan angka kedua pada indeks elemen e dan elemen b,

$$e_{i0} = a_{i..}.b_{..0} + a_{i..}.b_{..0} + a_{i..}.b_{..0}$$
$$e_{i1} = a_{i..}.b_{..1} + a_{i..}.b_{..1} + a_{i..}.b_{..1}$$
$$e_{i0} = a_{i..}.b_{..0} + a_{i..}.b_{..0} + a_{i..}.b_{..0}$$
$$e_{i1} = a_{i..}.b_{..1} + a_{i..}.b_{..1} + a_{i..}.b_{..1}$$

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 0 hingga angka 1, atau kita nyatakan j=0,1. Selanjutnya, masih dari persamaan (2.10), mari kita perhatikan perubahan angka indeks pada elemen a dan elemen b, dimana kita akan dapati pola sebagai berikut

$$e_{ij} = a_{i0}.b_{0j} + a_{i1}.b_{1j} + a_{i2}.b_{2j}$$
$$e_{ij} = a_{i0}.b_{0j} + a_{i1}.b_{1j} + a_{i2}.b_{2j}$$
$$e_{ij} = a_{i0}.b_{0j} + a_{i1}.b_{1j} + a_{i2}.b_{2j}$$
$$e_{ij} = a_{i0}.b_{0j} + a_{i1}.b_{1j} + a_{i2}.b_{2j}$$

Dan kita bisa mencantumkan huruf k sebagai pengganti angka-angka indeks yang polanya sama, dimana k bergerak mulai dari angka 0 hingga angka 2, atau kita nyatakan k=0,1,2.

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

Kemudian secara sederhana dapat ditulis sebagai berikut

$$e_{ij} = a_{ik}.b_{kj} + a_{ik}.b_{kj} + a_{ik}.b_{kj} \tag{2.11}$$

Selanjutnya dapat ditulis pula formula berikut

$$e_{ij} = \sum_{k=0}^{2} a_{ik} b_{kj} \tag{2.12}$$

dimana i=0,1; j=0,1; dan k=0,1,2.
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

$$e_{ij} = \sum_{k=0}^{m} a_{ik} b_{kj} \tag{2.13}$$

dengan i=0,1,...,n; j=0,1...,p; dan k=0,1...,m.

2.4.5 Komputasi perkalian matrik

Komputasi operasi perkalian antara matrik $\mathbf{A}_{2\times 3}$ dan $\mathbf{B}_{3\times 2}$ dilakukan melalui 2 tahap; **pertama** adalah memberikan nilai 0 (nol) pada elemen-elemen matrik $\mathbf{E}_{2\times 2}$ dengan cara

```
from numpy import array, zeros
E = zeros((2,2))   #ukuran matrik E adalah 2x2
```

kedua adalah menghitung perkalian matrik dengan cara

```
for i in range(0,2):
    for j in range(0,2):
        for k in range(0,3):
            E[i,j]=E[i,j]+A[i,k]*B[k,j]
```

Sebentar.., sebelum dilanjut tolong perhatikan penempatan indeks i, j dan k pada *script* di atas. Mengapa indeks i didahulukan daripada indeks j dan k? Ini bukan sesuatu yang kebetulan. Dan ini juga bukan sekedar mengikuti urutan huruf abjad i,j,k. Sekali lagi ingin saya tegaskan bahwa penempatan yang demikian semata-mata mengikuti aturan umum yaitu ***looping*** **yang**

indeksnya berubah paling cepat harus diletakkan paling dalam; sebaliknya, *looping* paling luar adalah *looping* yang indeksnya paling jarang berubah. Kalau anda perhatikan dengan teliti, pasti anda akan menemukan fakta bahwa indeks k paling cepat berubah. Kemudian disusul oleh indeks j. Lalu yang paling jarang berubah adalah indeks i. Itulah sebabnya, penempatan urutan indeks pada script di atas harus dimulai dari i terlebih dahulu sebagai *looping* terluar, kemudian indeks j, dan yang terakhir indeks k sebagai *looping* terdalam.

Tentu saja anda bisa mengubah ukuran matrik-nya sesuai dengan keperluan atau kebutuhan anda. Jika ukuran matrik **A** dinyatakan secara umum sebagai n x m dan matrik **B** berukuran m x p, maka bentuk pernyataan komputasinya dalam Python

```
E = zeros((n,p))
for i in range(0,n):
    for j in range(0,p):
        for k in range(0,m):
            E[i,j]=E[i,j]+A[i,k]*B[k,j]
```

dimana akan diperoleh hasil berupa matrik **E** yang berukuran n x p. Source-code lengkap untuk contoh soal yang ada di atas adalah

```
from numpy import array, zeros
A = array([[3.,8.,5.],\
           [6.,4.,7.]])   # A berukuran 2x3
B = array([[1.,3.],\
           [5.,9.],\
           [2.,4.]])      # B berukuran 3x2
n=2    # jumlah baris matrik A
m=3    # jumlah kolom matrik A sekaligus jumlah baris matrik B
p=2    # jumlah kolom matrik B
E = zeros((n,p))
for i in range(0,n):
    for j in range(0,p):
        for k in range(0,m):
            E[i,j]=E[i,j]+A[i,k]*B[k,j]
```

2.4.6 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

$$\begin{bmatrix}y_0\\y_1\end{bmatrix} = \begin{bmatrix}a_{00}.x_0 + a_{01}.x_1 + a_{02}.x_2\\ a_{10}.x_0 + a_{11}.x_1 + a_{12}.x_2\end{bmatrix}$$

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

$$\begin{aligned}y_0 &= a_{00}.x_0 + a_{01}.x_1 + a_{02}.x_2\\ y_1 &= a_{10}.x_0 + a_{11}.x_1 + a_{12}.x_2\end{aligned}$$

kemudian secara sederhana dapat diwakili oleh rumus berikut

$$y_i = \sum_{j=0}^{2} a_{ij}x_j$$

dimana i=0,1.
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

$$y_i = \sum_{j=0}^{m} a_{ij}x_j \tag{2.14}$$

dengan i=0,1,... ,n.

2.4.7 Komputasi perkalian matrik dan vektor-kolom

Sama seperti perkalian dua matrik, komputasi untuk operasi perkalian antara matrik **A** berukuran n x m dan vektor-kolom **x** berukuran m dilakukan melalui 2 tahap; pertama adalah memberikan nilai 0 (nol) pada elemen-elemen vektor-kolom **y** yang berukuran n. Lalu tahap kedua adalah melakukan proses perkalian. Kedua tahapan ini digabung jadi satu dalam program berikut ini

```
1 E = zeros((n,1))
2 for i in range(0,n):
```

```
    for k in range(0,m):
        E[i,0]=E[i,0]+A[i,k]*x[k,0]
```

Dengan demikian penyelesaian contoh di atas adalah

```
from numpy import array, zeros
A = array([[3.,8.,5.],\
           [6.,4.,7.]])   # matrik A berukuran 2x3
x = array([[2.],\
           [3.],\
           [4.]])         # vektor x berukuran 3x1
n=2    # jumlah baris matrik A
m=3    # jumlah kolom matrik A sekaligus jumlah baris vektor x
E = zeros((n,1))
for i in range(0,n):
    for k in range(0,m):
        E[i,0]=E[i,0]+A[i,k]*x[k,0]
```

2.5 Penutup

Demikianlah catatan singkat dan sederhana mengenai jenis-jenis matrik dasar yang seringkali dijumpai dalam pengolahan data secara numerik. Semuanya akan dijadikan acuan pada bab berikutnya.

2.6 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 3 source-code untuk melakukan transpose matrik A, matrik B dan vektor x, lalu ujilah ketiga source-code tersebut apakah sudah bisa berjalan sesuai tujuannya!
2. Buatlah source-code untuk menyelesaikan penjumlahan matrik **A** dan matrik **B**.
3. Buatlah source-code untuk menyelesaikan perkalian matrik **A** dan matrik **B**.
4. Buatlah source-code untuk menyelesaikan perkalian matrik **A** dan vektor **x**.
5. Buatlah source-code untuk menyelesaikan perkalian matrik **B** dan vektor **x**.

Bab 3

Function dan *Module*

✍ **Objektif :**

▷ Mengenalkan cara membuat *function* di Python.

▷ Mengenalkan cara membuat *module* di Python.

3.1 Akhir dari sebuah kebersamaan

Pada bab terdahulu, contoh-contoh *source-code* selalu menggunakan ***function*** *array*, *zeros* dan kadang *transpose*. *Function-function* tersebut tersimpan didalam ***module*** *numpy*. Secara default, Python memiliki *module numpy* berikut *function-function* yang ada didalamnya. Pada bab ini, kita akan mempelajari lebih jauh tentang bagaimana caranya membuat *function* dan *module* pribadi. Saya kira, topik ini cukup sulit. Ia bagaikan sebuah etape yang menanjak penuh tikungan. Karenanya, bagi yang masih gemar meng-*copy* atau menyalin PR dari teman dekat, bisa jadi inilah saat-saat terakhir kebersamaan kita (dalam konteks perkuliahan). Maksud saya, setelah hari ini kemungkinan anda akan kepayahan untuk bisa mengerti alur berfikir saya dan sulit memahami apa-apa yang saya ucapkan. Namun semoga itu tidak terjadi.

3.2 *Function*

Function bisa dianggap sebagai sebuah cara untuk mengekspresikan rumus atau formula matematika. Misalnya diketahui formula sebagai berikut

$$y = a^2 + 2 * b + c$$

Jika $a = 2$, $b = 3$ dan $c = 4$, maka dari *python-shell*, komputer bisa disuruh menghitung nilai y

```
>>> a=2
>>> b=3
>>> c=4
>>> y=a*a+2*b+c
```

```
>>> print y
14
```

cara lain adalah dengan membuat file $rumus1.py$, misalnya

```
1 a = 2
2 b = 3
3 c = 4
4 y = a*a+2*b+c
5 print y
```

Ada cara baru yang kemudian akan menjadi bahasan utama bab ini, yaitu dengan membuat *function*. Langkah pertama adalah membuat sebuah file yang berisi statemen-statemen berikut ini

```
1 def formula (a,b,c):
2     y = a*a+2*b+c
3     return y
```

kemudian simpanlah file tersebut dengan nama *fungsi.py*, lalu di-*run* dengan cara menekan tombol $F5$. Selanjutnya, dari *python-shell*, anda bisa memanggil *function* formula

```
>>> formula (2,3,4)
14
```

coba lagi dengan angka yang lain

```
>>> formula (7,8,9)
74
>>> formula (5.,8.,1.)
42.0
```

Jadi dengan adanya *function formula*, anda tidak perlu lagi mengetikan $y = a * a + 2 * b + c$ berulang kali. *Function* merupakan kumpulan statemen yang bisa dipanggil berulang kali tanpa harus mengetikan *source-code*-nya berkali-kali. *Function* yang baru saja kita buat di atas bisa dikembangkan menjadi seperti ini[1]

```
1 def formula (a,b,c):
2     y = a*a+2*b+c
3     x = a*b*c
4     return y,x
```

Ketika *function* tersebut dipanggil

```
>>> formula (2,3,4)
(14, 24)
```

[1]Jangan lupa untuk men-*save* dan men-*run* setiap kali ada modifikasi di dalam *function*.

angka pertama, yaitu 14 adalah milik y, sedangkan angka kedua (24) adalah miliknya x. Kalau dicoba dengan angka yang lain

```
>>> formula (7,8,9)
(74, 504)
>>> formula (5.,8.,1.)
(42.0, 40.0)
```

3.2.1 Mencari akar persamaan kuadrat

Rumus umum dari persamaan kuadrat adalah sebagai berikut

$$y = ax^2 + bx + c$$

akar-akar persamaan kuadrat dapat dicari dengan rumus abc

$$x_1 = \frac{-b + \sqrt{D}}{2a}$$
$$x_2 = \frac{-b - \sqrt{D}}{2a}$$

dimana D disebut diskriminan

$$D = b^2 - 4ac$$

Bergantung pada nilai a, b dan c, nilai diskriminan bisa

D	$>$	0	memiliki 2 akar berbeda
D	$=$	0	memiliki 2 akar kembar
D	$<$	0	memiliki 2 akar kompleks

Function berikut ini bisa dihandalkan untuk mencari akar-akar persamaan kuadrat

```
1  from __future__ import division
2  from numpy import sqrt,complex
3
4  def rumusabc(a,b,c):
5      D = b*b-4*a*c      # menghitung deskriminan D
6      if D > 0.0:        # jika D positif
7          x1 = (-b+sqrt(D))/(2*a)
8          x2 = (-b-sqrt(D))/(2*a)
9      elif D == 0.0:     # jika D sama dengan nol
10         x1 = -b/(2*a)
11         x2 = -b/(2*a)
12     else:              # dan jika D negatif
13         D = -D
14         x1r = -b/(2*a)
15         x1i = sqrt(D)/(2*a)
16         x1 = complex(x1r,x1i)
17         x2r = x1r
18         x2i = -x1i
```

```
19          x2 = complex(x2r,x2i)
20      return x1,x2
```

Silakan anda *save* dan jalankan lalu cek hasilnya[2]

```
>>> rumusabc(1,3,2)
(-1.0, -2.0)
>>> rumusabc(1,-4,4)
(2.0, 2.0)
>>> rumusabc(4,4,2)
((-0.5+0.5j), (-0.5-0.5j))
```

3.2.2 Perkalian matrik dan vektor

Sekarang kita akan membuat *function* untuk perkalian matrik dan vektor. Berdasarkan *source-code* yang lalu (di bab sebelumnya) perkalian matrik dan vektor adalah sebagai berikut

```
1  from numpy import array, zeros
2  A = array([[3.,8.,5.],\
3             [6.,4.,7.]])   # matrik A berukuran 2x3
4  x = array([[2.],\
5             [3.],\
6             [4.]])         # vektor x berukuran 3x1
7  n=2    # jumlah baris matrik A
8  m=3    # jumlah kolom matrik A sekaligus jumlah baris vektor x
9  E = zeros((n,1))
10 for i in range(0,n):
11     for k in range(0,m):
12         E[i,0]=E[i,0]+A[i,k]*x[k,0]
```

Source-code ini memiliki dua bagian utama, yaitu bagian inisialisasi dan bagian operasi perkalian. Bagian operasi perkalian akan dipisah menjadi sebuah *function* bernama *matxvek* yang dikhususkan untuk melayani operasi perkalian matrik dan vektor.

```
1  from numpy import zeros
2
3  def matxvek (A,x):
4      n=2   # jumlah baris matrik A
5      m=3   # jumlah kolom matrik A sekaligus jumlah baris vektor x
6      E = zeros((n,1))
7      for i in range(0,n):
8          for k in range(0,m):
9              E[i,0]=E[i,0]+A[i,k]*x[k,0]
10     return E
```

Simpanlah file ini dengan nama *komputasi.py*, lalu di-*run*. Kemudian ketikan *code* berikut pada *python-shell*

```
>>> from numpy import array
```

[2]Statemen pertama pada *function* ini, yaitu *from __future__ import division* telah dibahas pada Bab 1.

```
>>> A = array([[3.,8.,5.],\
               [6.,4.,7.]])
>>> x = array([[2.],\
               [3.],\
               [4.]])
>>> E = matxvek(A,x)
>>> print E
[[ 50.]
 [ 52.]]
```

Berhasil! *Function matxvek* bekerja sempurna.

Sebelum dilanjut, ada sedikit modifikasi yang diperlukan pada *function matxvek* yang tersimpan pada file *komputasi.py*, yaitu yang menyangkut inisialisasi nilai n dan m. Perhatikan perubahannya berikut ini (saya harap anda mengerti tanpa harus saya jelaskan).

```
1  from numpy import zeros, transpose
2
3  def matxvek (A,x):
4      n=len(A)   # cara lain untuk menghitung jumlah baris matrik A
5      m=len(transpose(A))   # cara lain untuk menghitung jumlah kolom matrik A dan baris vektor x
6      E = zeros((n,1))
7      for i in range(0,n):
8          for k in range(0,m):
9              E[i,0]=E[i,0]+A[i,k]*x[k,0]
10     return E
```

Simpanlah file ini, lalu di-*run* kembali. Ulangi lagi ketikan *code* berikut pada *python-shell*

```
>>> from numpy import array
>>> A = array([[3.,8.,5.],\
               [6.,4.,7.]])
>>> x = array([[2.],\
               [3.],\
               [4.]])
>>> E = matxvek(A,x)
>>> print E
[[ 50.]
 [ 52.]]
```

Modifikasi pada *function matxvek* bertujuan agar *function matxvek* bisa berlaku pada matrik A yang ukurannya bukan 2×3 saja. Contohnya

```
>>> A = array([[3.,8.,5.],\
               [6.,4.,7.],\
               [2.,1.,9.]])
>>> x = array([[2.],\
               [3.],\
```

```
               [4.]])
>>> E = matxvek(A,x)
>>> print E
[[ 50.]
 [ 52.]
 [ 43.]]
```

Jelas terlihat disini bahwa *function matxvek* bisa digunakan untuk ukuran matrik A yang bervariasi.

3.3 *Module*

Sebelum mendiskusikan *module,* silakan buka lagi file *fungsi.py* yang menyimpan *function formula* yang lalu

```
1  def formula (a,b,c):
2      y = a*a+2*b+c
3      return y
```

Function formula akan saya letakkan di dalam *komputasi.py*

```
1  from numpy import zeros, transpose
2
3  def matxvek (A,x):
4      n=len(A)
5      m=len(transpose(A))
6      E = zeros((n,1))
7      for i in range(0,n):
8          for k in range(0,m):
9              E[i,0]=E[i,0]+A[i,k]*x[k,0]
10     return E
11
12 def formula (a,b,c):
13     y = a*a+2*b+c
14     x = a*b*c
15     return y,x
```

Kemudian *function rumusabc* juga digabung ke dalam *komputasi.py*. Perhatikan perubahan yang terjadi di *source-code,* khususnya pada statemen awal

```
1  from __future__ import division
2  from numpy import zeros, transpose, sqrt,complex
3
4  def matxvek (A,x):
5      n=len(A)
6      m=len(transpose(A))
7      E = zeros((n,1))
8      for i in range(0,n):
9          for k in range(0,m):
10             E[i,0]=E[i,0]+A[i,k]*x[k,0]
11     return E
12
```

```
13 def formula (a,b,c):
14     y = a*a+2*b+c
15     x = a*b*c
16     return y,x
17 
18 def rumusabc (a,b,c):
19     D = b*b-4*a*c
20     if D > 0.0:
21         x1 = (-b+sqrt(D))/(2*a)
22         x2 = (-b-sqrt(D))/(2*a)
23     elif D == 0.0:
24         x1 = -b/(2*a)
25         x2 = -b/(2*a)
26     else:
27         D = -D
28         x1r = -b/(2*a)
29         x1i = sqrt(D)/(2*a)
30         x1 = complex(x1r,x1i)
31         x2r = x1r
32         x2i = -x1i
33         x2 = complex(x2r,x2i)
34     return x1,x2
```

jangan lupa di-*save* dan di-*run*. Kemudian buatlah sebuah file *cobamodul.py* yang isinya sebagai berikut

```
1  from numpy import array
2  from komputasi import matxvek
3  
4  A = array([[3.,8.,5.],\
5             [2.,1.,9.],\
6             [6.,4.,7.]])   # matrik A berukuran 3x3
7  x = array([[2.],\
8             [3.],\
9             [4.]])         # vektor x berukuran 3x1
10 hasil = matxvek(A,x)
11 print hasil
```

silakan anda *save* dan *run*. Inilah hasilnya

```
>>>
[[ 50.]
 [ 43.]
 [ 52.]]
```

Sampai disini anda telah berhasil membuat *module* pribadi, yaitu *module komputasi*. Sekarang saatnya membahas *module*. *Module* berisi kumpulan *function*. Kita baru saja membuat *module* yang diberi nama *komputasi*, dimana didalamnya terdapat 3 buah *function* yaitu *matxvek*, *formula* dan *rumusabc*. Untuk melihat seluruh *function* yang ada di *module komputasi*, ketikkan *code* berikut ini

```
>>> import komputasi
>>> dir(komputasi)
```

```
['__builtins__', '__doc__', '__file__', '__name__', 'complex',
'division', 'formula', 'matxvek', 'rumusabc', 'sqrt',
'transpose', 'zeros']
```

Yup, disana sudah ada *matxvek*, *formula* dan *rumusabc*. Tapi mengapa *transpose* dan *zeros* bisa tersimpan juga di *module komputasi*?

3.4 Latihan

1. Buatlah *module* baru yang berisi *function matxvek*, *function matxmat* (untuk operasi perkalian 2 matrik), *function matplusmat* (untuk operasi penjumlahan 2 matrik), *function rumusabc* dan *function formula*. *Module* diberi nama sesuai dengan nama anda masing-masing.

2. Buatlah source-code untuk menyelesaikan penjumlahan matrik **A** dan matrik **B** dengan memanfaatkan module yang selesai dibuat pada nomor sebelumnya.

3. Buatlah source-code untuk menyelesaikan perkalian matrik **A** dan matrik **B** dengan memanfaatkan module yang selesai dibuat pada nomor sebelumnya.

4. Buatlah source-code untuk menyelesaikan perkalian matrik **A** dan vektor **x** dengan memanfaatkan module yang selesai dibuat pada nomor sebelumnya.

5. Buatlah source-code untuk menyelesaikan perkalian matrik **B** dan vektor **x** dengan memanfaatkan module yang selesai dibuat pada nomor sebelumnya.

Berikut ini adalah matrik **A**, matrik **B**, dan vektor **x** yang digunakan untuk menyelesaikan soal latihan di atas

$$\mathbf{A} = \begin{bmatrix} 0.1 & 2.3 & -9.6 & -2.7 \\ 21.5 & 2.9 & 1.7 & 5.6 \\ 2.13 & 4.29 & 8.72 & -1.02 \\ -2.3 & 1.24 & -0.18 & 7.3 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 8.3 & 1.6 & 4.8 & 21.2 \\ 3.4 & 10.5 & 5.2 & 0.1 \\ 7.8 & -2.7 & 9.4 & -5.1 \\ 2.7 & -12.3 & -18.9 & 50.7 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 23.78 \\ -7.97 \\ 8.369 \\ 4.112 \end{bmatrix}$$

Bab 4

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.

4.1 Sistem persamaan linear

Secara umum, sistem persamaan linear dinyatakan sebagai berikut

$$P_j : \qquad a_{n1}x_0 + a_{n2}x_1 + ... + a_{nn}x_n = b_n \tag{4.1}$$

dimana a dan b merupakan konstanta, x adalah variable, sementara nilai $n = 0, 1, 2, 3, ...$ dan $j = 1, 2, 3, ...$

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

$$\begin{array}{lrcrcrcrcl} P_1: & x_0 & + & x_1 & & & + & 3x_3 & = & 4 \\ P_2: & 2x_0 & + & x_1 & - & x_2 & + & x_3 & = & 1 \\ P_3: & 3x_0 & - & x_1 & - & x_2 & + & 2x_3 & = & \text{-3} \\ P_4: & -x_0 & + & 2x_1 & + & 3x_2 & - & x_3 & = & 4 \end{array}$$

Problem dari sistem persamaan linear adalah bagaimana mencari angka-angka yang tepat untuk menggantikan variabel x_0, x_1, x_2, dan x_3 sehingga semua persamaan di atas menjadi benar. Untuk mendapatkan solusi tersebut, diperlukan langkah-langkah penyederhanaan sistem persamaan linear.

4.2 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

$$P_1 : \qquad x_0 + x_1 + 3x_3 = 4$$

  jika $\lambda = 2$, maka

$$2P_1 : \qquad 2x_0 + 2x_1 + 6x_3 = 8$$

- 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

$$\begin{aligned} P_2 : \qquad 2x_0 + x_1 - x_2 + x_3 &= 1 \\ 2P_1 : \qquad 2x_0 + 2x_1 + 6x_3 &= 8 \end{aligned}$$

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

$$P_2 : \qquad -x_1 - x_2 - 5x_3 = -7$$

  dimana variabel x_0 berhasil dihilangkan dari P_2.

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

$$\begin{aligned} P_2 : \qquad 2x_0 + x_1 - x_2 + x_3 &= 1 \\ P_3 : \qquad 3x_0 - x_1 - x_2 + 2x_3 &= -3 \end{aligned}$$

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

$$\begin{aligned} P_2 : \qquad 3x_0 - x_1 - x_2 + 2x_3 &= -3 \\ P_3 : \qquad 2x_0 + x_1 - x_2 + x_3 &= 1 \end{aligned}$$

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

$$\begin{aligned} P_1 : \qquad 3x_0 + 2x_1 - 5x_2 + 8x_3 &= 3 \\ P_2 : \qquad 4x_0 + 7x_1 - x_2 + 6x_3 &= 9 \end{aligned}$$

lalu anda diminta untuk menghilangkan variabel x_0 dari P_2. Itu artinya, anda diminta untuk memodifikasi P_2. 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_0 dari P_2 lewat operasi $(P_2 - \frac{4}{3}P_1)$. Selengkapnya adalah sebagai berikut

$$\begin{array}{rrcl} P_2: & 4x_0 + 7x_1 - x_2 + 6x_3 & = & 9 \\ \frac{4}{3}P_1: & \frac{4}{3}3x_0 + \frac{4}{3}2x_1 - \frac{4}{3}5x_2 + \frac{4}{3}8x_3 & = & \frac{4}{3}3 \end{array}$$

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

$$P_2: \quad \left(4 - \frac{4}{3}3\right)x_0 + \left(7 - \frac{4}{3}2\right)x_1 - \left(1 - \frac{4}{3}5\right)x_2 + \left(6 - \frac{4}{3}8\right)x_3 = \left(9 - \frac{4}{3}3\right)$$

Dengan sendirinya x_0 akan lenyap dari $P2$. Mudah-mudahan jelas sampai disini.

Sekarang, mari kita tinjau hal yang sama, yaitu menghilangkan x_0 dari P_2, namun menggunakan 'permainan' indeks[1]. Secara umum, P_1 dan P_2 bisa dinyatakan sebagai

$$\begin{array}{rrcl} P_1: & a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + a_{03}x_3 & = & a_{04} \\ P_2: & a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + a_{13}x_3 & = & a_{14} \end{array}$$

Agar x_0 hilang dari P_2, operasi yang benar adalah $(P_2 - \lambda P_1) \rightarrow (P_2)$, dimana $\lambda = \frac{a_{10}}{a_{00}}$. Dengan demikian, P_2 yang baru akan memenuhi

$$P_2: \left(a_{10} - \frac{a_{10}}{a_{00}}a_{00}\right)x_0 + \left(a_{11} - \frac{a_{10}}{a_{00}}a_{01}\right)x_1 + \left(a_{12} - \frac{a_{10}}{a_{00}}a_{02}\right)x_2 + \left(a_{13} - \frac{a_{10}}{a_{00}}a_{03}\right)x_3 = \left(a_{14} - \frac{a_{10}}{a_{00}}a_{04}\right)$$

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 sodorkan dihadapan anda tiga buah persamaan, yaitu P_1, P_2 dan P_3

$$\begin{array}{rrcl} P_1: & a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + a_{03}x_3 & = & a_{04} \\ P_2: & a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + a_{13}x_3 & = & a_{14} \\ P_3: & a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + a_{23}x_3 & = & a_{24} \end{array}$$

Bagaimana cara menghilangkan x_0 dari P_3 dengan memanfaatkan P_1??

[1]Ingat! Python memulai indeks-nya dari angka 0, bukan angka 1. Sehingga elemen pertama memiliki indeks a_{00}, bukan a_{11} sebagaimana yang berlaku di Matlab

Begini caranya, $(P_3 - \lambda P_1) \rightarrow (P_3)$, dengan $\lambda = \frac{a_{20}}{a_{00}}$..

$$P_3 : \left(a_{20} - \frac{a_{20}}{a_{00}}a_{00}\right)x_0 + \left(a_{21} - \frac{a_{20}}{a_{00}}a_{01}\right)x_1 + \left(a_{22} - \frac{a_{20}}{a_{00}}a_{02}\right)x_2 + \left(a_{23} - \frac{a_{20}}{a_{00}}a_{03}\right)x_3 = \left(a_{24} - \frac{a_{20}}{a_{00}}a_{04}\right)$$

Mudah-mudahan, polanya semakin terlihat jelas. Selanjutnya jika ada persamaan P_4 yang ingin dihilangkan $x_0$nya dengan memanfaatkan P_1, bagaimana caranya?

Tentu saja operasinya adalah $(P_4 - \lambda P_1) \rightarrow (P_4)$, dengan $\lambda = \frac{a_{30}}{a_{00}}$

$$P_4 : \left(a_{30} - \frac{a_{30}}{a_{00}}a_{00}\right)x_0 + \left(a_{31} - \frac{a_{30}}{a_{00}}a_{01}\right)x_1 + \left(a_{32} - \frac{a_{30}}{a_{00}}a_{02}\right)x_2 + \left(a_{33} - \frac{a_{30}}{a_{00}}a_{03}\right)x_3 = \left(a_{34} - \frac{a_{30}}{a_{00}}a_{04}\right)$$

4.3 Triangularisasi dan substitusi mundur

4.3.1 Contoh pertama

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

$$\begin{array}{lrcrcrcrcl} P_1 : & x_0 & + & x_1 & & & + & 3x_3 & = & 4 \\ P_2 : & 2x_0 & + & x_1 & - & x_2 & + & x_3 & = & 1 \\ P_3 : & 3x_0 & - & x_1 & - & x_2 & + & 2x_3 & = & \text{-3} \\ P_4 : & -x_0 & + & 2x_1 & + & 3x_2 & - & x_3 & = & 4 \end{array}$$

Sekali lagi saya tegaskan bahwa problem dari sistem persamaan linear adalah bagaimana mendapatkan angka-angka yang bisa menggantikan variabel x_0, x_1, x_2, dan x_3 sehingga semua persamaan di atas menjadi benar. Dengan berpegang pada ketiga teknik penyederhanaan tadi, sistem persamaan linear di atas dapat disederhanakan dengan langkah-langkah berikut ini:

1. Gunakan persamaan P_1 untuk menghilangkan variabel x_0 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 $(P_4 + P_1) \rightarrow (P_4)$[2]. Hasilnya akan seperti ini

$$\begin{array}{rrcl} P_1 : & x_0 + x_1 + 3x_3 & = & 4, \\ P_2 : & -x_1 - x_2 - 5x_3 & = & -7, \\ P_3 : & -4x_1 - x_2 - 7x_3 & = & -15, \\ P_4 : & 3x_1 + 3x_2 + 2x_3 & = & 8 \end{array}$$

   Kini x_0 telah hilang dari P_2, P_3 dan P_4.

2. Berdasarkan hasil ini, gunakan persamaan P_2 untuk menghilangkan variabel x_1 dari persamaan P_3 dan P_4 dengan cara $(P_3 - 4P_2) \rightarrow (P_3)$ dan $(P_4 + 3P_2) \rightarrow (P_4)$. Maka hasilnya

[2]Tahukah anda mengapa operasi untuk P_4 berbentuk $(P_4 + P_1) \rightarrow (P_4)$?

akan seperti ini

$$\begin{aligned} P_1: \quad & x_0 + x_1 + 3x_3 &&= 4, \\ P_2: \quad & -x_1 - x_2 - 5x_3 &&= -7, \\ P_3: \quad & 3x_2 + 13x_3 &&= 13, \\ P_4: \quad & -13x_3 &&= -13 \end{aligned}$$

Seandainya x_2 masih ada di persamaan P_4, maka diperlukan satu operasi lagi untuk menghilangkannya. Namun hasil operasi pada langkah ke-2 ternyata sudah otomatis menghilangkan x_2. Bentuk akhir dari keempat persamaan 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 angka-angka pengganti variabel x_0, x_1, x_2 dan x_3 dengan cara yang lebih mudah dibandingkan sebelum disederhanakan.

3. Selanjutnya kita jalankan proses **backward-substitution**. Melalui proses ini, yang pertama kali didapat adalah nilai pengganti bagi variabel x_3, kemudian x_2, lalu diikuti x_1, dan akhirnya x_0. Oleh karena itu, saya balik urutan persamaannya. Mohon diperhatikan..

$$\begin{aligned} P_4: \quad & x_3 = \frac{-13}{-13} &&= 1, \\ P_3: \quad & x_2 = \frac{1}{3}(13 - 13x_3) &&= \frac{1}{3}(13 - 13) = 0, \\ P_2: \quad & x_1 = -(-7 + 5x_3 + x_2) &&= -(-7 + 5 + 0) = 2, \\ P_1: \quad & x_0 = 4 - 3x_3 - x_1 &&= 4 - 3 - 2 = -1 \end{aligned}$$

Jadi solusinya adalah $x_0 = -1$, $x_1 = 2$, $x_2 = 0$ dan $x_3 = 1$. Coba sekarang anda cek, apakah semua solusi ini cocok dan tepat bila dimasukan ke sistem persamaan linear yang belum disederhanakan?

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

4.3.2 Contoh kedua

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

$$\begin{array}{lrcrcrcrcr} P_1: & x_0 & - & x_1 & + & 2x_2 & - & x_3 & = & \text{-8} \\ P_2: & 2x_0 & - & 2x_1 & + & 3x_2 & - & 3x_3 & = & \text{-20} \\ P_3: & x_0 & + & x_1 & + & x_2 & & & = & \text{-2} \\ P_4: & x_0 & - & x_1 & + & 4x_2 & + & 3x_3 & = & 4 \end{array}$$

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

1. Gunakan persamaan P_1 untuk menghilangkan x_0 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

$$\begin{aligned} P_1: \quad & x_0 - x_1 + 2x_2 - x_3 = -8, \\ P_2: \quad & -x_2 - x_3 = -4, \\ P_3: \quad & 2x_1 - x_2 + x_3 = 6, \\ P_4: \quad & 2x_2 + 4x_3 = 12 \end{aligned}$$

   Perhatikan persamaan P_2! Akibat dari langkah yang pertama tadi, selain x_0, ternyata x_1 juga hilang dari persamaan P_2. Kondisi ini bisa menggagalkan proses triangularisasi. Untuk itu, posisi P_2 mesti ditukar dengan persamaan yang berada dibawahnya, yaitu P_3 atau P_4. Supaya proses triangularisasi dilanjutkan kembali, 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

$$\begin{aligned} P_1: \quad & x_0 - x_1 + 2x_2 - x_3 = -8, \\ P_2: \quad & 2x_1 - x_2 + x_3 = 6, \\ P_3: \quad & -x_2 - x_3 = -4, \\ P_4: \quad & 2x_2 + 4x_3 = 12 \end{aligned}$$

3. Gunakan persamaan P_3 untuk menghilangkan x_2 dari persamaan P_4 dengan cara $(P_4 + 2P_3) \rightarrow (P_4)$. Hasilnya akan seperti ini

$$\begin{aligned} P_1: \quad & x_0 - x_1 + 2x_2 - x_3 = -8, \\ P_2: \quad & 2x_1 - x_2 + x_3 = 6, \\ P_3: \quad & -x_2 - x_3 = -4, \\ P_4: \quad & 2x_3 = 4 \end{aligned}$$

   Sampai disini proses triangularisasi telah selesai.

4. Selanjutnya adalah proses backward-substitution. Melalui proses ini, yang pertama kali didapat solusinya adalah x_3, kemudian x_2, lalu diikuti x_1, dan akhirnya x_0.

$$\begin{aligned} P_4: \quad & x_3 = \frac{4}{2} &&= 2, \\ P_3: \quad & x_2 = \frac{-4 + x_3}{-1} &&= 2, \\ P_2: \quad & x_1 = \frac{6 + x_2 - x_3}{2} &&= 3, \\ P_1: \quad & x_0 = -8 + x_1 - 2x_2 + x_3 &&= -7 \end{aligned}$$

Jadi solusinya adalah $x_0 = -7$, $x_1 = 3$, $x_2 = 2$ dan $x_3 = 2$.

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**.

4.4 Matrik dan Eliminasi Gauss dalam Python

4.4.1 Matrik augmentasi

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

$$\begin{aligned} a_{00}x_0 + a_{01}x_1 + \ldots + a_{0n}x_n &= b_0 \\ a_{10}x_0 + a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ \ldots\ldots\ldots\ldots\ldots &= \ldots \\ \ldots\ldots\ldots\ldots\ldots &= \ldots \\ a_{n0}x_0 + a_{n1}x_1 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

Bentuk operasi matrik yang sesuai dengan sistem persamaan linear di atas adalah

$$\begin{bmatrix} a_{00} & a_{01} & \ldots & a_{0n} \\ a_{10} & a_{11} & \ldots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n0} & a_{n1} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \tag{4.2}$$

Dalam upaya mencari solusi suatu sistem persamaan linear menggunakan metode eliminasi gauss, bentuk operasi matrik di atas dimanipulasi menjadi **matrik augment**, yaitu suatu matrik

yang berukuran n x $(n+1)$ seperti berikut ini:

$$\left[\begin{array}{cccc|c} a_{00} & a_{01} & \dots & a_{0n} & b_0 \\ a_{10} & a_{11} & \dots & a_{1n} & b_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n0} & a_{n1} & \dots & a_{nn} & b_n \end{array}\right] \Longrightarrow \left[\begin{array}{cccc|c} a_{00} & a_{01} & \dots & a_{0n} & a_{0,n+1} \\ a_{10} & a_{11} & \dots & a_{1n} & a_{1,n+1} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n0} & a_{n1} & \dots & a_{nn} & a_{n,n+1} \end{array}\right] \tag{4.3}$$

4.4.2 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

$$\begin{array}{lccccccccc} P_1: & x_0 & + & x_1 & & & + & 3x_3 & = & 4 \\ P_2: & 2x_0 & + & x_1 & - & x_2 & + & x_3 & = & 1 \\ P_3: & 3x_0 & - & x_1 & - & x_2 & + & 2x_3 & = & \text{-3} \\ P_4: & -x_0 & + & 2x_1 & + & 3x_2 & - & x_3 & = & 4 \end{array}$$

Sistem persamaan linear tersebut dapat dinyatakan dalam operasi matrik

$$\left[\begin{array}{cccc} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{array}\right] \left[\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c} 4 \\ 1 \\ -3 \\ 4 \end{array}\right]$$

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

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \end{array}\right]$$

Kemudian kita lakukan operasi triangularisai terhadap matrik augment, dimulai dari kolom pertama (yang tujuannya untuk menghilangkan x_0 dari P_2, P_3, dan P_4), yaitu

$$P_2: \left(a_{10} - \frac{a_{10}}{a_{00}}a_{00}\right)x_0 + \left(a_{11} - \frac{a_{10}}{a_{00}}a_{01}\right)x_1 + \left(a_{12} - \frac{a_{10}}{a_{00}}a_{02}\right)x_2 + \left(a_{13} - \frac{a_{10}}{a_{00}}a_{03}\right)x_3 = \left(a_{14} - \frac{a_{10}}{a_{00}}a_{04}\right)$$

$$P_3: \left(a_{20} - \frac{a_{20}}{a_{00}}a_{00}\right)x_0 + \left(a_{21} - \frac{a_{20}}{a_{00}}a_{01}\right)x_1 + \left(a_{22} - \frac{a_{20}}{a_{00}}a_{02}\right)x_2 + \left(a_{23} - \frac{a_{20}}{a_{00}}a_{03}\right)x_3 = \left(a_{24} - \frac{a_{20}}{a_{00}}a_{04}\right)$$

$$P_4: \left(a_{30} - \frac{a_{30}}{a_{00}}a_{00}\right)x_0 + \left(a_{31} - \frac{a_{30}}{a_{00}}a_{01}\right)x_1 + \left(a_{32} - \frac{a_{30}}{a_{00}}a_{02}\right)x_2 + \left(a_{33} - \frac{a_{30}}{a_{00}}a_{03}\right)x_3 = \left(a_{34} - \frac{a_{30}}{a_{00}}a_{04}\right)$$

Sekarang akan saya tulis *source code* Python 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..

```
from numpy import array
A = array([[1.,1.,0.,3.,4],\
           [2.,1.,-1.,1.,1],\
           [3.,-1.,-1.,2.,-3],\
           [-1.,2.,3.,-1,4]])     # Inisialisasi matrik Augmentasi

m=A[1,0]/A[0,0]                   # huruf m mewakili simbol lambda
A[1,0]=A[1,0]-m*A[0,0]
A[1,1]=A[1,1]-m*A[0,1]
A[1,2]=A[1,2]-m*A[0,2]
A[1,3]=A[1,3]-m*A[0,3]
A[1,4]=A[1,4]-m*A[0,4]

m=A[2,0]/A[0,0]
A[2,0]=A[2,0]-m*A[0,0]
A[2,1]=A[2,1]-m*A[0,1]
A[2,2]=A[2,2]-m*A[0,2]
A[2,3]=A[2,3]-m*A[0,3]
A[2,4]=A[2,4]-m*A[0,4]

m=A[3,0]/A[0,0]
A[3,0]=A[3,0]-m*A[0,0]
A[3,1]=A[3,1]-m*A[0,1]
A[3,2]=A[3,2]-m*A[0,2]
A[3,3]=A[3,3]-m*A[0,3]
A[3,4]=A[3,4]-m*A[0,4]
```

Hasilnya akan seperti ini

$$\left[\begin{array}{cccc|c} 1 & 1 & 0 & 3 & 4 \\ 0 & -1 & -1 & -5 & -7 \\ 0 & -4 & -1 & -7 & -15 \\ 0 & 3 & 3 & 2 & 8 \end{array}\right] \Rightarrow \left[\begin{array}{cccc|c} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \end{array}\right]$$

Pada kolom pertama, seluruh elemen berubah menjadi nol ($a_{10} = 0$, $a_{20} = 0$, dan $a_{30} = 0$) kecuali elemen yang paling atas a_{00}. Itu berarti kita sudah menghilangkan x_0 dari P_2, P_3, dan P_4. Sekarang dilanjutkan ke kolom kedua, dengan operasi yang hampir sama, untuk membuat elemen a_{21} dan a_{31} bernilai nol

$$P_3 : \left(a_{20} - \frac{a_{21}}{a_{11}}a_{10}\right)x_0 + \left(a_{21} - \frac{a_{21}}{a_{11}}a_{11}\right)x_1 + \left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \left(a_{23} - \frac{a_{21}}{a_{11}}a_{13}\right)x_3 = \left(a_{24} - \frac{a_{21}}{a_{11}}a_{14}\right)$$

$$P_4 : \left(a_{30} - \frac{a_{31}}{a_{11}}a_{10}\right)x_0 + \left(a_{31} - \frac{a_{31}}{a_{11}}a_{11}\right)x_1 + \left(a_{32} - \frac{a_{31}}{a_{11}}a_{12}\right)x_2 + \left(a_{33} - \frac{a_{31}}{a_{11}}a_{13}\right)x_3 = \left(a_{34} - \frac{a_{31}}{a_{11}}a_{14}\right)$$

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

```
m=A[2,1]/A[1,1]
A[2,0]=A[2,0]-m*A[1,0]
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]
```

```
7
8  m=A[3,1]/A[1,1]
9  A[3,0]=A[3,0]-m*A[1,0]
10 A[3,1]=A[3,1]-m*A[1,1]
11 A[3,2]=A[3,2]-m*A[1,2]
12 A[3,3]=A[3,3]-m*A[1,3]
13 A[3,4]=A[3,4]-m*A[1,4]
```

Hasilnya akan seperti dibawah ini. Itu berarti kita telah menghilangkan x_1 dari P_3, dan P_4; bahkan tanpa disengaja x_2 juga hilang dari P_4. Inilah bentuk **triangular**

$$\begin{bmatrix} 1 & 1 & 0 & 3 & | & 4 \\ 0 & -1 & -1 & -5 & | & -7 \\ 0 & 0 & 3 & 13 & | & 13 \\ 0 & 0 & 0 & -13 & | & -13 \end{bmatrix} \Rightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & | & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & | & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & | & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & | & a_{34} \end{bmatrix}$$

Walaupun x_2 sudah hilang dari P_4, sebaiknya source-code penghapusan x_2 dari P_4 tetap ditambahkan pada source-code sebelumnya agar source-code tersebut menjadi lengkap.

```
1  m=A[3,2]/A[2,2]
2  A[3,0]=A[3,0]-m*A[1,0]
3  A[3,1]=A[3,1]-m*A[1,1]
4  A[3,2]=A[3,2]-m*A[1,2]
5  A[3,3]=A[3,3]-m*A[1,3]
6  A[3,4]=A[3,4]-m*A[1,4]
```

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_3,

$$x_3 = \frac{a_{34}}{a_{33}} = \frac{-13}{-13} = 1$$

ini dapat dinyatakan dalam rumus umum, yaitu

$$x_{n-1} = \frac{a_{n-1,n}}{a_{n-1,n-1}}$$

lalu dilanjutkan dengan x_2, x_1, dan x_0.

$$\begin{aligned} x_2 = \frac{a_{24} - a_{23}x_3}{a_{22}} &= \frac{13 - [(13)(1)]}{3} = 0 \\ x_1 = \frac{a_{14} - (a_{12}x_2 + a_{13}x_3)}{a_{11}} &= \frac{(-7) - [(-1)(0) + (-5)(1)]}{(-1)} = 2 \\ x_0 = \frac{a_{04} - (a_{01}x_1 + a_{02}x_2 + a_{03}x_3)}{a_{00}} &= \frac{4 - [(1)(2) + (0)(0) + (3)(1)]}{1} = -1 \end{aligned}$$

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

```
1  X = zeros((4,1))
2
3  X[3,0]=A[3,4]/A[3,3]
```

```
X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]
X[0,0]=(A[0,4]-(A[0,1]*X[1,0]+A[0,2]*X[2,0]+A[0,3]*X[3,0]))/A[0,0]
```

4.4.3 *Source-code* dasar

Proses triangularisasi dan substitusi-mundur dibakukan menjadi algoritma metode eliminasi gauss yang dapat diterapkan dalam berbagai bahasa pemrograman komputer, misalnya fortran, C, java, pascal, matlab, dan lain-lain. Berikut ini saya tampilkan *source-code* dalam bahasa Python sebagaimana langkah-langkah diatas

```
from numpy import array, zeros

#~~~~~~inisialisasi matrik augment~~~~~#
A = array([[1.,1.,0.,3.,4],\
           [2.,1.,-1.,1.,1],\
           [3.,-1.,-1.,2.,-3],\
           [-1.,2.,3.,-1,4]])
print A

#~~~~~~proses triangularisasi~~~~~~~~~~#
#-----menghilangkan x0 dari P2, P3 dan P4 -----#
m=A[1,0]/A[0,0]             # huruf m mewakili simbol lambda
A[1,0]=A[1,0]-m*A[0,0]
A[1,1]=A[1,1]-m*A[0,1]
A[1,2]=A[1,2]-m*A[0,2]
A[1,3]=A[1,3]-m*A[0,3]
A[1,4]=A[1,4]-m*A[0,4]

m=A[2,0]/A[0,0]
A[2,0]=A[2,0]-m*A[0,0]
A[2,1]=A[2,1]-m*A[0,1]
A[2,2]=A[2,2]-m*A[0,2]
A[2,3]=A[2,3]-m*A[0,3]
A[2,4]=A[2,4]-m*A[0,4]

m=A[3,0]/A[0,0]
A[3,0]=A[3,0]-m*A[0,0]
A[3,1]=A[3,1]-m*A[0,1]
A[3,2]=A[3,2]-m*A[0,2]
A[3,3]=A[3,3]-m*A[0,3]
A[3,4]=A[3,4]-m*A[0,4]

#-----menghilangkan x1 dari P3 dan P4 -----#

m=A[2,1]/A[1,1]
A[2,0]=A[2,0]-m*A[1,0]
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]

m=A[3,1]/A[1,1]
A[3,0]=A[3,0]-m*A[1,0]
A[3,1]=A[3,1]-m*A[1,1]
A[3,2]=A[3,2]-m*A[1,2]
```

```
46 A[3,3]=A[3,3]-m*A[1,3]
47 A[3,4]=A[3,4]-m*A[1,4]
48
49 #-----menghilangkan x2 dari P4 dst-----#
50
51 m=A[3,2]/A[2,2]
52 A[3,0]=A[3,0]-m*A[1,0]
53 A[3,1]=A[3,1]-m*A[1,1]
54 A[3,2]=A[3,2]-m*A[1,2]
55 A[3,3]=A[3,3]-m*A[1,3]
56 A[3,4]=A[3,4]-m*A[1,4]
57
58 print A
59
60 #~~~~~~proses substitusi-mundur~~~~~~~~#
61
62 X = zeros((4,1))
63
64 X[3,0]=A[3,4]/A[3,3]
65 X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
66 X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]
67 X[0,0]=(A[0,4]-(A[0,1]*X[1,0]+A[0,2]*X[2,0]+A[0,3]*X[3,0]))/A[0,0]
68
69 print X
```

4.4.4 Optimasi *source code* bagian triangular

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 69 baris statemen, sehingga perlu dilakukan proses optimasi untuk memperkecil jumlah baris statemen (tanpa menyalahi hasil perhitungan).

Langkah optimasi yang pertama difokuskan pada baris statemen ke 12 hingga ke 17, yaitu

```
m=A[1,0]/A[0,0]
A[1,0]=A[1,0]-m*A[0,0]
A[1,1]=A[1,1]-m*A[0,1]
A[1,2]=A[1,2]-m*A[0,2]
A[1,3]=A[1,3]-m*A[0,3]
A[1,4]=A[1,4]-m*A[0,4]
```

Bagian ini dapat dioptimasi menjadi

```
m=A[1,0]/A[0,0]
for i in range(0,5):
    A[1,i]=A[1,i]-m*A[0,i]
```

Langkah optimasi yang sama juga bisa diterapkan untuk rangkaian baris statemen dari baris ke 19 hingga 24 dan baris ke 26 hingga 31 (yang terdapat pada *source-code* dasar), sehingga masing-masing akan menjadi

```
m=A[2,0]/A[0,0]
for i in range(0,5):
    A[2,i]=A[2,i]-m*A[0,i]
```

dan

```
m=A[3,0]/A[0,0]
for i in range(0,5):
    A[3,i]=A[3,i]-m*A[0,i]
```

Ternyata, pola yang sama juga masih bisa ditemui (pada *source-code* dasar) hingga baris statemen ke 56. Dengan demikian, setidaknya, tahapan pertama ini akan menghasil *source-code* baru hasil optimasi awal yaitu

```
1  from numpy import array, zeros
2
3  #~~~~~~inisialisasi matrik augment~~~~~#
4  A = array([[1.,1.,0.,3.,4],\
5             [2.,1.,-1.,1.,1],\
6             [3.,-1.,-1.,2.,-3],\
7             [-1.,2.,3.,-1,4]])
8  print A
9
10 #~~~~~~proses triangularisasi~~~~~~~~~~#
11 #-----menghilangkan x0 dari P2, P3 dan P4 -----#
12
13 m=A[1,0]/A[0,0]
14 for i in range(0,5):
15     A[1,i]=A[1,i]-m*A[0,i]
16
17 m=A[2,0]/A[0,0]
18 for i in range(0,5):
19     A[2,i]=A[2,i]-m*A[0,i]
20
21 m=A[3,0]/A[0,0]
22 for i in range(0,5):
23     A[3,i]=A[3,i]-m*A[0,i]
24
25 #-----menghilangkan x1 dari P3 dan P4 -----#
26
27 m=A[2,1]/A[1,1]
28 for i in range(0,5):
29     A[2,i]=A[2,i]-m*A[1,i]
30
31 m=A[3,1]/A[1,1]
32 for i in range(0,5):
33     A[3,i]=A[3,i]-m*A[1,i]
34
35 #-----menghilangkan x2 dari P4 -----#
36
37 m=A[3,2]/A[2,2]
38 for i in range(0,5):
39     A[3,i]=A[3,i]-m*A[2,i]
40
41 print A
42
43 #~~~~~~proses substitusi-mundur~~~~~~~~#
44
45 X = zeros((4,1))
46
47 X[3,0]=A[3,4]/A[3,3]
```

```
48 X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
49 X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]
50 X[0,0]=(A[0,4]-(A[0,1]*X[1,0]+A[0,2]*X[2,0]+A[0,3]*X[3,0]))/A[0,0]
51
52 print X
```

Sekarang, *source-code* eliminasi gauss telah mengecil menjadi hanya 52 baris statemen saja (sebelumnya ada 69 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-13 hingga ke-39. Optimasi tahap dua dilakukan untuk menyederhanakan bagian tersebut, yaitu

```
for j in range(1,4):
    m=A[j,0]/A[0,0]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[0,i]

#-----menghilangkan x1 dari P3 dan P4 -----#
for j in range(2,4):
    m=A[j,1]/A[1,1]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[1,i]

#-----menghilangkan x2 dari P4 -----#
for j in range(3,4):
    m=A[j,2]/A[2,2]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[2,i]
```

Dengan demikian *source-code* keseluruhan menjadi

```
1  from numpy import array, zeros
2
3  #~~~~~~inisialisasi matrik augment~~~~~#
4  A = array([[1.,1.,0.,3.,4],\
5             [2.,1.,-1.,1.,1],\
6             [3.,-1.,-1.,2.,-3],\
7             [-1.,2.,3.,-1,4]])
8  print A
9
10 #~~~~~~proses triangularisasi~~~~~~~~~~#
11 #-----menghilangkan x0 dari P2, P3 dan P4 -----#
12 for j in range(1,4):
13     m=A[j,0]/A[0,0]
14     for i in range(0,5):
15         A[j,i]=A[j,i]-m*A[0,i]
16
17 #-----menghilangkan x1 dari P3 dan P4 -----#
18 for j in range(2,4):
19     m=A[j,1]/A[1,1]
20     for i in range(0,5):
21         A[j,i]=A[j,i]-m*A[1,i]
22
23 #-----menghilangkan x2 dari P4 -----#
24 for j in range(3,4):
```

```
25      m=A[j,2]/A[2,2]
26      for i in range(0,5):
27          A[j,i]=A[j,i]-m*A[2,i]
28
29  print A
30
31  #~~~~~~proses substitusi-mundur~~~~~~~~#
32
33  X = zeros((4,1))
34
35  X[3,0]=A[3,4]/A[3,3]
36  X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
37  X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]
38  X[0,0]=(A[0,4]-(A[0,1]*X[1,0]+A[0,2]*X[2,0]+A[0,3]*X[3,0]))/A[0,0]
39
40  print X
```

Hasil optimasi pada tahap kedua ini baru mampu mengecilkan *source-code* menjadi hanya 40 baris statemen saja. Sekarang perhatikan baris statemen ke-12 hingga ke-27. Saya bisa letakkan indeks k disana

```
k=0
for j in range(1,4):
    m=A[j,k]/A[k,k]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[k,i]

#-----menghilangkan x1 dari P3 dan P4 -----#
k=1
for j in range(2,4):
    m=A[j,k]/A[k,k]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[k,i]

#-----menghilangkan x2 dari P4 -----#
k=2
for j in range(3,4):
    m=A[j,k]/A[k,k]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[k,i]
```

Bagian ini bisa diotak-atik sedikit menjadi (Coba anda temukan perubahannya!)

```
k=0
for j in range(k+1,4):
    m=A[j,k]/A[k,k]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[k,i]

#-----menghilangkan x1 dari P3 dan P4 -----#
k=1
for j in range(k+1,4):
    m=A[j,k]/A[k,k]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[k,i]
```

```
#-----menghilangkan x2 dari P4 -----#
k=2
for j in range(k+1,4):
    m=A[j,k]/A[k,k]
    for i in range(0,5):
        A[j,i]=A[j,i]-m*A[k,i]
```

selanjutnya dioptimasi menjadi

```
for k in range(0,3):
    for j in range(k+1,4):
        m=A[j,k]/A[k,k]
        for i in range(0,5):
            A[j,i]=A[j,i]-m*A[k,i]
```

dan *source-code* yang tadinya ada 40 baris statemen, sekarang berubah menjadi hanya 28 baris statemen saja

```
1  from numpy import array, zeros
2
3  #~~~~~~inisialisasi matrik augment~~~~~#
4  A = array([[1.,1.,0.,3.,4],\
5             [2.,1.,-1.,1.,1],\
6             [3.,-1.,-1.,2.,-3],\
7             [-1.,2.,3.,-1,4]])
8  print A
9
10 #~~~~~~proses triangularisasi~~~~~~~~~~#
11 for k in range(0,3):
12     for j in range(k+1,4):
13         m=A[j,k]/A[k,k]
14         for i in range(0,5):
15             A[j,i]=A[j,i]-m*A[k,i]
16
17 print A
18
19 #~~~~~~proses substitusi-mundur~~~~~~~~#
20
21 X = zeros((4,1))
22
23 X[3,0]=A[3,4]/A[3,3]
24 X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
25 X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]
26 X[0,0]=(A[0,4]-(A[0,1]*X[1,0]+A[0,2]*X[2,0]+A[0,3]*X[3,0]))/A[0,0]
27
28 print X
```

Bagian proses triangularisasi yang semula terdiri atas banyak baris statemen, sekarang dapat diwakili oleh 5 baris statemen saja. Itu artinya proses optimasi telah berjalan efektif. Sekarang saya perlihatkan sedikit modifikasi (agak repot untuk menerangkannya dengan kalimat, tapi saya berharap anda dapat mengerti)

```
1  from numpy import array, zeros
2
```

```
3  #~~~~~~inisialisasi matrik augment~~~~~#
4  A = array([[1.,1.,0.,3.,4],\
5             [2.,1.,-1.,1.,1],\
6             [3.,-1.,-1.,2.,-3],\
7             [-1.,2.,3.,-1,4]])
8  print A
9  n=len(A)
10
11 #~~~~~~proses triangularisasi~~~~~~~~~~#
12 for k in range(0,n-1):
13     for j in range(k+1,n):
14         m=A[j,k]/A[k,k]
15         for i in range(0,n+1):
16             A[j,i]=A[j,i]-m*A[k,i]
17
18 print A
19
20 #~~~~~~proses substitusi-mundur~~~~~~~~#
21
22 X = zeros((4,1))
23
24 X[3,0]=A[3,4]/A[3,3]
25 X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
26 X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]
27 X[0,0]=(A[0,4]-(A[0,1]*X[1,0]+A[0,2]*X[2,0]+A[0,3]*X[3,0]))/A[0,0]
28
29 print X
```

4.4.5 Optimasi *source code* bagian substitusi mundur

Ok. Sekarang kita beralih ke bagian substitusi mundur. Optimasi yang pertama dilakukan adalah baris ke-24

```
1  from numpy import array, zeros
2
3  #~~~~~~inisialisasi matrik augment~~~~~#
4  A = array([[1.,1.,0.,3.,4],\
5             [2.,1.,-1.,1.,1],\
6             [3.,-1.,-1.,2.,-3],\
7             [-1.,2.,3.,-1,4]])
8  print A
9  n=len(A)
10
11 #~~~~~~proses triangularisasi~~~~~~~~~~#
12 for k in range(0,n-1):
13     for j in range(k+1,n):
14         m=A[j,k]/A[k,k]
15         for i in range(0,n+1):
16             A[j,i]=A[j,i]-m*A[k,i]
17
18 print A
19
20 #~~~~~~proses substitusi-mundur~~~~~~~~#
21
22 X = zeros((4,1))
23
24 X[n-1,0]=A[n-1,n]/A[n-1,n-1]
```

```
25
26 X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
27 X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]
28 X[0,0]=(A[0,4]-(A[0,1]*X[1,0]+A[0,2]*X[2,0]+A[0,3]*X[3,0]))/A[0,0]
29
30 print X
```

Selanjutnya, saya memilih mulai dari baris statemen ke-28, dimana sedikit perubahan dilakukan padanya

```
#~~~~~~proses substitusi-mundur~~~~~~~~#

X = zeros((4,1))

X[n-1,0]=A[n-1,n]/A[n-1,n-1]

X[2,0]=(A[2,4]-A[2,3]*X[3,0])/A[2,2]
X[1,0]=(A[1,4]-(A[1,2]*X[2,0]+A[1,3]*X[3,0]))/A[1,1]

S=0
for i in range(1,4):
    S=S+A[0,i]*X[i,0]
X[0,0]=(A[0,4]-S)/A[0,0]

print X
```

Kemudian diikuti oleh perubahan dengan pola yang sama yang diterapakan pada baris statemen 27 dan 26

```
#~~~~~~proses substitusi-mundur~~~~~~~~#

X = zeros((4,1))

X[n-1,0]=A[n-1,n]/A[n-1,n-1]

S=0
for i in range(3,4):
    S=S+A[2,i]*X[i,0]
X[2,0]=(A[2,4]-S)/A[2,2]

S=0
for i in range(2,4):
    S=S+A[1,i]*X[i,0]
X[1,0]=(A[1,4]-S)/A[1,1]

S=0
for i in range(1,4):
    S=S+A[0,i]*X[i,0]
X[0,0]=(A[0,4]-S)/A[0,0]
```

Sebelum dilanjut, semua angka 4 diganti oleh variabel n

```
#~~~~~~proses substitusi-mundur~~~~~~~~#

X = zeros((n,1))
```

```
X[n-1,0]=A[n-1,n]/A[n-1,n-1]

S=0
for i in range(3,n):
    S=S+A[2,i]*X[i,0]
X[2,0]=(A[2,n]-S)/A[2,2]

S=0
for i in range(2,n):
    S=S+A[1,i]*X[i,0]
X[1,0]=(A[1,n]-S)/A[1,1]

S=0
for i in range(1,n):
    S=S+A[0,i]*X[i,0]
X[0,0]=(A[0,n]-S)/A[0,0]
```

Lalu kita nyatakan indeks j seperti ini

```
#~~~~~~proses substitusi-mundur~~~~~~~~#

X = zeros((n,1))

X[n-1,0]=A[n-1,n]/A[n-1,n-1]

j=2
S=0
for i in range(j+1,n):
    S=S+A[j,i]*X[i,0]
X[j,0]=(A[j,n]-S)/A[j,j]

j=1
S=0
for i in range(j+1,n):
    S=S+A[j,i]*X[i,0]
X[j,0]=(A[j,n]-S)/A[j,j]

j=0
S=0
for i in range(j+1,n):
    S=S+A[j,i]*X[i,0]
X[j,0]=(A[j,n]-S)/A[j,j]
```

Inilah yang akhirnya dioptimasi menjadi

```
for j in range(2,-1,-1):
    S=0
    for i in range(j+1,n):
        S=S+A[j,i]*X[i,0]
    X[j,0]=(A[j,n]-S)/A[j,j]
```

lalu dinyatakan dalam n

```
for j in range(n-2,-1,-1):
    S=0
```

```
    for i in range(j+1,n):
        S=S+A[j,i]*X[i,0]
    X[j,0]=(A[j,n]-S)/A[j,j]
```

Setelah melalui proses optimasi yang cukup melelahkan, *source-code* hasil akhir optimasi adalah seperti berikut ini

```
1  from numpy import array, zeros
2
3  #~~~~~~inisialisasi matrik augment~~~~~#
4  A = array([[1.,1.,0.,3.,4],\
5             [2.,1.,-1.,1.,1],\
6             [3.,-1.,-1.,2.,-3],\
7             [-1.,2.,3.,-1,4]])
8  print A
9
10 #===== METODE ELIMINASI GAUSS ==========#
11 n=len(A)
12 #~~~~~~proses triangularisasi~~~~~~~~~~#
13 for k in range(0,n-1):
14     for i in range(k+1,n):
15         m=A[i,k]/A[k,k]
16         for j in range(0,n+1):
17             A[i,j]=A[i,j]-m*A[k,j]
18
19 #~~~~~~proses substitusi-mundur~~~~~~~~#
20 X = zeros((n,1))
21 X[n-1,0]=A[n-1,n]/A[n-1,n-1]
22 for j in range(n-2,-1,-1):
23     S=0
24     for i in range(j+1,n):
25         S=S+A[j,i]*X[i,0]
26     X[j,0]=(A[j,n]-S)/A[j,j]
27 #=====================================#
28 print X
```

Fantastis!! Sekarang jumlah statemen baris-nya hanya 28. Padahal source-code dasarnya terdiri dari 69 baris statemen.

4.4.6 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

$$\begin{array}{lccccccccc}
P_1: & x_0 & - & x_1 & + & 2x_2 & - & x_3 & = & \text{-8} \\
P_2: & 2x_0 & - & 2x_1 & + & 3x_2 & - & 3x_3 & = & \text{-20} \\
P_3: & x_0 & + & x_1 & + & x_2 & & & = & \text{-2} \\
P_4: & x_0 & - & x_1 & + & 4x_2 & + & 3x_3 & = & 4
\end{array}$$

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

4.4.7 *Pivoting*

Pada baris ke-15, yang merupakan bagian dari proses triangularisasi dalam *source code* di atas, terdapat

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

elemen $A[k,k]$ 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 $[k,k]$, maka tidak diragukan lagi bahwa ia pasti menempati posisi di elemen diagonal. 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*[3]. Penambahan proses *pivoting* kedalam *source code* eliminasi Gauss dimulai dari baris ke-15 sampai baris ke-20 berikut ini

```
1  from numpy import array, zeros
2
3  #~~~~~~inisialisasi matrik augment~~~~~#
4  A = array([[1.,-1.,2.,-1.,-8.],\
5             [2.,-2.,3.,-3.,-20],\
6             [1.,1.,1.,0.,-2],\
7             [1.,-1.,4.,3,4]])
8  print A
9
10 #===== METODE ELIMINASI GAUSS =========#
11 n=len(A)
12 #~~~~~~proses triangularisasi~~~~~~~~~~#
13 for k in range(0,n-1):
14     #-----proses pivot dari sini----------#
15     if A[k,k]==0:
16         for s in range(0,n+1):
17             v=A[k,s]
18             u=A[k+1,s]
19             A[k,s]=u
20             A[k+1,s]=v
21     #-----proses pivot sampai sini----------#
22     for i in range(k+1,n):
23         m=A[i,k]/A[k,k]
24         for j in range(0,n+1):
25             A[i,j]=A[i,j]-m*A[k,j]
26
27 #~~~~~~proses substitusi-mundur~~~~~~~~#
28 X = zeros((n,1))
29 X[n-1,0]=A[n-1,n]/A[n-1,n-1]
30 for j in range(n-2,-1,-1):
31     S=0
32     for i in range(j+1,n):
33         S=S+A[j,i]*X[i,0]
```

[3]Catatan singkat dan *source code* mengenai *pivoting* sudah diterangkan di Bab-2

```
34      X[j,0]=(A[j,n]-S)/A[j,j]
35  #=====================================#
36  print X
```

OK. Sekarang source-code diatas sudah bekerja dengan baik.

4.4.8 Kembali ke bentuk $Ax = b$

Sistem persamaan linear dalam pembahasan kita sejak awal memiliki rumusan $\mathbf{A}x = \mathbf{b}$ yang dijabarkan dalam operasi matrik

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix}$$

dimana

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \quad \text{dan} \quad \mathbf{b} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix}$$

Pada semua *source-code* yang baru saja dibuat di atas, inisialisasi selalu dimulai dari matrik augment yang menggabungkan matrik **A** dan vektor **b** seperti berikut

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

Oleh karena itu sebaiknya source-code tersebut perlu membedakan matrik **A** dan vektor **b** pada inisialisasi awalnya, baru setelah itu keduanya digabung menjadi matrik augment. Untuk maksud tersebut, modifikasi yang perlu dilakukan adalah

```
1   from numpy import array, zeros
2
3   #~~~~~~inisialisasi matrik A~~~~~#
4   A = array([[1.,1.,0.,3.],\
5              [2.,1.,-1.,1.],\
6              [3.,-1.,-1.,2.],\
7              [-1.,2.,3.,-1.]])
8   print A
9   #~~~~~~inisialisasi vektor b~~~~~#
10  b = array([[4.],\
11             [1],\
12             [-3],\
13             [4]])
14  print b
```

```
#~~~~~~menggabungkan matrik A dan vektor b kedalam matrik augment C~~~~~#
n=len(A)
C=zeros((n,n+1))
for i in range(0,n):
    for j in range(0,n):
        C[i,j]=A[i,j]
for i in range(0,len(A)):
    C[i,n]=b[i,0]
print C
```

Ya.. apa boleh buat, dalam *source code* ini saya terpaksa mengganti nama matrik augment yang semula bernama A (sebagaimana pada *source-code* yang sudah-sudah) menjadi C. Sehingga keseluruhan source code eliminasi gauss ditulis sebagai berikut

```
from numpy import array, zeros

#~~~~~~inisialisasi matrik A~~~~~#
A = array([[1.,1.,0.,3.],\
           [2.,1.,-1.,1.],\
           [3.,-1.,-1.,2.],\
           [-1.,2.,3.,-1.]])
#~~~~~~inisialisasi vektor b~~~~~#
b = array([[4.],\
           [1],\
           [-3],\
           [4]])
#~~~~~~menggabungkan matrik A dan vektor b kedalam matrik augment C~~~~~#
n=len(A)
C=zeros((n,n+1))
for i in range(0,n):
    for j in range(0,n):
        C[i,j]=A[i,j]
for i in range(0,n):
    C[i,n]=b[i,0]

#~~~~~~proses triangularisasi~~~~~~~~~~#
for k in range(0,n-1):
    #-----proses pivot dari sini----------#
    if C[k][k]==0:
        for s in range(0,n+1):
            v=C[k,s]
            u=C[k+1,s]
            C[k,s]=u
            C[k+1,s]=v
    #-----proses pivot sampai sini----------#
    for j in range(k+1,n):
        m=C[j,k]/C[k,k]
        for i in range(0,n+1):
            C[j,i]=C[j,i]-m*C[k,i]

#~~~~~~proses substitusi-mundur~~~~~~~~#
X = zeros((n,1))

X[n-1,0]=C[n-1,n]/C[n-1,n-1]
for j in range(n-2,-1,-1):
    S=0
    for i in range(j+1,n):
```

```
44          S=S+C[j,i]*X[i,0]
45      X[j,0]=(C[j,n]-S)/C[j,j]
46
47  print X
```

4.4.9 *Function* eliminasi gauss

Pendefinisian *function* eliminasi gauss merupakan langkah paling akhir dari proses pembuatan *source code* ini. Berdasarkan *source code* di atas, *function* eliminasi gauss bisa dimulai dari baris ke-14 hingga baris ke-45. Berikut ini adalah cara pendefinisiannya

```
1   def eliminasi_gauss(A,b):
2       n=len(A)
3       C=zeros((n,n+1))
4       for i in range(0,n):
5           for j in range(0,n):
6               C[i,j]=A[i,j]
7       for i in range(0,n):
8           C[i,n]=b[i,0]
9
10      for k in range(0,n-1):
11          if C[k,k]==0:
12              for s in range(0,n+1):
13                  v=C[k,s]
14                  u=C[k+1,s]
15                  C[k,s]=u
16                  C[k+1,s]=v
17          for j in range(k+1,n):
18              m=C[j,k]/C[k,k]
19              for i in range(0,n+1):
20                  C[j,i]=C[j,i]-m*C[k,i]
21      X = zeros((n,1))
22      X[n-1,0]=C[n-1,n]/C[n-1,n-1]
23      for j in range(n-2,-1,-1):
24          S=0
25          for i in range(j+1,n):
26              S=S+C[j,i]*X[i,0]
27              X[j,0]=(C[j,n]-S)/C[j,j]
28      return X
```

Dengan adanya *function eliminasi_gauss*[4], maka *source-code* untuk menyelesaikan sistem persamaan linear dengan metode eliminasi gauss dapat ditulis secara sangat sederhana. Berikut ini contohnya..

```
1   from numpy import array
2   from komputasi import eliminasi_gauss
3
4   #~~~~~~inisialisasi matrik A~~~~~#
5   A = array([[1.,1.,0.,3.],\
6              [2.,1.,-1.,1.],\
7              [3.,-1.,-1.,2.],\
8              [-1.,2.,3.,-1.]])
```

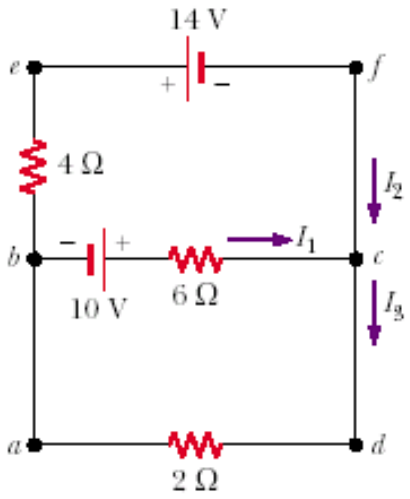
[4]Function ini saya letakkan di dalam modul komputasi. Modul ini adalah mudul *bikinan* sendiri. Saya kira anda mampu membuat modul yang sama seperti saya jika anda telah memahami bab-bab sebelumnya.

```
#~~~~~~inisialisasi vektor b~~~~~#
b = array([[4.],\
           [1],\
           [-3],\
           [4]])
#~~~~~~mencari solusi sistem persamaan linear~~~~~#
print eliminasi_gauss(A,b)
```

4.5 Contoh aplikasi

4.5.1 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:

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

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

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

Kemudian dinyatakan dalam bentuk matriks:

$$\begin{bmatrix} 1 & 1 & -1 \\ 6 & -4 & 0 \\ 6 & 0 & 2 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 24 \\ 10 \end{bmatrix} \tag{4.4}$$

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

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

Dengan demikian, 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.

Proses perhitungan di atas dilakukan oleh komputer dengan menjalankan *source-code* yang sudah kita buat. Inisialisasi matrik **A** dan vektor **b** disesuaikan dengan persamaan (4.4)

```
1  from numpy import array
2  from komputasi import eliminasi_gauss
3
4  #~~~~~~inisialisasi matrik A~~~~~#
5  A = array([[1.,1.,-1.],\
6             [6.,-4.,0.],\
7             [6.,0.,2.]])
8  #~~~~~~inisialisasi vektor b~~~~~#
9  b = array([[0.],\
10            [24.],\
11            [10]])
12 #~~~~~~mencari solusi sistem persamaan linear~~~~~#
13 print eliminasi_gauss(A,b)
```

Hasilnya adalah

```
[[ 2.]
```

```
[-3.]
[-1.]]
```

Hasil ini sama persis dengan perhitungan secara manual.

4.5.2 Menghitung invers matrik

Sekali lagi saya ulangi apa yang pernah kita bahas di awal bab ini yaitu bahwa sistem persamaan linear dapat dinyatakan sebagai berikut:

$$\begin{aligned} a_{00}x_0 + a_{01}x_1 + \ldots + a_{0n}x_n &= b_0 \\ a_{10}x_0 + a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ \ldots\ldots\ldots\ldots\ldots &= \ldots \\ \ldots\ldots\ldots\ldots\ldots &= \ldots \\ a_{n0}x_0 + a_{n1}x_1 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

Sistem persamaan linear tersebut dapat dinyatakan dalam bentuk operasi matrik,

$$\mathbf{Ax} = \mathbf{b} \tag{4.5}$$

sehingga bentuknya menjadi seperti ini:

$$\begin{bmatrix} a_{00} & a_{01} & \ldots & a_{0n} \\ a_{10} & a_{11} & \ldots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n0} & a_{n1} & \ldots & a_{nn} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

dimana

$$\mathbf{A} = \begin{bmatrix} a_{00} & a_{01} & \ldots & a_{0n} \\ a_{10} & a_{11} & \ldots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n0} & a_{n1} & \ldots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

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.

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \tag{4.6}$$

Misalnya diketahui,

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

Lalu bagaimana cara mendapatkan matrik invers, $\mathbf{A}^{-1}$? Mengacu pada persamaan (4.6)

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

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} i_{00} & i_{01} & i_{02} & i_{03} \\ i_{10} & i_{11} & i_{12} & i_{13} \\ i_{20} & i_{21} & i_{22} & i_{23} \\ i_{30} & i_{31} & i_{32} & i_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

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

$$\mathbf{A}^{-1} = \begin{bmatrix} i_{00} & i_{01} & i_{02} & i_{03} \\ i_{10} & i_{11} & i_{12} & i_{13} \\ i_{20} & i_{21} & i_{22} & i_{23} \\ i_{30} & i_{31} & i_{32} & i_{33} \end{bmatrix}$$

Elemen-elemen matrik invers, $\mathbf{A}^{-1}$ dapat diperoleh dengan memecah operasi matrik pada persamaan (4.7) menjadi 4 tahapan perhitungan. Tahapan pertama adalah

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} i_{00} \\ i_{10} \\ i_{20} \\ i_{30} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.8}$$

Dengan trik seperti ini, kita bisa memandangnya sama persis dengan contoh penyelesaian sistem persamaan linear menggunakan metode eliminasi Gauss maupun metode Gauss-Jordan.

Sudah bisa ditebak, tahapan perhitungan yang kedua sudah pasti akan seperti ini

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} i_{01} \\ i_{11} \\ i_{21} \\ i_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.9}$$

Lalu, tahapan perhitungan yang ketiga adalah

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} i_{02} \\ i_{12} \\ i_{22} \\ i_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{4.10}$$

Dan tahapan perhitungan yang keempat

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} i_{03} \\ i_{13} \\ i_{23} \\ i_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.11}$$

Source-code[5] untuk memproses seluruh tahapan di atas adalah

```
1  from numpy import array, zeros
2  from komputasi import *
3
4  #~~~~~~inisialisasi matrik~~~~~#
5  A = array([[1.,-1.,2.,-1.],\
6             [2.,-2.,3.,-3.],\
7             [1.,1.,1.,0.],\
8             [1.,-1.,4.,3.]])
9  print A
10 Ai = zeros((4,4)) #
11 i = zeros((4,1))
12
13 #-----tahap pertama-----#
14 b = array([[1.],\
15            [0.],\
16            [0.],\
17            [0.]])
18
19 i=eliminasi_gauss(A,b)
20
21 Ai[0,0]=i[0,0]
22 Ai[1,0]=i[1,0]
23 Ai[2,0]=i[2,0]
24 Ai[3,0]=i[3,0]
25
```

[5] *Source-code* di atas menggunakan modul komputasi yang didalamnya terdapat *function* gauss_jordan(A,b). Jadi, tanpa *file* modul komputasi, *source-code* tidak akan berjalan sukses. Saya kira ini bukan persoalan yang rumit bagi mereka yang telah mempelajari bab-bab sebelumnya.

```
#-----tahap kedua-----#
b = array([[0.],\
           [1.],\
           [0.],\
           [0.]])

i=eliminasi_gauss(A,b)

Ai[0,1]=i[0,0]
Ai[1,1]=i[1,0]
Ai[2,1]=i[2,0]
Ai[3,1]=i[3,0]

#-----tahap ketiga-----#
b = array([[0.],\
           [0.],\
           [1.],\
           [0.]])

i=eliminasi_gauss(A,b)

Ai[0,2]=i[0,0]
Ai[1,2]=i[1,0]
Ai[2,2]=i[2,0]
Ai[3,2]=i[3,0]

#-----tahap keempat-----#
b = array([[0.],\
           [0.],\
           [0.],\
           [1.]])

i=eliminasi_gauss(A,b)

Ai[0,3]=i[0,0]
Ai[1,3]=i[1,0]
Ai[2,3]=i[2,0]
Ai[3,3]=i[3,0]

print Ai
```

Source-code di atas menghasilkan matrik invers, $\mathbf{A}^{-1}$

```
[[-7.5  3.5  0.5  1. ]
 [ 3.  -1.5  0.5 -0.5]
 [ 4.5 -2.   0.  -0.5]
 [-2.5  1.   0.   0.5]]
```

Optimasi *source-code* perlu dilakukan agar lebih singkat

```
from numpy import array, zeros
from komputasi import *

#~~~~~~inisialisasi matrik~~~~~#
A = array([[1.,-1.,2.,-1.],\
           [2.,-2.,3.,-3.],\
```

```
7            [1.,1.,1.,0.],\
8            [1.,-1.,4.,3.]])
9  print A
10 #----proses inversi mulai dari sini----#
11 n = len(A)
12 b = zeros((n,1))
13 Ai = zeros((n,n))
14 i = zeros((n,1))
15 for j in range(0,n):
16     b[j,0]=1
17     i=eliminasi_gauss(A,b)
18     for k in range(0,n):
19         Ai[k,j]=i[k,0]
20     b[j,0]=0
21 #----proses inversi berakhir di sini----#
22 print Ai
```

Hasil perhitungannya tidak berbeda

```
[[-7.5  3.5  0.5  1. ]
 [ 3.  -1.5  0.5 -0.5]
 [ 4.5 -2.   0.  -0.5]
 [-2.5  1.   0.   0.5]]
```

Selanjutnya, mulai dari baris ke-11 sampai ke-20 dapat dinyatakan sebagai *function* yang (misalnya) diberi nama *function invers_matrik* (Jangan lupa untuk menambahkan statemen *return Ai* dibaris akhir *function* tersebut). Kemudian *function* tersebut diletakan di dalam *module komputasi*. Dengan demikian, untuk memperoleh matrik inversi, cukup dilakukan dengan source code berikut

```
1  from numpy import array
2  from komputasi import *
3
4  #~~~~~~inisialisasi matrik augment~~~~~#
5  A = array([[1.,-1.,2.,-1.],\
6             [2.,-2.,3.,-3.],\
7             [1.,1.,1.,0.],\
8             [1.,-1.,4.,3.]])
9  print A
10 print invers_matrik(A)
```

Hasilnya pun akan tetap sama

```
[[-7.5  3.5  0.5  1. ]
 [ 3.  -1.5  0.5 -0.5]
 [ 4.5 -2.   0.  -0.5]
 [-2.5  1.   0.   0.5]]
```

4.6 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 5

Aplikasi Eliminasi Gauss pada Masalah Inversi

Objektif :

- ▷ Mengenalkan teknik inversi model garis
- ▷ Mengenalkan teknik inversi model parabola
- ▷ Mengenalkan teknik inversi model bidang

Pada bab ini, saya mencoba menjelaskan contoh aplikasi Metode Eliminasi Gauss pada teknik inversi. Ada 3 contoh yang akan disinggung yaitu inversi model garis, model parabola dan model bidang. Inversi adalah suatu proses pengolahan data pengukuran lapangan yang bertujuan untuk mengestimasi parameter fisis suatu obyek. Mari kita mulai dari model garis.

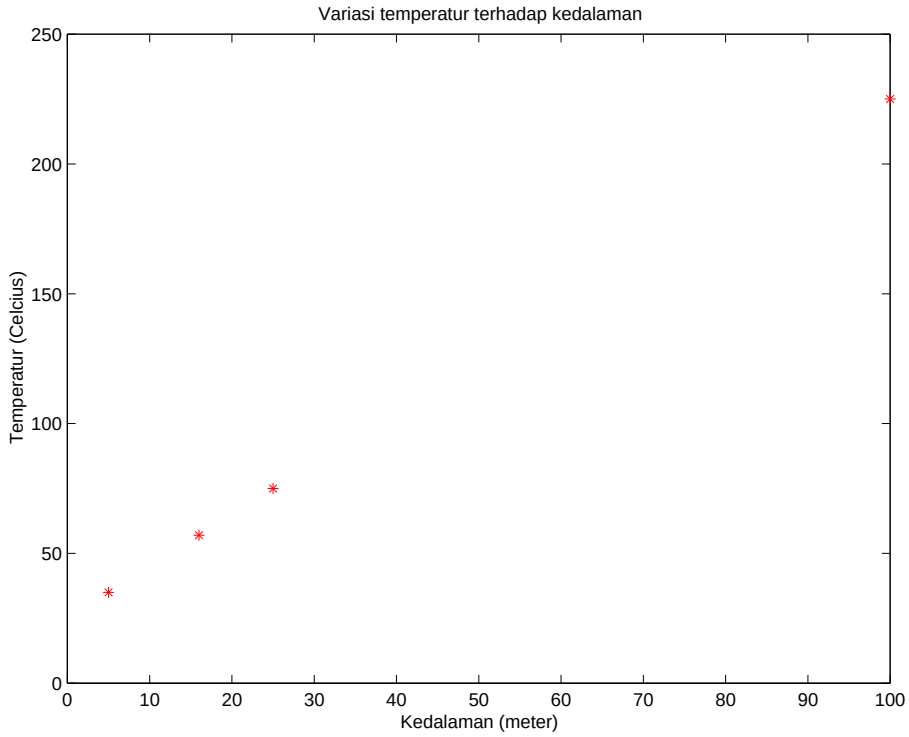
5.1 Inversi Model Garis

Pengukuran temperatur terhadap kedalaman di bawah permukaan bumi menunjukkan fakta bahwa semakin masuk kedalam perut bumi, temperatur semakin tinggi. Misalnya telah dilakukan sebanyak empat kali (N = 4) pengukuran temperatur (T_i) pada kedalaman yang berbeda beda (z_i). Data pengukuran disajikan sebagai berikut:

Tabel 5.1: Data temperatur bawah permukaan tanah terhadap kedalaman

Pengukuran ke-i	Kedalaman (m)	Temperatur $(^\circ\mathrm{C})$
1	$z_0 = 5$	$T_0 = 35$
2	$z_1 = 16$	$T_1 = 57$
3	$z_2 = 25$	$T_2 = 75$
4	$z_3 = 100$	$T_3 = 225$

Grafik sebaran data observasi ditampilkan pada Gambar (5.1). Lalu kita berasumsi bahwa variasi temperatur terhadap kedalaman ditentukan oleh suatu model matematika yang diru-



Gambar 5.1: Sebaran data observasi antara temperatur dan kedalaman

muskan berikut ini:

$$m_0 + m_1 z_i = T_i \tag{5.1}$$

dimana m_0 dan m_1 adalah konstanta-konstanta yang akan dicari. Pada rumus diatas, m_0 dan m_1 disebut **parameter model**. Jadi model matematika tersebut hanya memiliki dua buah parameter model, $(M = 2)$. Sementara jumlah data observasi ada empat, $(N = 4)$, yaitu nilai-nilai kedalaman, z_i, dan temperatur, T_i. Kasus inversi dimana $N > M$ dikenal sebagai kasus **over-determined**. Berdasarkan model tersebut, kita bisa menyatakan temperatur dan kedalaman sebagai berikut:

$$\begin{aligned} m_0 + m_1 z_0 &= T_0 \\ m_0 + m_1 z_1 &= T_1 \\ m_0 + m_1 z_2 &= T_2 \\ m_0 + m_1 z_3 &= T_3 \end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} 1 & z_0 \\ 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \end{bmatrix} = \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{5.2}$$

Lalu ditulis secara singkat

$$G\mathbf{m} = \mathbf{d} \tag{5.3}$$

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 mendapatkan nilai m_0 dan m_1 pada vektor kolom **m**? Manipulasi berikut ini bisa menjawabnya

$$G^T G\mathbf{m} = G^T\mathbf{d} \tag{5.4}$$

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

$$G = \begin{bmatrix} 1 & z_0 \\ 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \end{bmatrix} \quad \Rightarrow \quad G^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ z_0 & z_1 & z_2 & z_3 \end{bmatrix}$$

2. Tentukan $G^T G$

$$G^T G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ z_0 & z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} 1 & z_0 \\ 1 & z_1 \\ 1 & z_2 \\ 1 & z_3 \end{bmatrix} = \begin{bmatrix} N & \sum z_i \\ \sum z_i & \sum z_i^2 \end{bmatrix}$$

   dimana $N = 4$ dan $i = 0, 1, 2, 3$.

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

$$G^T\mathbf{d} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ z_0 & z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \end{bmatrix}$$

4. Sekarang persamaan (5.4) dapat dinyatakan sebagai

$$\begin{bmatrix} N & \sum z_i \\ \sum z_i & \sum z_i^2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \end{bmatrix} \tag{5.5}$$

5. Aplikasikan metode **Eliminasi Gauss**. Dimulai dari menentukan matrik augment

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

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

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

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. Hasilnya adalah

$$\left[\begin{array}{cc|c} 4 & 146 & 392 \\ 0 & 5577 & 11154 \end{array}\right] = \left[\begin{array}{cc|c} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{array}\right]$$

8. Terakhir, tentukan konstanta m_0 dan m_1 yang merupakan elemen-elemen vektor kolom **m**, dengan proses substitusi mundur. Pertama tentukan m_1

$$m_1 = \frac{a_{12}}{a_{11}} = \frac{11154}{5577} = 2$$

lalu tentukan m_0

$$m_0 = \frac{a_{02} - a_{01}m_1}{a_{00}} = \frac{392 - (146)(2)}{4} = 25$$

5.1.1 *Source code* python inversi model garis

Source-code inversi model garis ini dibangun dari *function* transpose matriks, perkalian matrik dan eliminasi gauss yang tersimpan di *module komputasi*.

```
from komputasi import *

#~~~~~~inisialisasi matrik kernel G~~~~~#
G = array([[1.,5.],\
           [1.,16.],\
           [1.,25.],\
           [1.,100.]])
print G
#~~~~~~inisialisasi vektor d~~~~~#
d = array([[35.],\
           [57],\
           [75],\
           [225]])
print d
#~~~~~~proses inversi~~~~~#
GT = transpose(G)
GTG = matxmat(GT,G)
GTd = matxvek(GT,d)
m = eliminasi_gauss(GTG,GTd)
print m
```

Sebuah *function* spesial bisa didefinisikan untuk menyatakan proses inversi

```
def inversi(G,d):
    GT = transpose(G)
```

```
    GTG = matxmat(GT,G)
    GTd = matxvek(GT,d)
    m = eliminasi_gauss(GTG,GTd)
    return m
```

Kemudian *function* ini digabungkan ke modul komputasi. Untuk memanfaatkannya bisa dengan cara berikut

```
from komputasi import *

#~~~~~~inisialisasi matrik kernel G~~~~~#
G = array([[1.,5.],\
           [1.,16.],\
           [1.,25.],\
           [1.,100.]])
print G
#~~~~~~inisialisasi vektor d~~~~~#
d = array([[35.],\
           [57],\
           [75],\
           [225]])
print d
#~~~~~~proses inversi~~~~~#
print inversi(G,d)
```

Demikianlah contoh aplikasi metode Eliminasi Gauss dengan substitusi mundur. 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**: $y = m0 + m1x$. Selanjutnya mari kita pelajari inversi model parabola.

5.2 Inversi Model Parabola

Pengukuran temperatur terhadap kedalaman di bawah permukaan bumi menunjukkan bahwa semakin dalam, temperatur semakin tinggi. Misalnya telah dilakukan sebanyak delapan kali $(N = 8)$ pengukuran temperatur (T_i) pada kedalaman yang berbeda beda (z_i). Tabel pengukuran secara sederhana disajikan seperti ini:

Tabel 5.2: Data temperatur bawah permukaan tanah terhadap kedalaman

Pengukuran ke-i	Kedalaman (m)	Temperatur (^OC)
1	$z_0 = 5$	$T_0 = 21,75$
2	$z_1 = 8$	$T_1 = 22,68$
3	$z_2 = 14$	$T_2 = 25,62$
4	$z_3 = 21$	$T_3 = 30,87$
5	$z_4 = 30$	$T_4 = 40,5$
6	$z_5 = 36$	$T_5 = 48,72$
7	$z_6 = 45$	$T_6 = 63,75$
8	$z_7 = 60$	$T_7 = 96$

Lalu kita berasumsi bahwa variasi temperatur terhadap kedalaman ditentukan oleh rumus

berikut ini:

$$m_0 + m_1 z_i + m_2 z_i^2 = T_i \tag{5.6}$$

dimana m_0, m_1 dan m_2 adalah konstanta-konstanta yang akan dicari. Rumus di atas disebut **model**. Sedangkan m_0, m_1 dan m_2 disebut **model parameter**. Jadi pada model di atas terdapat tiga buah model parameter, $(M = 3)$. Adapun yang berlaku sebagai **data** adalah nilai-nilai temperatur T_0, T_1,..., dan T_7. Berdasarkan model tersebut, kita bisa menyatakan temperatur dan kedalaman masing-masing sebagai berikut:

$$\begin{aligned}
m_0 + m_1 z_0 + m_2 z_0^2 &= T_0 \\
m_0 + m_1 z_1 + m_2 z_1^2 &= T_1 \\
m_0 + m_1 z_2 + m_2 z_2^2 &= T_2 \\
m_0 + m_1 z_3 + m_2 z_3^2 &= T_3 \\
m_0 + m_1 z_4 + m_2 z_4^2 &= T_4 \\
m_0 + m_1 z_5 + m_2 z_5^2 &= T_5 \\
m_0 + m_1 z_6 + m_2 z_6^2 &= T_6 \\
m_0 + m_1 z_7 + m_2 z_7^2 &= T_7
\end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} 1 & z_0 & z_0^2 \\ 1 & z_1 & z_1^2 \\ 1 & z_2 & z_2^2 \\ 1 & z_3 & z_3^2 \\ 1 & z_4 & z_4^2 \\ 1 & z_5 & z_5^2 \\ 1 & z_6 & z_6^2 \\ 1 & z_7 & z_7^2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \end{bmatrix} \tag{5.7}$$

Lalu ditulis secara singkat

$$G\mathbf{m} = \mathbf{d} \tag{5.8}$$

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 mendapatkan nilai m_0, m_1 dan m_2 pada vektor kolom **m**? Manipulasi berikut ini bisa menjawabnya

$$G^T G\mathbf{m} = G^T\mathbf{d} \tag{5.9}$$

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

$$G = \begin{bmatrix} 1 & z_0 & z_0^2 \\ 1 & z_1 & z_1^2 \\ 1 & z_2 & z_2^2 \\ 1 & z_3 & z_3^2 \\ 1 & z_4 & z_4^2 \\ 1 & z_5 & z_5^2 \\ 1 & z_6 & z_6^2 \\ 1 & z_7 & z_7^2 \end{bmatrix} \quad \Rightarrow \quad G^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ z_0 & z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 \\ z_0^2 & z_1^2 & z_2^2 & z_3^2 & z_4^2 & z_5^2 & z_6^2 & z_7^2 \end{bmatrix}$$

2. Tentukan $G^T G$

$$G^T G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ z_0 & z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 \\ z_0^2 & z_1^2 & z_2^2 & z_3^2 & z_4^2 & z_5^2 & z_6^2 & z_7^2 \end{bmatrix} \begin{bmatrix} 1 & z_0 & z_0^2 \\ 1 & z_1 & z_1^2 \\ 1 & z_2 & z_2^2 \\ 1 & z_3 & z_3^2 \\ 1 & z_4 & z_4^2 \\ 1 & z_5 & z_5^2 \\ 1 & z_6 & z_6^2 \\ 1 & z_7 & z_7^2 \end{bmatrix} = \begin{bmatrix} N & \sum z_i & \sum z_i^2 \\ \sum z_i & \sum z_i^2 & \sum z_i^3 \\ \sum z_i^2 & \sum z_i^3 & \sum z_i^4 \end{bmatrix}$$

dimana $N = 8$ dan $i = 0, 1, 2, ..., 7$.

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

$$G^t \mathbf{d} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ z_0 & z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 \\ z_0^2 & z_1^2 & z_2^2 & z_3^2 & z_4^2 & z_5^2 & z_6^2 & z_7^2 \end{bmatrix} \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \\ \sum z_i^2 T_i \end{bmatrix}$$

4. Sekarang persamaan (5.14) dapat dinyatakan sebagai (ini khan **least square** juga...!?)

$$\begin{bmatrix} N & \sum z_i & \sum z_i^2 \\ \sum z_i & \sum z_i^2 & \sum z_i^3 \\ \sum z_i^2 & \sum z_i^3 & \sum z_i^4 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} \sum T_i \\ \sum z_i T_i \\ \sum z_i^2 T_i \end{bmatrix} \tag{5.10}$$

5. Aplikasikan metode **Eliminasi Gauss**. Matrik augment ditentukan sebagai berikut

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

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

$$\left[\begin{array}{ccc|c} 8 & 219 & 8547 & 349,89 \\ 0 & 2551,88 & 159448,88 & 3316,57 \\ 0 & 0 & 693609,48 & 13872,19 \end{array}\right] \tag{5.11}$$

Seperti catatan yang lalu, saya ingin menyertakan pula notasi masing-masing elemen pada matrik augment sebelum melakukan proses substitusi mundur.

$$\left[\begin{array}{ccc|c} 8 & 219 & 8547 & 349,89 \\ 0 & 2551,88 & 159448,88 & 3316,57 \\ 0 & 0 & 693609,48 & 13872,19 \end{array}\right] \Leftrightarrow \left[\begin{array}{ccc|c} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{array}\right]$$

10. Terakhir, tentukan konstanta m_0, m_1 dan m_2 yang merupakan elemen-elemen vektor kolom **m**, dengan proses substitusi mundur. Pertama tentukan m_2

$$m_2 = \frac{a_{23}}{a_{22}} = \frac{13872,19}{693609,48} = 0,02$$

lalu m_1

$$m_1 = \frac{a_{13} - a_{12}m_3}{a_{11}} = \frac{3316,57 - (159448,88)(0,02)}{2551,88} = 0,05$$

dan m_0

$$m_0 = \frac{a_{03} - (a_{01}m_2 + a_{02}m_3)}{a_{00}} = \frac{349,89 - [(219)(0,05) + (8547)(0,02)}{8} = 21$$

5.2.1 *Source code* python inversi model parabola

Perbedaan utama *source code* ini dengan *source code* inversi model garis terletak pada inisialisasi elemen-elemen matrik kernel. Elemen-elemen matrik kernel sangat ditentukan oleh model matematika yang digunakan. Seperti pada *source code* ini, matrik kernelnya diturunkan dari persamaan parabola.

```
from komputasi import *

#~~~~~~inisialisasi matrik kernel G~~~~~#
G = array([[1.,5.,5.*5.],\
           [1.,8.,8.*8.],\
           [1.,14.,14.*14.],\
           [1.,21.,21.*21.],\
           [1.,30.,30.*30.],\
           [1.,36.,36.*36.],\
           [1.,45.,45.*45.],\
           [1.,60.,60.*60.]])
print G
#~~~~~~inisialisasi vektor d~~~~~#
d = array([[21.75],\
           [22.68],\
           [25.62],\
           [30.87],\
           [40.5],\
           [48.72],\
           [63.75],\
           [96]])
print d
#~~~~~~proses inversi~~~~~#
print inversi(G,d)
```

Demikianlah contoh aplikasi metode Eliminasi Gauss dengan substitusi mundur. 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: $y = m_0 + m_1x + m_2x^2$. Pada catatan berikutnya, saya akan membahas model yang mengandung tiga model parameter dalam 2 dimensi.

5.3 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:

$$m_0 + m_1 x_i + m_2 y_i = d_i \tag{5.12}$$

dimana m_0, m_1 dan m_2 merupakan model parameter yang akan dicari. Adapun yang berlaku sebagai **data** adalah $d_0, d_1, d_2, ..., d_i$. Berdasarkan model tersebut, kita bisa menyatakan temperatur dan kedalaman masing-masing sebagai berikut:

$$\begin{aligned}
m_0 + m_1 x_0 + m_2 y_0 &= d_0 \\
m_0 + m_1 x_1 + m_2 y_1 &= d_1 \\
m_0 + m_1 x_2 + m_2 y_2 &= d_2 \\
\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots& \\
m_0 + m_1 x_{N-1} + m_2 y_{N-1} &= d_{N-1}
\end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_{N-1} & y_{N-1} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix}$$

Lalu ditulis secara singkat

$$G\mathbf{m} = \mathbf{d} \tag{5.13}$$

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_0, m_1 dan m_2 pada vektor kolom **m**? Manipulasi berikut ini bisa menjawabnya

$$G^T G\mathbf{m} = G^T\mathbf{d} \tag{5.14}$$

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

$$G = \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_{N-1} & y_{N-1} \end{bmatrix} \quad \Rightarrow \quad G^t = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_{N-1} \\ y_0 & y_1 & y_2 & \cdots & y_{N-1} \end{bmatrix}$$

2. Tentukan G^TG

$$G^TG = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_{N-1} \\ y_0 & y_1 & y_2 & \cdots & y_{N-1} \end{bmatrix} \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_{N-1} & y_{N-1} \end{bmatrix} = \begin{bmatrix} N & \sum x_i & \sum y_i \\ \sum x_i & \sum x_i^2 & \sum x_i y_i \\ \sum y_i & \sum x_i y_i & \sum y_i^2 \end{bmatrix}$$

dimana N = jumlah data. dan $i = 0, 1, 2, ..., N$.

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

$$G^T\mathbf{d} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_{N-1} \\ y_0 & y_1 & y_2 & \cdots & y_{N-1} \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix} = \begin{bmatrix} \sum d_i \\ \sum x_i d_i \\ \sum y_i d_i \end{bmatrix}$$

4. Sekarang, persamaan (5.14) dapat dinyatakan sebagai

$$\begin{bmatrix} N & \sum x_i & \sum y_i \\ \sum x_i & \sum x_i^2 & \sum x_i y_i \\ \sum y_i & \sum x_i y_i & \sum y_i^2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \end{bmatrix} = \begin{bmatrix} \sum d_i \\ \sum x_i d_i \\ \sum y_i d_i \end{bmatrix} \tag{5.15}$$

5. Aplikasikan metode **Eliminasi Gauss**. 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 garis** 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): $d = m_0 + m_1 x + m_2 y$.

5.4 Contoh aplikasi

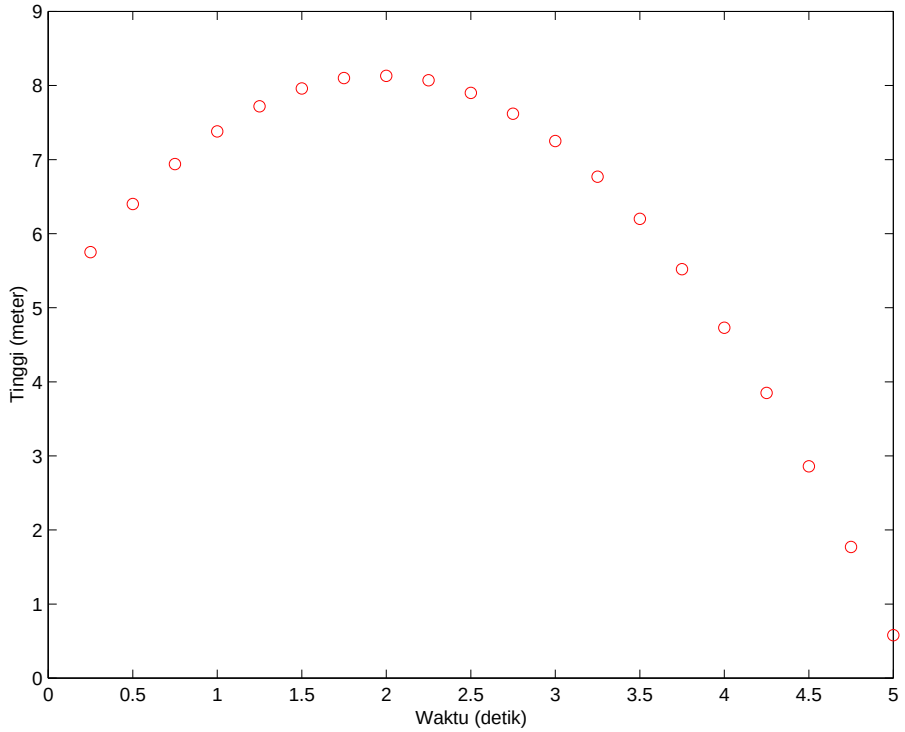
5.4.1 Menghitung gravitasi di planet X

Seorang astronot tiba di suatu planet yang tidak dikenal. Setibanya disana, ia segera mengeluarkan kamera otomatis, lalu melakukan eksperimen kinematika yaitu dengan melempar batu vertikal ke atas. Hasil foto-foto yang terekam dalam kamera otomatis adalah sebagai berikut

Tabel 5.3: 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		

Plot data pengukuran waktu vs ketinggian diperlihatkan pada Gambar 5.2. Anda diminta un-



Gambar 5.2: Grafik data pengukuran gerak batu

tuk 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 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 diformulasikan sebagai berikut

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

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

$$v_o t - \frac{1}{2} g t^2 = h - h_o \tag{5.16}$$

Selanjut, didefinisikan m_0 dan m_1 sebagai berikut

$$m_0 = v_o \qquad m_1 = -\frac{1}{2} g \tag{5.17}$$

sehingga persamaan model GLBB menjadi

$$m_0 t_i + m_1 t_i^2 = h_i - 5 \tag{5.18}$$

dimana i menunjukkan data ke-i.

Langkah berikutnya adalah menentukan nilai tiap-tiap elemen matrik kernel, yaitu dengan memasukan data observasi kedalam model matematika (persamaan (5.18))

$$\begin{aligned}
m_0 t_0 + m_1 t_0^2 &= h_0 - 5 \\
m_0 t_1 + m_1 t_1^2 &= h_1 - 5 \\
m_0 t_2 + m_1 t_2^2 &= h_2 - 5 \\
\vdots \quad \vdots \quad &= \vdots \\
m_0 t_{19} + m_1 t_{19}^2 &= h_{19} - 5
\end{aligned}$$

Semua persamaan tersebut dapat dinyatakan dalam operasi matrik berikut ini:

$$\begin{bmatrix} t_0 & t_0^2 \\ t_1 & t_1^2 \\ t_2 & t_2^2 \\ t_3 & t_3^2 \\ \vdots & \vdots \\ t_{18} & t_{18}^2 \\ t_{19} & t_{19}^2 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \end{bmatrix} = \begin{bmatrix} h_0 - 5 \\ h_1 - 5 \\ h_2 - 5 \\ \vdots \\ h_{18} - 5 \\ h_{19} - 5 \end{bmatrix}$$

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

$$G^T G\mathbf{m} = G^T\mathbf{d} \tag{5.19}$$

Selanjutnya, untuk mendapatkan m_0 dan m_1, prosedur inversi dilakukan satu-per-satu

1. Menentukan transpos matrik kernel, yaitu G^T

$$G = \begin{bmatrix} t_0 & t_0^2 \\ t_1 & t_1^2 \\ t_2 & t_2^2 \\ t_3 & t_3^2 \\ \vdots & \vdots \\ t_{18} & t_{18}^2 \\ t_{19} & t_{19}^2 \end{bmatrix} \quad \Rightarrow \quad G^T = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 & \dots & t_{18} & t_{19} \\ t_0^2 & t_1^2 & t_2^2 & t_3^2 & \dots & t_{18}^2 & t_{19}^2 \end{bmatrix}$$

2. Menentukan $G^T G$

$$G^T G = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 & \dots & t_{18} & t_{19} \\ t_0^2 & t_1^2 & t_2^2 & t_3^2 & \dots & t_{18}^2 & t_{19}^2 \end{bmatrix} \begin{bmatrix} t_0 & t_0^2 \\ t_1 & t_1^2 \\ t_2 & t_2^2 \\ t_3 & t_3^2 \\ \vdots & \vdots \\ t_{18} & t_{18}^2 \\ t_{19} & t_{19}^2 \end{bmatrix} = \begin{bmatrix} \sum t_i^2 & \sum t_i^3 \\ \sum t_i^3 & \sum t_i^4 \end{bmatrix}$$

   dimana $N = 20$ dan $i = 0, 1, 2, ..., 19$.

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

$$G^T\mathbf{d} = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 & \dots & t_{18} & t_{19} \\ t_0^2 & t_1^2 & t_2^2 & t_3^2 & \dots & t_{18}^2 & t_{19}^2 \end{bmatrix} \begin{bmatrix} h_0 - 5 \\ h_1 - 5 \\ h_2 - 5 \\ \vdots \\ h_{18} - 5 \\ h_{19} - 5 \end{bmatrix} = \begin{bmatrix} \sum t_i(h_i - 5) \\ \sum t_i^2(h_i - 5) \end{bmatrix}$$

4. Sekarang persamaan (5.19) dapat dinyatakan sebagai

$$\begin{bmatrix} \sum t_i^2 & \sum t_i^3 \\ \sum t_i^3 & \sum t_i^4 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \end{bmatrix} = \begin{bmatrix} \sum t_i(h_i - 5) \\ \sum t_i^2(h_i - 5) \end{bmatrix} \tag{5.20}$$

Berikut adalah *source code* inversi dalam python untuk memecahkan masalah ini

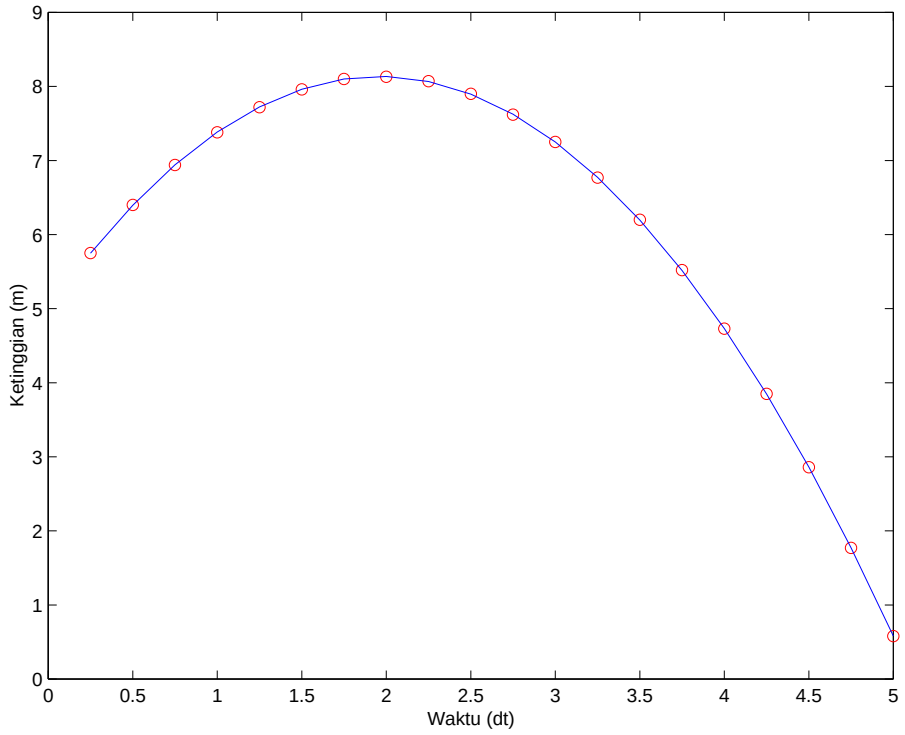
```
1 from komputasi import *
2
```

```
t = zeros((20,1))
t[0,0]=0.25
t[1,0]=0.50
t[2,0]=0.75
t[3,0]=1.00
t[4,0]=1.25
t[5,0]=1.50
t[6,0]=1.75
t[7,0]=2.00
t[8,0]=2.25
t[9,0]=2.50
t[10,0]=2.75
t[11,0]=3.00
t[12,0]=3.25
t[13,0]=3.5
t[14,0]=3.75
t[15,0]=4.
t[16,0]=4.25
t[17,0]=4.5
t[18,0]=4.75
t[19,0]=5.

h = zeros((20,1))
h[0,0]=5.75
h[1,0]=6.40
h[2,0]=6.94
h[3,0]=7.38
h[4,0]=7.72
h[5,0]=7.96
h[6,0]=8.10
h[7,0]=8.13
h[8,0]=8.07
h[9,0]=7.90
h[10,0]=7.62
h[11,0]=7.25
h[12,0]=6.77
h[13,0]=6.20
h[14,0]=5.52
h[15,0]=4.73
h[16,0]=3.85
h[17,0]=2.86
h[18,0]=1.77
h[19,0]=0.58

#~~~~~~inisialisasi matrik kernel G~~~~~#
G = zeros((20,2))
for i in range(0,20):
    G[i,0]=t[i,0]
    G[i,1]=t[i,0]*t[i,0]
print G
#~~~~~~inisialisasi vektor d~~~~~#
d = zeros((20,1))
for i in range(0,20):
    d[i,0]=h[i,0]-5
print d
#~~~~~~proses inversi~~~~~#
print inversi(G,d)
```

Hasil inversinya adalah nilai kecepatan awal yaitu saat batu dilempar ke atas adalah sebe-



Gambar 5.3: Grafik hasil inversi parabola

sar $m_0 = v_o$ = 3,2004 m/dt. Adapun percepatan gravitasi diperoleh dari m_1 dimana $m_1 = -\frac{1}{2}g$ = -0,8169; maka disimpulkan nilai g adalah sebesar 1,6338 m/dt^2.

Gambar 5.3 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.

Bab 6

Metode *LU Decomposition*

✍ **Objektif :**

▷ Mengenalkan teknik faktorisasi matrik.

▷ Mengenalkan aplikasi LU Decomposition pada sistem persamaan linear.

▷ Merumuskan algoritma LU Decomposition.

6.1 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

$$A\mathbf{x} = \mathbf{b} \tag{6.1}$$

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,

$$A = LU \tag{6.2}$$

sehingga persamaan (6.1) menjadi

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

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

$$U\mathbf{x} = \mathbf{y} \tag{6.3}$$

Langkah tersebut tidak bermaksud untuk menghitung vektor $\mathbf{y}$, melainkan untuk menghitung vektor $\mathbf{x}$. Artinya, sebelum persamaan (6.3) dieksekusi, nilai-nilai yang menempati elemen-elemen vektor $\mathbf{y}$ harus sudah diketahui. Lalu bagaimana cara memperoleh vektor $\mathbf{y}$? Begini caranya,

$$L\mathbf{y} = \mathbf{b} \tag{6.4}$$

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

$$\begin{array}{lrcrcrcrcl}
P_1: & x_0 & + & x_1 & & & + & 3x_3 & = & 4 \\
P_2: & 2x_0 & + & x_1 & - & x_2 & + & x_3 & = & 1 \\
P_3: & 3x_0 & - & x_1 & - & x_2 & + & 2x_3 & = & \text{-3} \\
P_4: & -x_0 & + & 2x_1 & + & 3x_2 & - & x_3 & = & 4
\end{array}$$

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

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix} \tag{6.5}$$

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,

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \\ 13 \\ -13 \end{bmatrix} \tag{6.6}$$

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 (6.6), 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 (6.5) menjadi,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix} \tag{6.7}$$

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

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix}$$

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

$$\begin{aligned} y_0 &= 4, \\ 2y_0 + y_1 &= 1, \\ 3y_0 + 4y_1 + y_2 &= -3, \\ -y_0 - 3y_1 + y_3 &= 4 \end{aligned}$$

maka diperoleh $y_0 = 4$, $y_1 = -7$, $y_2 = 13$, $y_3 = -13$.
Langkah terakhir adalah proses substitusi-mundur untuk menghitung vektor **x**, dimana $U\mathbf{x} = \mathbf{y}$,

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \\ 13 \\ -13 \end{bmatrix}$$

Melalui proses ini, yang pertama kali didapat solusinya adalah x_3, kemudian x_2, lalu diikuti x_1, dan akhirnya x_0.

$$\begin{aligned} x_3 &= 1 \\ x_2 &= \frac{1}{3}(13 - 13x_3) = 0 \\ x_1 &= -(-7 + 5x_3 + x_2) = 2 \\ x_0 &= 4 - 3x_3 - x_1 = -1 \end{aligned}$$

akhirnya diperoleh solusi $x_0 = -1$, $x_1 = 2$, $x_2 = 0$, dan $x_3 = 1$. Demikianlah contoh penyelesaian sistem persamaan linear dengan metode *LU-decomposition*.

6.1.1 *Source code* dalam Python

Source code metode *LU-decomposition* yang telah dioptimasi adalah sebagai berikut

```
1 from numpy import array,zeros
2
3 #~~~~~~inisialisasi matrik augment~~~~~#
4 A = array([[1.,1.,0.,3.],\
5             [2.,1.,-1.,1.],\
6             [3.,-1.,-1.,2.],\
7             [-1.,2.,3.,-1.]])
```

```
print A
b = array([[4.],\
           [1.],\
           [-3],\
           [4]])
print b

n=len(A)
#========== matrik L dari sini ====================#
L=zeros((n,n))
for i in range(0,n):
    L[i][i]=1

#~~~~~~proses triangularisasi~~~~~~#
for k in range(0,n-1):
#-----proses pivot dari sini-------#
    if A[k][k]==0:
        for s in range(0,n):
            v=A[k][s]
            u=A[k+1][s]
            A[k][s]=u
            A[k+1][s]=v
#-----proses pivot sampai sini-----#
    for j in range(k+1,n):
        m=A[j][k]/A[k][k]
        L[j][k]=m   # nilai m disimpan di matrik L
        for i in range(0,n):
            A[j][i]=A[j][i]-m*A[k][i]
#========== matrik L sampai sini ==================#

#========== matrik U dari sini ====================#
U=zeros((n,n))
for i in range(0,n):
    for j in range(0,n):
        U[i][j]=A[i][j]
#========== matrik U sampai sini ==================#

#------proses substitusi maju------#
y=zeros((n,1))
y[0][0]=b[0][0]/L[0][0]
for j in range(1,n):
    S=0
    for i in range(0,j):
        S=S+y[i][0]*L[j][i]
    y[j][0]=b[j][0]-S

#------proses substitusi mundur----#
x=zeros((n,1))
x[n-1][0]=y[n-1][0]/U[n-1][n-1]
for j in range(n-2,-1,-1):
    S=0
    for i in range(j+1,n):
        S=S+U[j][i]*x[i][0]
    x[j][0]=(y[j][0]-S)/U[j][j]

print x
```

6.2 Perubahan vektor b

Sedikit perbedaan antara metode LU-decomposition dan metode Eliminasi Gauss yang mungkin perlu diketahui adalah sekali saja matrik A berhasil difaktorkan, maka vektor **b** bisa diganti nilainya sesuai dengan sistem persamaan linear yang lain. Sebagai contoh, misalnya seluruh nilai di ruas kanan diganti menjadi

$$\begin{array}{lrcrcrcrcr} P_1: & x_0 & + & x_1 & & & + & 3x_3 & = & 8 \\ P_2: & 2x_0 & + & x_1 & - & x_2 & + & x_3 & = & 7 \\ P_3: & 3x_0 & - & x_1 & - & x_2 & + & 2x_3 & = & 14 \\ P_4: & -x_0 & + & 2x_1 & + & 3x_2 & - & x_3 & = & \text{-}7 \end{array}$$

Dalam operasi matrik menjadi

$$\begin{bmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 14 \\ -7 \end{bmatrix} \tag{6.8}$$

Perhatikan baik-baik! Matrik A sama persis dengan contoh sebelumnya. Perbedaannya hanya pada vektor **b**. Selanjutnya, dengan metode *LU-decomposition*, persamaan (6.8) menjadi

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ 14 \\ -7 \end{bmatrix} \tag{6.9}$$

Silakan anda lanjutkan proses perhitungannya dengan mencari vektor **y** sesuai contoh yang telah diberikan sebelumnya. Pada akhirnya akan diperoleh solusi sebagai berikut: $x_0 = 3$, $x_1 = -1$, $x_2 = 0$, dan $x_3 = 2$.

6.3 Penutup

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 7

Metode Iterasi

✍ **Objektif :**

▷ Mengenalkan konsep Norm.

▷ Mengenalkan iterasi Jacobi.

▷ Mengenalkan iterasi Gauss-Seidel.

▷ Mengenalkan iterasi *Succesive-Over-Relaxation (SOR)*.

7.1 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 menuliskan suatu vektor-kolom. Sebagaimana tertulis pada bab-bab sebelumnya, biasanya suatu vektor-kolom ditulis sebagai

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \tag{7.1}$$

Dengan operasi transpose, vektor-kolom tersebut dapat dinyatakan sebagai

$$\mathbf{x} = \begin{bmatrix} x_0; & x_1; & \dots & x_n \end{bmatrix}^T \tag{7.2}$$

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 (7.1), lebih sering digunakan pada penulisan operasi matrik. Dan untuk mempersingkat penulisan istilah vektor-kolom, maka saya ingin ganti saja menjadi sebuah kata yaitu vektor.

7.2 Pengertian Norm

Vektor **x**=$(x_0; x_1; ...; x_n)^T$ memiliki norm ℓ_2 dan ℓ_∞ yang didefinisikan sebagai

$$\ell_2 = \|x\|_2 = \{\sum_{i=1}^{n} x_i^2\}^{1/2} \tag{7.3}$$

dan

$$\ell_\infty = \|x\|_\infty = \max_{1 \le i \le n} |x_i| \tag{7.4}$$

Contoh: **x**=$(3; -2; 8; 5)^T$ memiliki norm ℓ_2 yaitu

$$\ell_2 = \|x\|_2 = \sqrt{(3)^2 + (-2)^2 + (8)^2 + (5)^2} = 10,0995$$

Source-code dalam bahasa Python untuk menghitung norm ℓ_2 adalah

```
from numpy import array,sqrt

#~~~~~~inisialisasi vektor x~~~~~#
x = array([[3.],\
           [-2],\
           [8],\
           [5]])
print x

n=len(x)
S=0
for i in range(0,n):
    S=S+(x[i][0])*(x[i][0])
print sqrt(S)
```

Sementara itu, formulasi norm ℓ_∞ adalah

$$\ell_\infty = \|x\|_\infty = \max\{(3), (-2), (8), (5)\} = 8$$

source-code python-nya, sangat sederhana

```
from numpy import array,sqrt

#~~~~~~inisialisasi vektor x~~~~~#
x = array([[3.],\
           [-2],\
           [8],\
           [5]])
print x

print max(abs(x))
```

Saya menyarankan agar kedua norm ini diingat-ingat dengan baik, karena akan banyak disinggung pada catatan-catatan berikutnya.

7.2.1 Perhitungan norm-selisih

Misalnya kita punya dua buah vektor, yaitu **xlama** dan **xbaru**

$$\mathbf{xlama} = \begin{bmatrix} 3 \\ -4 \\ 8 \\ 1 \end{bmatrix} \qquad \mathbf{xbaru} = \begin{bmatrix} 5 \\ 6 \\ -7 \\ 9 \end{bmatrix}$$

Norm selisih antara keduanya dapat dihitung dengan bantuan *source-code* berikut ini

```
from numpy import array,zeros,sqrt
from komputasi import norm2

#~~~~~~inisialisasi vektor x~~~~~#
xlama = array([[3.],\
           [-4.],\
           [8],\
           [1]])
print xlama
xbaru = array([[5.],\
           [6.],\
           [-7.],\
           [9]])
print xbaru

n=len(xlama)
x=zeros((n,1))
for i in range(0,n):
    x[i][0]=xlama[i][0]-xbaru[i][0]
print x

#---- hasil norm 2 ----#
print norm2(x)
#---- hasil norm tak-hingga ----#
print max(abs(x))
```

Cara perhitungan norm-selisih seperti ini akan diterapkan pada kebanyakan metode iterasi. Jadi tolong diingat baik-baik!!

7.3 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 ini. Diketahui sistem persamaan linear

$$\begin{aligned} 10x_0 - x_1 + 2x_2 &= 6 \\ -x_0 + 11x_1 - x_2 + 3x_3 &= 25 \\ 2x_0 - x_1 + 10x_2 - x_3 &= -11 \\ 3x_1 - x_2 + 8x_3 &= 15 \end{aligned}$$

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 memindahkan semua variabel x ke sebelah kiri tanda *sama-dengan* sendirian, seperti ini

$$\begin{aligned} x_0 &= \frac{1}{10}x_1 - \frac{2}{10}x_2 + \frac{6}{10} \\ x_1 &= \frac{1}{11}x_0 + \frac{1}{11}x_2 - \frac{3}{11}x_3 + \frac{25}{11} \\ x_2 &= -\frac{2}{10}x_0 + \frac{1}{10}x_1 + \frac{1}{10}x_3 - \frac{11}{10} \\ x_3 &= -\frac{3}{8}x_1 + \frac{1}{8}x_2 + \frac{15}{8} \end{aligned}$$

Kita bisa menyatakan bahwa nilai x_0, x_1, x_2 dan x_3 yang berada di ruas kiri tanda *sama-dengan* sebagai $x^{(baru)}$. Sementara nilai x_0, x_1, x_2 dan x_3 yang berada di ruas kanan tanda *sama-dengan* sebagai $x^{(lama)}$. Sehingga sistem persamaan tersebut ditulis seperti ini

$$\begin{aligned} x_0^{(baru)} &= \frac{1}{10}x_1^{(lama)} - \frac{2}{10}x_2^{(lama)} + \frac{6}{10} \\ x_1^{(baru)} &= \frac{1}{11}x_0^{(lama)} + \frac{1}{11}x_2^{(lama)} - \frac{3}{11}x_3^{(lama)} + \frac{25}{11} \\ x_2^{(baru)} &= -\frac{2}{10}x_0^{(lama)} + \frac{1}{10}x_1^{(lama)} + \frac{1}{10}x_3^{(lama)} - \frac{11}{10} \\ x_3^{(baru)} &= -\frac{3}{8}x_1^{(lama)} + \frac{1}{8}x_2^{(lama)} + \frac{15}{8} \end{aligned}$$

yang kemudian dinyatakan dalam bentuk operasi matrik

$$\begin{bmatrix} x_0^{(baru)} \\ x_1^{(baru)} \\ x_2^{(baru)} \\ x_3^{(baru)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_0^{(lama)} \\ x_1^{(lama)} \\ x_2^{(lama)} \\ x_3^{(lama)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

dan dinyatakan dalam formulasi sederhana yaitu

$$\mathbf{x}^{(baru)} = T\mathbf{x}^{(lama)} + c$$

Kalau kita ubah sedikit saja, dimana $xbaru$ dinyatakan dengan k, sementara $xlama$ dinyatakan dengan $(k-1)$ maka

$$\begin{bmatrix} x_0^{(k)} \\ x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_0^{(k-1)} \\ x_1^{(k-1)} \\ x_2^{(k-1)} \\ x_3^{(k-1)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

akhirnya kita dapatkan formulasi umum yang dirumuskan sebagai berikut

$$\mathbf{x}^k = T\mathbf{x}^{k-1} + c \qquad \text{dimana } k = 1, 2, 3, ..., n \tag{7.5}$$

Pada persamaan di atas, indeks k menunjukan sudah berapa kali perhitungan iterasi telah dilakukan. Iterasi adalah proses perhitungan yang berulang-ulang. Mari kita fokuskan sejenak pada indeks k ini. Ketika $k = 1$, maka operasi matrik di atas akan menjadi

$$\begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_0^{(0)} \\ x_1^{(0)} \\ x_2^{(0)} \\ x_3^{(0)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

Jika kita tentukan nilai-nilai awal vektor $\mathbf{x}^{(0)}$ sebagai berikut $x_0^{(0)} = 0$, $x_1^{(0)} = 0$, $x_2^{(0)} = 0$ dan $x_3^{(0)} = 0$, maka

$$\begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

Hasil perhitungan di atas adalah vektor $\mathbf{x}^{(1)}$ dengan elemen-elemen-nya adalah

$$\begin{aligned} x_0^{(1)} &= \frac{6}{10} \\ x_1^{(1)} &= \frac{25}{11} \\ x_2^{(1)} &= -\frac{11}{10} \\ x_3^{(1)} &= \frac{15}{8} \end{aligned}$$

atau $\mathbf{x}^{(1)} = (0,6000; 2,2727; -1,1000; 1,8750)^T$. Ini merupakan hasil **iterasi** pertama. Setelah

itu, proses perhitungan diulang kembali guna mendapatkan hasil iterasi kedua, dimana $k = 2$

$$\begin{bmatrix} x_0^{(2)} \\ x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

$$\begin{bmatrix} x_0^{(2)} \\ x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} 0,6000 \\ 2,2727 \\ -1,1000 \\ 1,8750 \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

maka elemen-elemen vektor $\mathbf{x}^{(2)}$ yang kita dapat adalah $\mathbf{x}^{(2)} = (1,0473; 1,7159; -0,8052; 0,8852)^T$.

Setelah itu proses perhitungan diulangi kembali guna mendapatkan hasil iterasi ketiga ($k = 3$). Caranya adalah dengan memasukan vektor $\mathbf{x}^{(2)} = (1,0473; 1,7159; -0,8052; 0,8852)^T$ ke ruas kanan,

$$\begin{bmatrix} x_0^{(3)} \\ x_1^{(3)} \\ x_2^{(3)} \\ x_3^{(3)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} x_0^{(2)} \\ x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

$$\begin{bmatrix} x_0^{(3)} \\ x_1^{(3)} \\ x_2^{(3)} \\ x_3^{(3)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{10} & -\frac{2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & -\frac{3}{11} \\ -\frac{2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & -\frac{3}{8} & \frac{1}{8} & 0 \end{bmatrix} \begin{bmatrix} 1,0473 \\ 1,7159 \\ -0,8052 \\ 0,8852 \end{bmatrix} + \begin{bmatrix} \frac{6}{10} \\ \frac{25}{11} \\ -\frac{11}{10} \\ \frac{15}{8} \end{bmatrix}$$

maka kita akan memperoleh elemen-elemen vektor $\mathbf{x}^{(3)} = (0,9326; 2,0530; -1,0493; 1,1309)^T$. 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 vektor $\mathbf{x}^{(3)}$ di atas, kita sudah melakukan tiga kali proses iterasi. Lantas sampai kapan proses iterasi ini terus berlanjut? Jawabnya adalah sampai vektor $\mathbf{x}^{(baru)}$ mendekati solusi yang sesungguhnya, 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 *source code* Python untuk metode iterasi Jacobi.

7.3.1 *Source code* Python metode iterasi Jacobi

Sebagai upaya pembelajaran, sengaja saya mulai dengan menampilkan *source code* yang paling mentah terlebih dahulu, lalu selangkah demi selangkah dimodifikasi hingga menjadi *source code* yang teroptimasi. Pertama-tama kita buat *source code* seperti ini

```
1  from numpy import array,zeros
2
3  #~~~~~~inisialisasi matrik A~~~~~#
4  A = array([[10.,-1.,2.,0.],\
5             [-1.,11.,-1.,3.],\
6             [2.,-1.,10.,-1.],\
7             [0.,3.,-1.,8.]])
8  print A
9  #~~~~~~inisialisasi vektor b~~~~~#
10 b = array([[6.],\
11            [25],\
12            [-11],\
13            [15]])
14 print b
15
16 c=zeros((4,1))
17 T=zeros((4,4))
18
19 #~~~~~~menghitung matrik T dan vektor c~~~~~#
20 T[0][1]=-1.*A[0][1]/A[0][0]
21 T[0][2]=-1.*A[0][2]/A[0][0]
22 T[0][3]=-1.*A[0][3]/A[0][0]
23 c[0][0]=b[0][0]/A[0][0]
24 T[1][0]=-1.*A[1][0]/A[1][1]
25 T[1][2]=-1.*A[1][2]/A[1][1]
26 T[1][3]=-1.*A[1][3]/A[1][1]
27 c[1][0]=b[1][0]/A[1][1]
28 T[2][0]=-1.*A[2][0]/A[2][2]
29 T[2][1]=-1.*A[2][1]/A[2][2]
30 T[2][3]=-1.*A[2][3]/A[2][2]
31 c[2][0]=b[2][0]/A[2][2]
32 T[3][0]=-1.*A[3][0]/A[3][3]
33 T[3][1]=-1.*A[3][1]/A[3][3]
34 T[3][2]=-1.*A[3][2]/A[3][3]
35 c[3][0]=b[3][0]/A[3][3]
36
37 print T
38 print c
```

Source code di atas akan mempersiapkan elemen-elemen matrik T dan vektor $\mathbf{c}$ yang dihitung berdasarkan elemen-elemen matrik A dan vektor $\mathbf{b}$. Sehingga mengubah sistem persamaan linear dalam bentuk matrik, dari

$$A\mathbf{x} = \mathbf{b} \qquad \text{menjadi} \qquad \mathbf{x}^{baru} = T\mathbf{x}^{lama} + \mathbf{c}$$

Sebelum masuk ke perhitungan iterasi, mari kita lakukan optimasi pada bagian penyusunan matrik T dan vektor $\mathbf{c}$. Kita mulai dari baris ke-20 hingga ke-23. Jika kita munculkan indeks j, maka pada bagian itu bisa dimodifikasi menjadi

```
j=0
T[j][1]=-1.*A[j][1]/A[j][j]
T[j][2]=-1.*A[j][2]/A[j][j]
T[j][3]=-1.*A[j][3]/A[j][j]
c[j][0]=b[j][0]/A[j][j]
```

Kemudian kita munculkan indeks i, dimana variasi nilai i hanya 1,2, dan 3. Dengan cara ini, beberapa baris bisa dibuang

```
j=0
for i in range (1,4):
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]
```

Namun ada sedikit masalah bila kita menengok ke baris-24 hingga baris-27 (ketika $j = 1$). Disana, variasi nilai i-nya tidak sama. Ia bergerak mulai dari 0, kemudian menjadi 2 dan akhirnya 3. Untuk mensiasatinya, variasi nilai i dibuka dari 0 sampai 3, namun perlu disisipkan perintah *if*

```
j=1
for i in range (0,4):
    if i==j:
       i=i+1
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]
```

Cara seperti ini berlaku untuk semua nilai j sehingga source code lengkapnya akan menjadi

```
1  from numpy import array,zeros
2
3  #~~~~~~inisialisasi matrik A~~~~~#
4  A = array([[10.,-1.,2.,0.],\
5             [-1.,11.,-1.,3.],\
6             [2.,-1.,10.,-1.],\
7             [0.,3.,-1.,8.]])
8  print A
9  #~~~~~~inisialisasi vektor b~~~~~#
10 b = array([[6.],\
11            [25],\
12            [-11],\
13            [15]])
14 print b
15
16 c=zeros((4,1))
17 T=zeros((4,4))
18
19 #~~~~~~menghitung matrik T dan vektor c~~~~~#
20 j=0
21 for i in range(0,4):
22     if i==j:
23         i=i+1
```

```
24      T[j][i]=-1.*A[j][i]/A[j][j]
25  c[j][0]=b[j][0]/A[j][j]
26
27  j=1
28  for i in range(0,4):
29      if i==j:
30          i=i+1
31      T[j][i]=-1.*A[j][i]/A[j][j]
32  c[j][0]=b[j][0]/A[j][j]
33
34  j=2
35  for i in range(0,4):
36      if i==j:
37          i=i+1
38      T[j][i]=-1.*A[j][i]/A[j][j]
39  c[j][0]=b[j][0]/A[j][j]
40
41  j=3
42  for i in range(0,3):
43      T[j][i]=-1.*A[j][i]/A[j][j]
44  c[j][0]=b[j][0]/A[j][j]
45
46  print T
47  print c
```

Harap diperhatikan! Ketika j masuk ke angka 3, perintah if dihapuskan karena memang perhitungannya sudah cukup sampai di situ. Sampai disini, mulai dari baris ke-20 sampai baris ke 29, bisa dioptimasi menjadi

```
1   from numpy import array,zeros
2
3   #~~~~~~inisialisasi matrik A~~~~~#
4   A = array([[10.,-1.,2.,0.],\
5              [-1.,11.,-1.,3.],\
6              [2.,-1.,10.,-1.],\
7              [0.,3.,-1.,8.]])
8   print A
9   #~~~~~~inisialisasi vektor b~~~~~#
10  b = array([[6.],\
11             [25],\
12             [-11],\
13             [15]])
14  print b
15
16  c=zeros((4,1))
17  T=zeros((4,4))
18
19  #~~~~~~menghitung matrik T dan vektor c~~~~~#
20  for j in range(0,3):
21      for i in range(0,4):
22          if i==j:
23              i=i+1
24          T[j][i]=-1.*A[j][i]/A[j][j]
25      c[j][0]=b[j][0]/A[j][j]
26
27  j=3
28  for i in range(0,3):
```

```
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]

print T
print c
```

Selanjutnya dengan menambahkan variabel n untuk menghitung $len(A)$, maka perbaikan optimasi bisa dilakukan

```
1  from numpy import array,zeros
2  
3  #~~~~~~inisialisasi matrik A~~~~~#
4  A = array([[10.,-1.,2.,0.],\
5             [-1.,11.,-1.,3.],\
6             [2.,-1.,10.,-1.],\
7             [0.,3.,-1.,8.]])
8  print A
9  #~~~~~~inisialisasi vektor b~~~~~#
10 b = array([[6.],\
11            [25],\
12            [-11],\
13            [15]])
14 print b
15 
16 #~~~~~~menghitung matrik T dan vektor c~~~~~#
17 n=len(A)
18 c=zeros((n,1))
19 T=zeros((n,n))
20 for j in range(0,n-1):
21     for i in range(0,n):
22         if i==j:
23             i=i+1
24         T[j][i]=-1.*A[j][i]/A[j][j]
25     c[j][0]=b[j][0]/A[j][j]
26 j=n-1
27 for i in range(0,n-1):
28     T[j][i]=-1.*A[j][i]/A[j][j]
29 c[j][0]=b[j][0]/A[j][j]
30 #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
31 
32 print T
33 print c
```

Demikianlah langkah-langkah optimasi sehingga kita memperoleh source-code yang general untuk menyusun elemen-elemen matrik T dan vektor c, yaitu mulai dari baris ke-17 hingga baris ke-29.

Selanjutnya, kita fokuskan kepada proses iterasi. Hasil iterasi pertama diperoleh setelah source-code di atas ditambah dengan perintah untuk menghitung $\mathbf{x}^{baru}$

```
from numpy import array,zeros
from komputasi import *

#~~~~~~inisialisasi matrik A~~~~~#
A = array([[10.,-1.,2.,0.],\
           [-1.,11.,-1.,3.],\
```

```
7             [2.,-1.,10.,-1.],\
8             [0.,3.,-1.,8.]])
9   print A
10  #~~~~~~inisialisasi vektor b~~~~~#
11  b = array([[6.],\
12            [25],\
13            [-11],\
14            [15]])
15  print b
16
17  #~~~~~~menghitung matrik T dan vektor c~~~~~#
18  n=len(A)
19  c=zeros((n,1))
20  T=zeros((n,n))
21  for j in range(0,n-1):
22      for i in range(0,n):
23          if i==j:
24              i=i+1
25          T[j][i]=-1.*A[j][i]/A[j][j]
26      c[j][0]=b[j][0]/A[j][j]
27  j=n-1
28  for i in range(0,n-1):
29      T[j][i]=-1.*A[j][i]/A[j][j]
30  c[j][0]=b[j][0]/A[j][j]
31  #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
32
33  print T
34  print c
35
36  xlama=zeros((n,1))
37  xbaru=zeros((n,1))
38
39  #~~~~~~menghitung xbaru~~~~~#
40  xbaru=matxvek(T,xlama)+c
41
42  print xbaru
```

Sampai disini, **xbaru** yang didapat adalah hasil iterasi pertama, yaitu $\mathbf{x}^{(1)} = (0,6000; 2,2727; -1,1000; 1,8750)^T$. Kemudian, untuk melangkah ke iterasi ke-2, **xbaru** yang telah didapat mesti di*copy*[1] dahulu ke **xlama**.

```
1   from numpy import array,zeros
2   from komputasi import *
3
4   #~~~~~~inisialisasi matrik A~~~~~#
5   A = array([[10.,-1.,2.,0.],\
6             [-1.,11.,-1.,3.],\
7             [2.,-1.,10.,-1.],\
8             [0.,3.,-1.,8.]])
9   print A
10  #~~~~~~inisialisasi vektor b~~~~~#
11  b = array([[6.],\
12            [25],\
13            [-11],\
```

[1]Perhatian!!! Cara meng*copy* matrik atau vektor di Python berbeda sekali dengan di bahasa C, bahasa Fortran, maupun Matlab. Uraian tentang cara mengcopy di Python telah dijelaskan pada Bab 1

```
           [15]])
print b

#~~~~~~menghitung matrik T dan vektor c~~~~~#
n=len(A)
c=zeros((n,1))
T=zeros((n,n))
for j in range(0,n-1):
    for i in range(0,n):
        if i==j:
            i=i+1
        T[j][i]=-1.*A[j][i]/A[j][j]
    c[j][0]=b[j][0]/A[j][j]
j=n-1
for i in range(0,n-1):
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

print T
print c

xlama=zeros((n,1))
xbaru=zeros((n,1))

#~~~~~~menghitung xbaru~~~~~#
xbaru=matxvek(T,xlama)+c
print xbaru

xlama=xbaru.copy()
xbaru=matxvek(T,xlama)+c
print xbaru
```

Demikian hal nya untuk melangkah ke iterasi ke-3 dan ke-4

```
from numpy import array,zeros
from komputasi import *

#~~~~~~inisialisasi matrik A~~~~~#
A = array([[10.,-1.,2.,0.],\
           [-1.,11.,-1.,3.],\
           [2.,-1.,10.,-1.],\
           [0.,3.,-1.,8.]])
print A
#~~~~~~inisialisasi vektor b~~~~~#
b = array([[6.],\
           [25],\
           [-11],\
           [15]])
print b

#~~~~~~menghitung matrik T dan vektor c~~~~~#
n=len(A)
c=zeros((n,1))
T=zeros((n,n))
for j in range(0,n-1):
    for i in range(0,n):
        if i==j:
```

```
24            i=i+1
25        T[j][i]=-1.*A[j][i]/A[j][j]
26    c[j][0]=b[j][0]/A[j][j]
27 j=n-1
28 for i in range(0,n-1):
29     T[j][i]=-1.*A[j][i]/A[j][j]
30 c[j][0]=b[j][0]/A[j][j]
31 #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
32
33 print T
34 print c
35
36 xlama=zeros((n,1))
37 xbaru=zeros((n,1))
38
39 #~~~~~~menghitung xbaru~~~~~#
40 xbaru=matxvek(T,xlama)+c
41 print xbaru
42
43 xlama=xbaru.copy()
44 xbaru=matxvek(T,xlama)+c
45 print xbaru
46
47 xlama=xbaru.copy()
48 xbaru=matxvek(T,xlama)+c
49 print xbaru
50
51 xlama=xbaru.copy()
52 xbaru=matxvek(T,xlama)+c
53 print xbaru
```

Dari baris ke-40 sampai ke-53 dapat dioptimasi menjadi

```
xlama=zeros((n,1))
xbaru=zeros((n,1))
iterasi=4
for i in range(1,iterasi+1):
    xlama=xbaru.copy()
    xbaru=matxvek(T,xlama)+c
    print xbaru
```

Agar proses perhitungan berulang sebanyak 10 kali, anda cukup mengganti jumlah iterasinya menjadi $iterasi = 10$.

```
xlama=zeros((n,1))
xbaru=zeros((n,1))
iterasi=10
for i in range(1,iterasi+1):
    xlama=xbaru.copy()
    xbaru=matxvek(T,xlama)+c
    print xbaru
```

Sampai disini kita sudah mendapati kemajuan yang cukup berarti, dimana source-code lengkap kita sekarang berbentuk

```
from numpy import array,zeros
from komputasi import *

#~~~~~~inisialisasi matrik A~~~~~#
A = array([[10.,-1.,2.,0.],\
           [-1.,11.,-1.,3.],\
           [2.,-1.,10.,-1.],\
           [0.,3.,-1.,8.]])
print A
#~~~~~~inisialisasi vektor b~~~~~#
b = array([[6.],\
           [25],\
           [-11],\
           [15]])
print b

#~~~~~~menghitung matrik T dan vektor c~~~~~#
n=len(A)
c=zeros((n,1))
T=zeros((n,n))
for j in range(0,n-1):
    for i in range(0,n):
        if i==j:
            i=i+1
        T[j][i]=-1.*A[j][i]/A[j][j]
    c[j][0]=b[j][0]/A[j][j]
j=n-1
for i in range(0,n-1):
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

xlama=zeros((n,1))
xbaru=zeros((n,1))
iterasi=10
for i in range(1,iterasi+1):
    xlama=xbaru.copy()
    xbaru=matxvek(T,xlama)+c
    print xbaru
```

Hasil dari keseluruhan iterasi dari iterasi ke-1 hingga iterasi ke-10 disajikan pada Tabel 7.1.

Tabel 7.1: 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

Kita bisa saksikan 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.

Sekarang mari kita hitung norm-selisih dari masing-masing hasil iterasi secara berurutan. Dimulai dari mencari norm-selisih antara hasil iterasi ke-1 dan ke-2. Lalu dilanjutkan dengan hasil iterasi ke-2 dan ke-3, begitu seterusnya hingga antara hasil iterasi yang ke-9 dan ke-10. Dalam prakteknya, kita cukup menambahkan source code norm-selisih

```
from numpy import array,zeros
from komputasi import *

#~~~~~~inisialisasi matrik A~~~~~#
A = array([[10.,-1.,2.,0.],\
           [-1.,11.,-1.,3.],\
           [2.,-1.,10.,-1.],\
           [0.,3.,-1.,8.]])
print A
#~~~~~~inisialisasi vektor b~~~~~#
b = array([[6.],\
           [25],\
           [-11],\
           [15]])
print b

#~~~~~~menghitung matrik T dan vektor c~~~~~#
n=len(A)
c=zeros((n,1))
T=zeros((n,n))
for j in range(0,n-1):
    for i in range(0,n):
        if i==j:
            i=i+1
        T[j][i]=-1.*A[j][i]/A[j][j]
    c[j][0]=b[j][0]/A[j][j]
j=n-1
for i in range(0,n-1):
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

xlama=zeros((n,1))
xbaru=zeros((n,1))
iterasi=10
for i in range(1,iterasi+1):
    xbaru=matxvek(T,xlama)+c
    normselisih=norm2(xbaru-xlama)
    print normselisih
    xlama=xbaru.copy()
    print xbaru
```

Tabel dibawah ini memperlihatkan hasil norm-selisih hingga iterasi ke-10. Hasil perhitungan norm-selisih tersebut, saya beri nama epsilon, ϵ, dimana semakin kecil nilai epsilon, ϵ, menandakan hasil iterasinya semakin dekat dengan solusi. Hasil norm-selisih yang semakin kecil pada iterasi ke-10 menunjukan bahwa hasil iterasi ke-10 adalah hasil yang paling dekat dengan solusi yang sebenarnya.

Tabel 7.2: Hasil perhitungan norm-selisih (dengan ℓ_2) 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	1,2557	0,4967	0,2189	...	0,0012

Kembali ke pertanyaan penting yang tadi yaitu kriteria apa yang digunakan sehingga suatu hasil iterasi bisa dikatakan paling dekat dengan solusi yang sebenarnya? Jawabnya: tergantung besar kecilnya nilai ϵ. Artinya kalau nilai ϵ ditentukan sebesar 0,2 , maka iterasi akan berhenti pada iterasi ke-4. Atau kalau nilai ϵ ditentukan sebesar 0,001 , maka proses iterasi akan berhenti pada iterasi ke-10. Kesimpulannya, semakin kecil nilai ϵ, semakin panjang proses iterasinya, namun hasil akhirnya semakin akurat. Jadi nilai ϵ berperan penting untuk menghentikan proses iterasi. Dalam hal ini, ϵ lebih umum dikenal dengan istilah *stopping-criteria*. Modifikasi source code yang menyertakan batas toleransi epsilon adalah

```
from numpy import array,zeros
from komputasi import *

#~~~~~~inisialisasi matrik A~~~~~#
A = array([[10.,-1.,2.,0.],\
           [-1.,11.,-1.,3.],\
           [2.,-1.,10.,-1.],\
           [0.,3.,-1.,8.]])
print A
#~~~~~~inisialisasi vektor b~~~~~#
b = array([[6.],\
           [25],\
           [-11],\
           [15]])
print b

#~~~~~~menghitung matrik T dan vektor c~~~~~#
n=len(A)
c=zeros((n,1))
T=zeros((n,n))
for j in range(0,n-1):
    for i in range(0,n):
        if i==j:
            i=i+1
        T[j][i]=-1.*A[j][i]/A[j][j]
    c[j][0]=b[j][0]/A[j][j]
j=n-1
for i in range(0,n-1):
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#

xlama=zeros((n,1))
xbaru=zeros((n,1))
iterasi=1000
epsilon=0.001
for i in range(1,iterasi+1):
    xbaru=matxvek(T,xlama)+c
    normselisih=norm2(xbaru-xlama)
    if normselisih<epsilon:
        break
```

```
42      xlama=xbaru.copy()
43      print xbaru
```

OK. Lengkap sudah! Metode yang baru saja kita bahas ini disebut metode **Iterasi Jacobi**.

7.4 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

$$\begin{aligned}
x_0^{(baru)} &= \frac{1}{10}x_1^{(lama)} - \frac{2}{10}x_2^{(lama)} + \frac{6}{10} \\
x_1^{(baru)} &= \frac{1}{11}x_0^{(baru)} + \frac{1}{11}x_2^{(lama)} - \frac{3}{11}x_3^{(lama)} + \frac{25}{11} \\
x_2^{(baru)} &= -\frac{2}{10}x_0^{(baru)} + \frac{1}{10}x_1^{(baru)} + \frac{1}{10}x_3^{(lama)} - \frac{11}{10} \\
x_3^{(baru)} &= -\frac{3}{8}x_1^{(baru)} + \frac{1}{8}x_2^{(baru)} + \frac{15}{8}
\end{aligned}$$

Pada baris pertama, x_0^{baru} dihitung berdasarkan x_1^{lama} dan x_2^{lama}. Kemudian x_0^{baru} tersebut langsung dipakai pada baris kedua untuk menghitung x_1^{baru}. Selanjutnya x_0^{baru} dan x_1^{baru} digunakan pada baris ketiga untuk mendapatkan x_2^{baru}. Begitu seterusnya hingga x_3^{baru} pun diperoleh pada baris keempat. Sistem persamaan tersebut dapat dinyatakan dalam indeks *k* seperti dibawah ini dimana *k* adalah jumlah iterasi.

$$\begin{aligned}
x_0^{(k)} &= \frac{1}{10}x_1^{(k-1)} - \frac{2}{10}x_2^{(k-1)} + \frac{6}{10} \\
x_1^{(k)} &= \frac{1}{11}x_0^{(k)} + \frac{1}{11}x_2^{(k-1)} - \frac{3}{11}x_3^{(k-1)} + \frac{25}{11} \\
x_2^{(k)} &= -\frac{2}{10}x_0^{(k)} + \frac{1}{10}x_1^{(k)} + \frac{1}{10}x_3^{(k-1)} - \frac{11}{10} \\
x_3^{(k)} &= -\frac{3}{8}x_1^{(k)} + \frac{1}{8}x_2^{(k)} + \frac{15}{8}
\end{aligned}$$

Misalnya kita tentukan nilai-nilai awal $\mathbf{x}^{(0)}$ sebagai berikut $x_0^{(0)} = 0$, $x_1^{(0)} = 0$, $x_2^{(0)} = 0$ dan $x_3^{(0)} = 0$. 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

$$\begin{aligned}
x_0^{(1)} &= 0,6000 \\
x_1^{(1)} &= 2,3272 \\
x_2^{(1)} &= -0,9873 \\
x_3^{(1)} &= 0,8789
\end{aligned}$$

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 7.3: 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.

7.4.1 *Source code* iterasi Gauss-Seidel

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.

```
from numpy import array,zeros
from komputasi import *

#~~~~~~inisialisasi matrik A~~~~~#
A = array([[10.,-1.,2.,0.],\
           [-1.,11.,-1.,3.],\
           [2.,-1.,10.,-1.],\
           [0.,3.,-1.,8.]])
print A
#~~~~~~inisialisasi vektor b~~~~~#
b = array([[6.],\
           [25],\
           [-11],\
           [15]])
print b

n=len(A)
iterasi=50000       #jumlah maksimum iterasi
toleransi=0.0001    #batas toleransi
xlama=zeros((n,1))
xbaru=zeros((n,1))
c=zeros((n,1))
T=zeros((n,n))

#===== Menghitung matrik T dan vektor c =====
for j in range(0,n-1):
    for i in range(0,n):
```

```
        if i==j:
            i=i+1
        T[j][i]=-1.*A[j][i]/A[j][j]
    c[j][0]=b[j][0]/A[j][j]
j=n-1
for i in range(0,n-1):
    T[j][i]=-1.*A[j][i]/A[j][j]
c[j][0]=b[j][0]/A[j][j]

#===== Metode Gauss-Seidel ======================
for m in range(1,iterasi):
    S=0
    for i in range(0,n):
        S=S+T[0][i]*xlama[i][0]
    xbaru[0][0]=S+c[0][0]
    for k in range(1,n):
        P=0
        for j in range(0,k):
            P=P+T[k][j]*xbaru[j][0]
        S=0
        for i in range(k,n):
            S=S+T[k][i]*xlama[i][0]
        xbaru[k][0]=P+S+c[k][0]
    x=xbaru-xlama
    normselisih=norm2(x)
    if normselisih<toleransi:
        break
    xlama=xbaru.copy()

#===== Mencetak hasil perhitungan ===============
print 'iterasi ke', m
print xbaru
```

Perumusan metode Iterasi Gauss-Seidel dapat dinyatakan sebagai berikut:

$$x_i^{(k)} = \frac{-\sum_{j=1}^{i-1}\left(a_{ij}x_j^{(k)}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j^{(k-1)}\right) + b_i}{a_{ii}} \tag{7.6}$$

dimana i=1,2,3,...,n.

7.5 Iterasi dengan Relaksasi

Metode Iterasi Relaksasi (*Relaxation method*) dinyatakan dengan rumus berikut:

$$x_i^{(k)} = (1-\omega)\, x_i^{(k-1)} + \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right] \tag{7.7}$$

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 7.4: 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 7.5: 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.

7.5.1 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 8

Interpolasi

Objektif :

- Mengenalkan Interpolasi Lagrange
- Mengenalkan Interpolasi Spline-cubic

8.1 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

$$P(x) = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 \tag{8.1}$$

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 $x = x_2$

$$P(x_2) = \frac{(x_2 - x_1)(x_2 - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x_2 - x_0)(x_2 - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x_2 - x_0)(x_2 - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2 = y_2$$

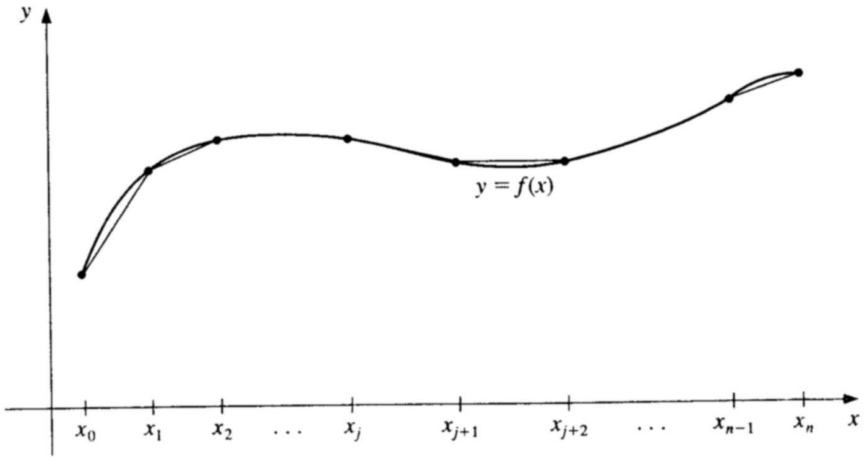
Terbukti bahwa fungsi polonomial

$$P(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2 \qquad (8.2)$$

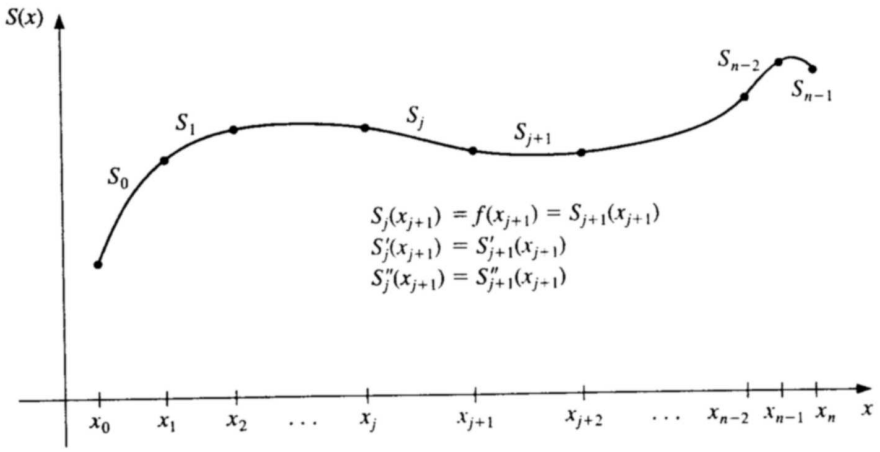
melewati ketiga titik tadi.

Kalau kita bandingkan antara persamaan (8.1) dan persamaan (8.2), terlihat bahwa derajat persamaan (8.2) lebih tinggi dibandingkan dengan derajat persamaan (8.1). Hal ini terlihat dari x^2 pada persamaan (8.2) sementara pada persamaan (8.1) hanya ada x. persamaan (8.2) disebut funsi polinomial berderajat 2, sedangkan persamaan (8.1) disebut fungsi polinomial berderajat 1.

8.2 Interpolasi Cubic Spline



Gambar 8.1: Fungsi $f(x)$ dengan sejumlah titik data



Gambar 8.2: Pendekatan dengan polinomial cubic spline

Diketahui suatu fungsi $f(x)$ (Figure 8.1) 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 8.2) 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 8.2. 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

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \tag{8.3}$$

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 (13.7)

$$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3$$

dimana $j = 0, 1, ..., n-2$. Sekarang, kita nyatakan $h_j = x_{j+1} - x_j$, sehingga

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \tag{8.4}$$

Kemudian, turunan pertama dari persamaan (13.7) 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

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2 \tag{8.5}$$

Berikutnya, kita hitung turunan kedua dari persamaan (13.7)

$$S''_j(x) = 2c_j + 6d_j(x - x_j) \tag{8.6}$$

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 $x = x_{j+1}$

$$\begin{aligned} c_{j+1} = S''_j(x_{j+1}) &= c_j + 3d_j(x_{j+1} - x_j) \\ c_{j+1} &= c_j + 3d_j h_j \end{aligned} \tag{8.7}$$

dan d_j bisa dinyatakan

$$d_j = \frac{1}{3h_j}(c_{j+1} - c_j)$$

dari sini, persamaan (8.4) dapat ditulis kembali

$$\begin{aligned} a_{j+1} &= a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \\ &= a_j + b_j h_j + c_j h_j^2 + \frac{h_j^2}{3}(c_{j+1} - c_j) \\ &= a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1}) \end{aligned} \tag{8.8}$$

sementara persamaan (8.5) menjadi

$$\begin{aligned} b_{j+1} &= b_j + 2c_j h_j + 3d_j h_j^2 \\ &= b_j + 2c_j h_j + h_j(c_{j+1} - c_j) \\ &= b_j + h_j(c_j + c_{j+1}) \end{aligned} \tag{8.9}$$

Sampai sini masih bisa diikuti, bukan? Selanjutnya, kita coba mendapatkan b_j dari persamaan (8.8)

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}) \tag{8.10}$$

dan untuk b_{j-1}

$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j) \tag{8.11}$$

Langkah berikutnya adalah mensubtitusikan persamaan (8.10) dan persamaan (8.11) kedalam persamaan (8.9),

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \tag{8.12}$$

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 (8.6)

$$S''(x_0) = 2c_0 + 6d_0(x_0 - x_0) = 0$$

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 (8.12) 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 (8.10) 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,

$$2h_0c_0 + h_0c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \tag{8.13}$$

Sementara pada $x_n = b_n$ dengan persamaan (8.9)

$$f'(b) = b_n = b_{n-1} + h_{n-1}(c_{n-1} + c_n)$$

sedangkan b_{n-1} bisa didapat dari persamaan (8.11) 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

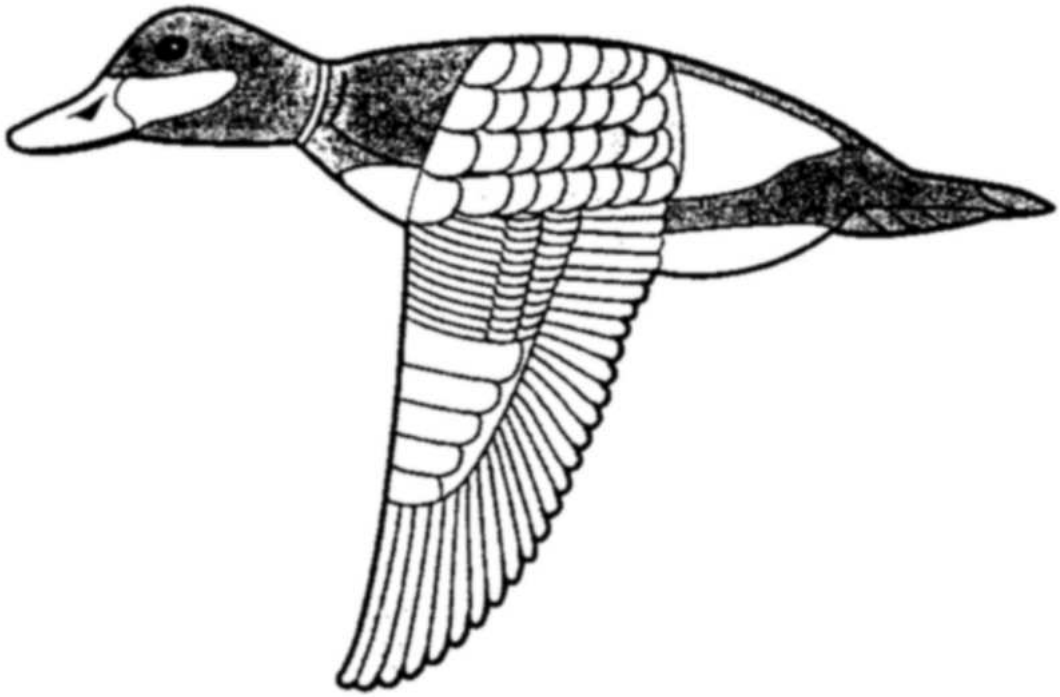
$$\begin{aligned} f'(b) &= \frac{1}{h_{n-1}}(a_n - a_{n-1}) - \frac{h_{n-1}}{3}(2c_{n-1}j + c_n) + h_{n-1}(c_{n-1} + c_n) \\ &= \frac{1}{h_{n-1}}(a_n - a_{n-1} + \frac{h_{n-1}}{3}(c_{n-1}j + 2c_n) \end{aligned}$$

dan akhirnya kita peroleh

$$h_{n-1}c_{n-1} + 2h_{n-1}C_n = 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \tag{8.14}$$

Persamaan (8.13) dan persamaan (8.14) ditambah persamaan (8.12 membentuk operasi matrik $\mathbf{Ax} = \mathbf{b}$ dimana

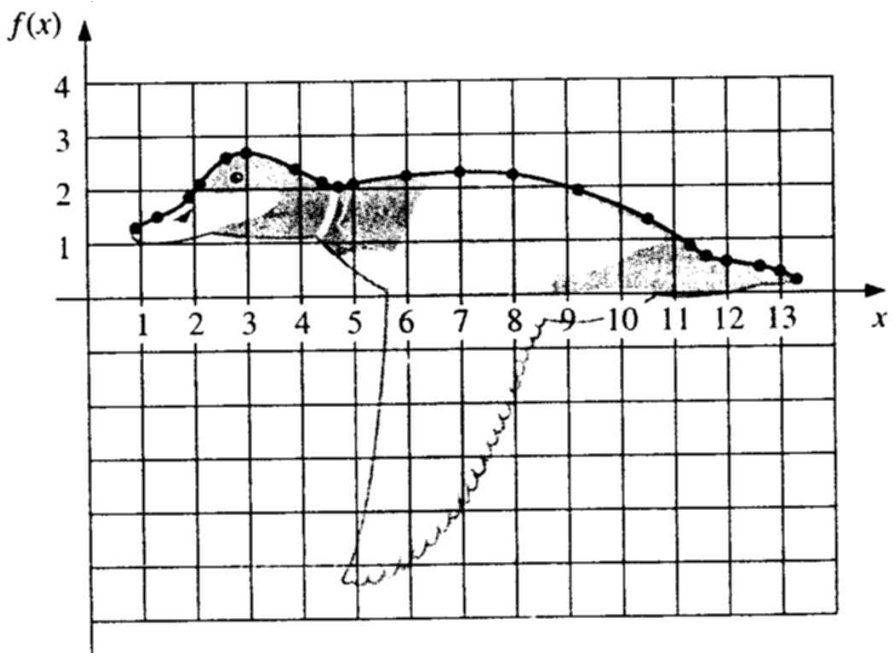
$$\mathbf{A} = \begin{bmatrix} 2h_0 & h_0 & 0 & \dots & \dots & \dots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & 0 & \dots & \dots & 0 \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \dots & \dots & \dots & 0 & h_{n-1} & 2h_{n-1} \end{bmatrix}$$



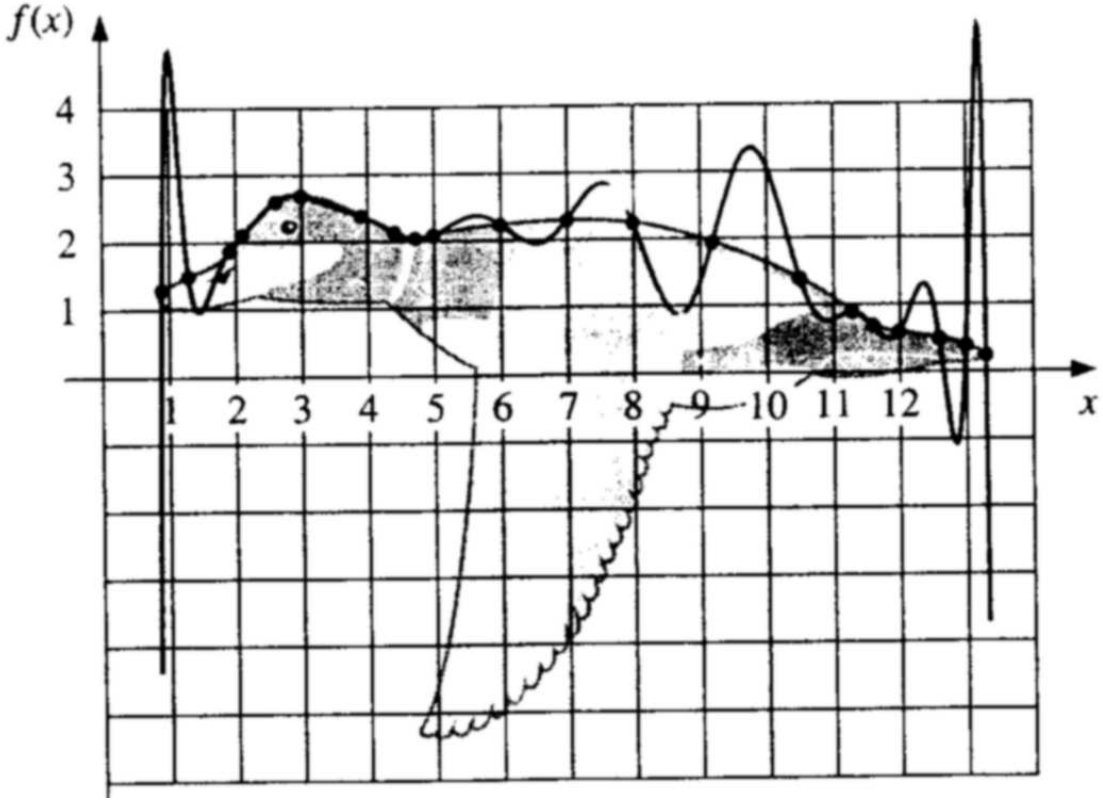
Gambar 8.3: Profil suatu object

$$\mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \end{bmatrix}$$

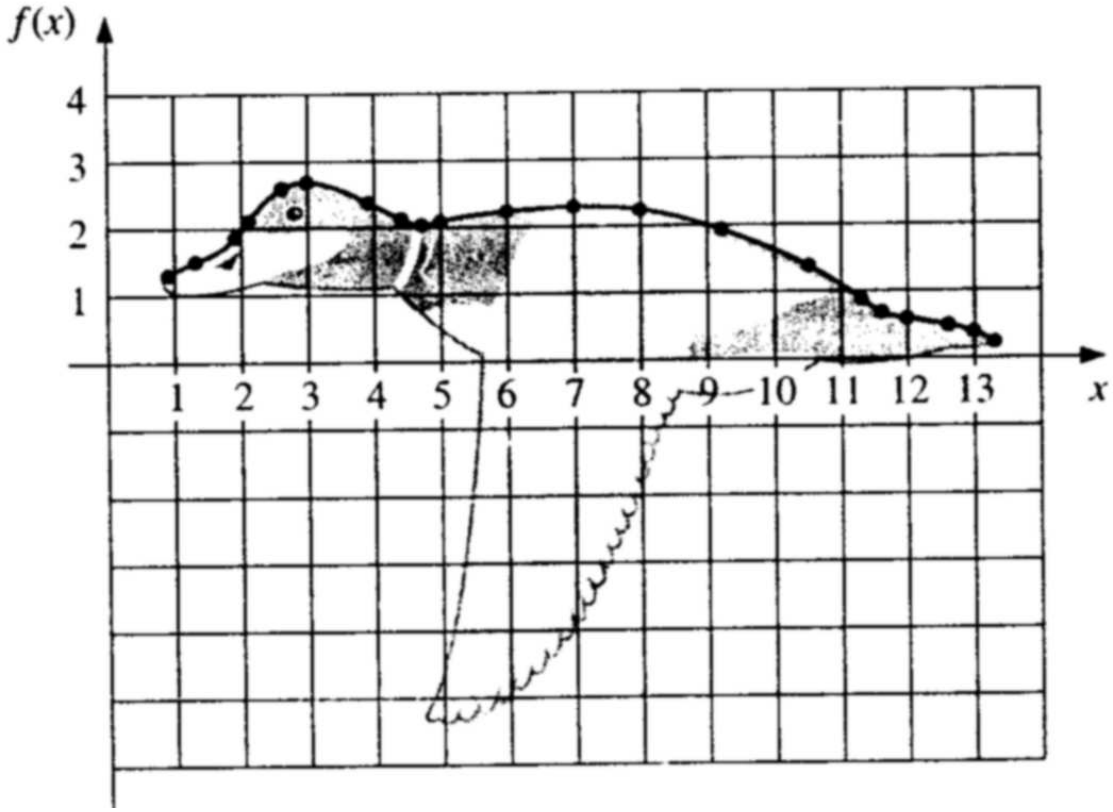


Gambar 8.4: Sampling titik data



Gambar 8.5: 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 8.6: Hasil interpolasi lagrange

Bab 9

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

9.1 Metode Euler

Suatu persamaan diferensial ($\frac{dy}{dt}$) dinyatakan dalam fungsi $f(t, y)$, dimana $y(t)$ adalah persamaan asalnya

$$\frac{dy}{dt} = f(t, y), \qquad a \leq t \leq b, \qquad y(a) = \alpha \tag{9.1}$$

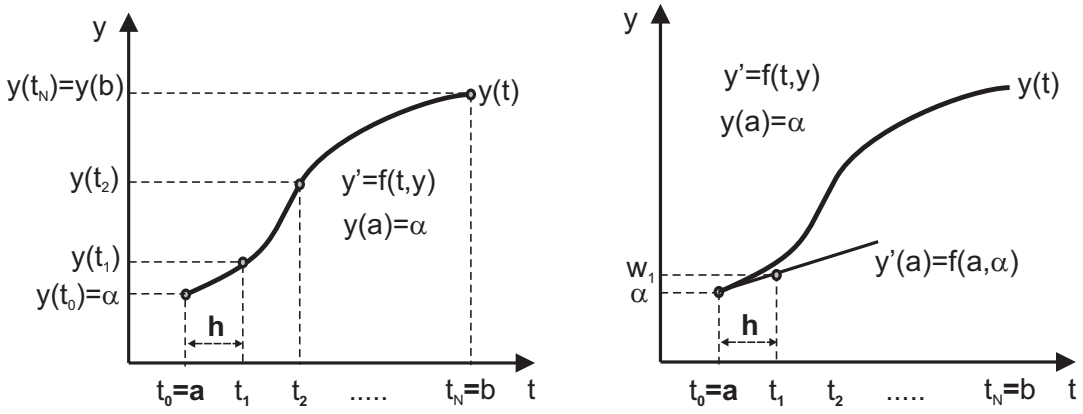
Nilai t dibatasi dari a hingga ke b. Sementara, syarat awal telah diketahui yaitu pada saat $t = a$ maka y bernilai α. Akan tetapi kita sama sekali tidak tahu bentuk formulasi persamaan asalnya $y(t)$. Gambar 9.1 memperlihatkan kurva persamaan asal $y(t)$ yang tidak diketahui bentuk formulasinya. Tantangannya 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

$$h = \frac{b - a}{N} \tag{9.2}$$

dengan N adalah bilangan *integer* positif. Nilai h ini juga dikenal dengan nama ***step size***. Selanjutnya nilai t diantara a dan b ditentukan berdasarkan

$$t_i = a + ih, \qquad i = 0, 1, 2, ..., N \tag{9.3}$$



Gambar 9.1: *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

$$y(t_{i+1}) = y(t_i) + (t_{i+1} - t_i)y'(t_i) + \frac{(t_{i+1} - t_i)^2}{2}y''(\xi_i) \tag{9.4}$$

dengan memasukkan $h = (t_{i+1} - t_i)$, maka

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi_i) \tag{9.5}$$

dan, karena $y(t)$ memenuhi persamaan diferensial (9.25), dimana $y'(t_i)$ tak lain adalah fungsi turunan $f(t_i, y(t_i))$, maka

$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{h^2}{2}y''(\xi_i) \tag{9.6}$$

Metode Euler dibangun dengan pendekatan bahwa suku terakhir dari persamaan (9.6), 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

$$w_{i+1} = w_i + hf(t_i, w_i) \qquad \text{dengan syarat awal} \quad w_0 = \alpha \tag{9.7}$$

dimana $i = 0, 1, 2, .., N - 1$.

Contoh

Diketahui persamaan diferensial

$$y' = y - t^2 + 1 \quad \text{batas interval:} \quad 0 \leq t \leq 2 \quad \text{syarat awal:} \quad y(0) = 0,5 \tag{9.8}$$

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 (9.8) adalah

$$y(t) = (t+1)^2 - 0,5e^t \tag{9.9}$$

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
```

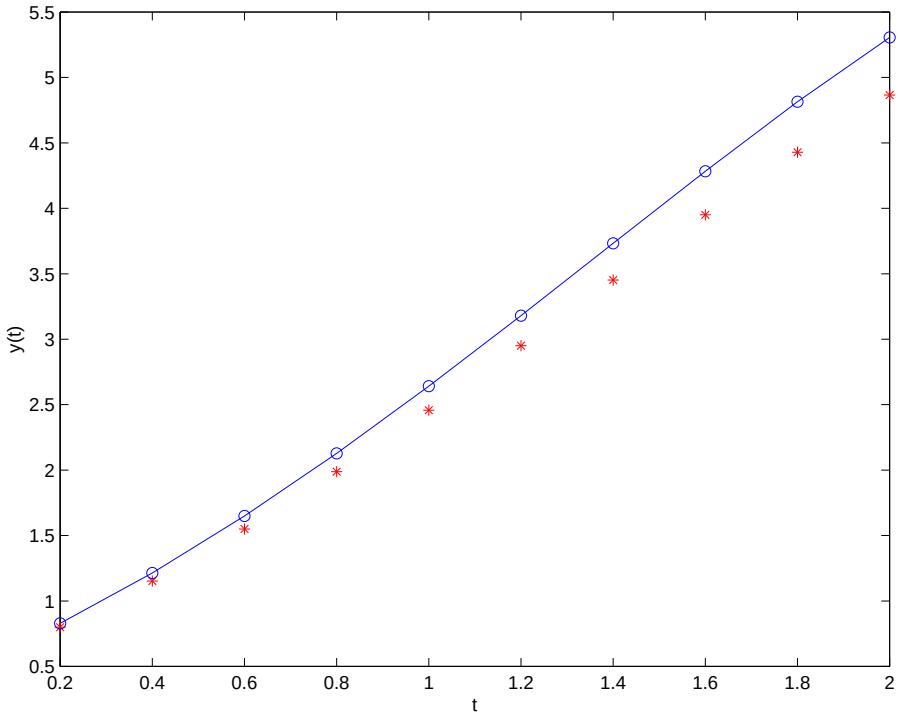
Tabel 9.1: Solusi yang ditawarkan oleh metode euler w_i dan solusi exact $y(t_i)$ serta selisih antara keduanya

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 (9.2) 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 9.2: 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 (9.9). 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.

9.2 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

$$\begin{aligned} w_0 &= \alpha \\ k_1 &= hf(t_i, w_i) && (9.10)\\ k_2 &= hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1) && (9.11)\\ k_3 &= hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2) && (9.12)\\ k_4 &= hf(t_{i+1}, w_i + k_3) && (9.13)\\ w_{i+1} &= w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) && (9.14) \end{aligned}$$

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
```

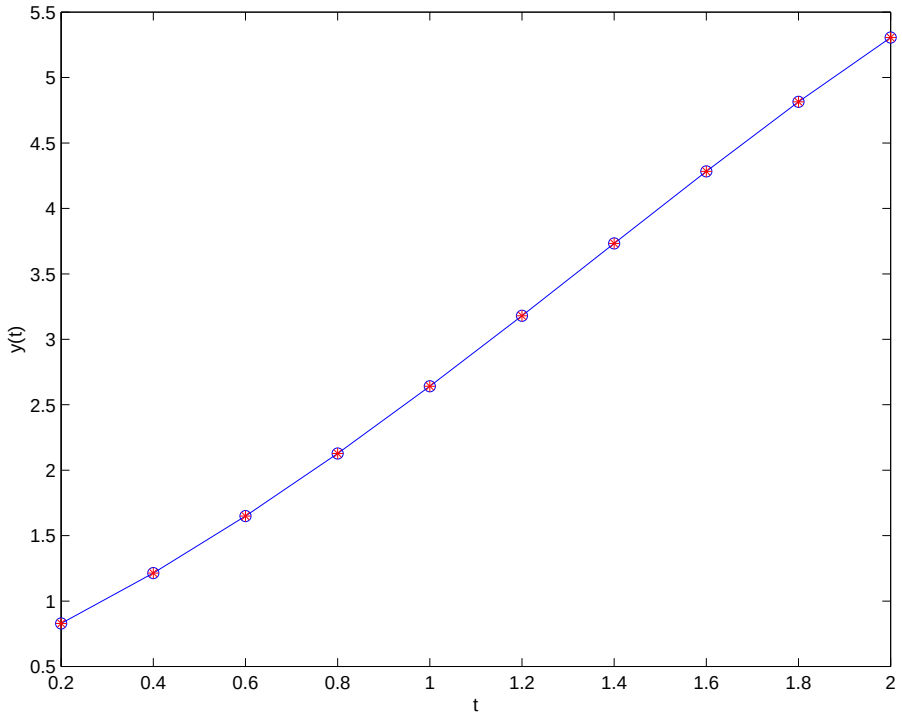
Dibandingkan dengan metode Euler, tingkat pertumbuhan *truncation error,* pada kolom $|w_i - y_i|$ (lihat Tabel 9.2), jauh lebih rendah sehingga metode Runge-Kutta Orde Empat lebih disukai untuk membantu menyelesaikan persamaan-diferensial-biasa.

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 9.2: 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 9.3: 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 (9.9). 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.

9.2.1 Aplikasi: Pengisian muatan pada kapasitor

Sebuah kapasitor yang tidak bermuatan dihubungkan secara seri dengan sebuah resistor dan baterry (Gambar 9.4). Diketahui ϵ = 12 volt, C = 5,00 μF dan R = 8,00 $\times 10^5 \Omega$. Saat saklar

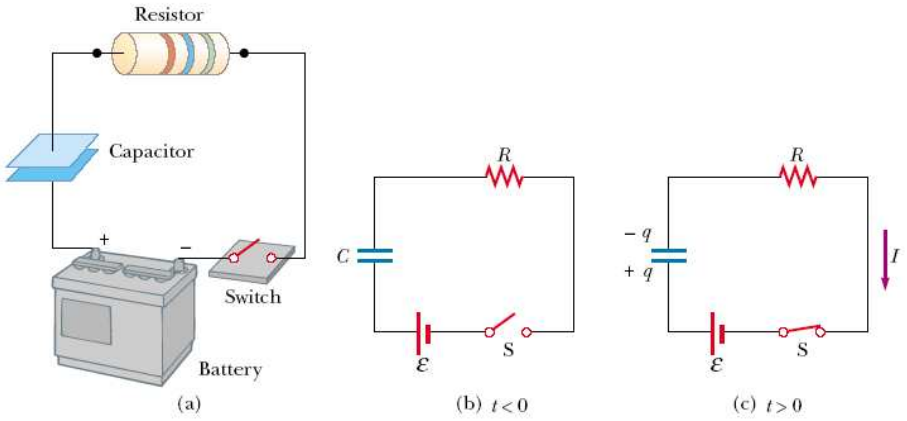
dihubungkan (t=0), muatan belum ada (q=0).

$$\frac{dq}{dt} = \frac{\epsilon}{R} - \frac{q}{RC} \tag{9.15}$$

Solusi exact persamaan (9.15) adalah

$$q_{exact} = q(t) = C\epsilon\left(1 - e^{-t/RC}\right) \tag{9.16}$$

Anda bisa lihat semua suku di ruas kanan persamaan (9.15) tidak mengandung variabel



Gambar 9.4: Rangkaian RC

t. Padahal persamaan-persamaan turunan pada contoh sebelumnya mengandung variabel t. Apakah persamaan (9.15) 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 (9.15) 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
```

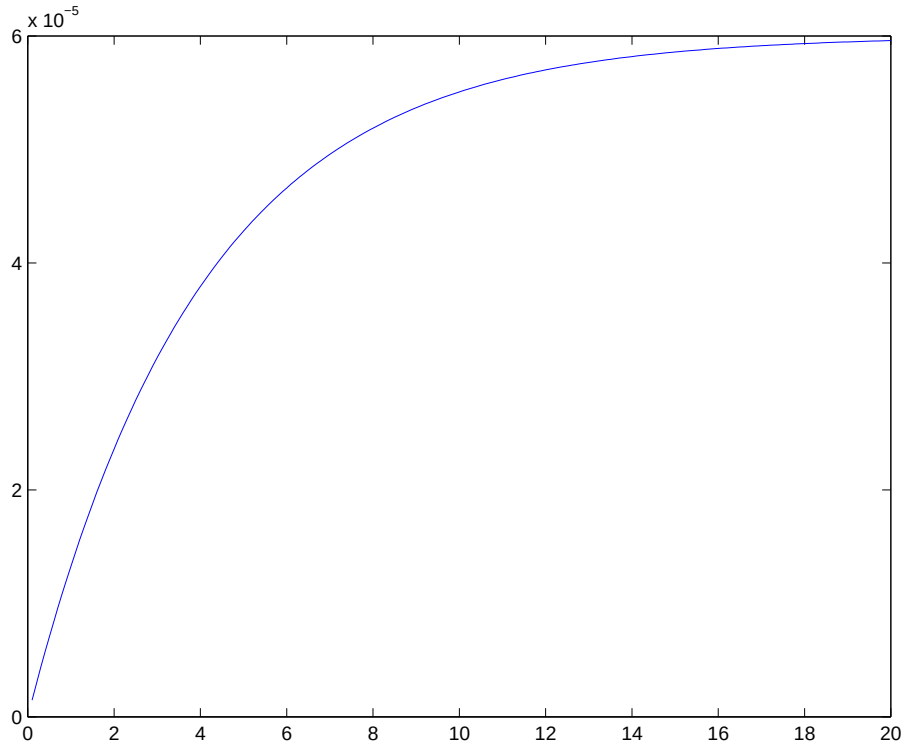
Adapun *script* fungsi turunannya (*futur.m*) adalah sebagai berikut:

```
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 9.3: Perbandingan antara hasil perhitungan numerik lewat metode Runge Kutta dan hasil perhitungan dari solusi exact, yaitu persamaan (9.16)

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 9.5 memperlihatkan kurva penumpukan muatan q terhadap waktu t – dengan batas atas interval waktu dinaikkan hingga 20 –.



Gambar 9.5: 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.

9.3 Metode *Finite Difference*

Suatu persamaan diferensial dapat dinyatakan sebagai berikut:

$$\frac{d^2y}{dx^2}(x) = p(x)\frac{dy}{dx}(x) + q(x)y(x) + r(x), \qquad a \leq x \leq b, \qquad y(a) = \alpha, \qquad y(b) = \beta \tag{9.17}$$

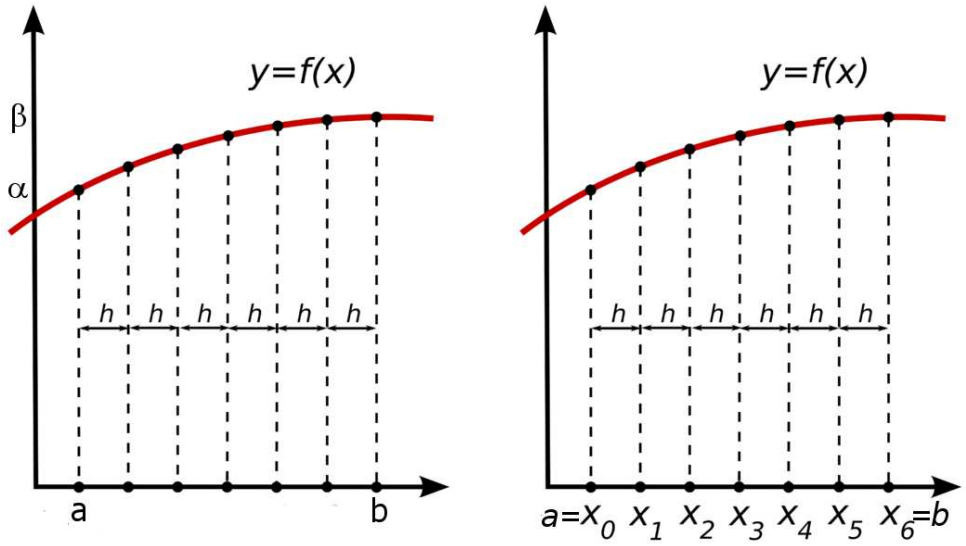
dimana a, b, α dan β adalah konstanta-konstanta yang sudah diketahui nilainya. Persamaan 9.17 dapat dinyatakan sebagai berikut

$$y'' = p(x)y' + q(x)y + r(x) \tag{9.18}$$

Persamaan 9.18 dapat diselesaikan melalui pendekatan numerik terhadap y'' dan y'. Caranya adalah pertama, kita menentukan sebuah angka integer[2] sembarang yang diberi nama N, dimana nilai N harus lebih besar dari nol, $N > 0$. Kemudian nilai interval h ditentukan dengan cara

$$h = \frac{b-a}{N+1} \tag{9.19}$$

[2]integer = bilangan asli yaitu 1,2,3,4,..dan seterusnya



Gambar 9.6: 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 $x_6 = b$

Lihat Gambar 9.6. Titik-titik x yang merupakan sub-interval antara a dan b dapat dinyatakan sebagai

$$x_i = a + ih, \qquad i = 0, 1, ..., N + 1 \tag{9.20}$$

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

$$y(x_{i+1}) = y(x_i + h) = y(x_i) + hy'(x_i) + \frac{h^2}{2}y''(x_i) \tag{9.21}$$

dan

$$y(x_{i-1}) = y(x_i - h) = y(x_i) - hy'(x_i) + \frac{h^2}{2}y''(x_i) \tag{9.22}$$

Jika kedua persamaan ini dijumlahkan, maka akan diperoleh

$$y(x_{i+1}) + y(x_{i-1}) = 2y(x_i) + h^2y''(x_i)$$

Dari sini $y''(x_i)$ dapat diturunkan rumusnya melalui langkah-langkah berikut

$$h^2y''(x_i) = y(x_{i+1}) - 2y(x_i) + y(x_{i-1})$$

$$y''(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} \tag{9.23}$$

Dengan cara yang sama, $y'(x_i)$ dapat dicari dengan menerapkan operasi pengurangan pada persamaan 9.21 dan 9.22, sehingga diperoleh

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_{i-1})}{2h} \tag{9.24}$$

Selanjutnya persamaan (9.23) dan (9.24) disubstitusikan kedalam persamaan (9.18)

$$\begin{aligned}
\frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} &= p(x_i)\frac{y(x_{i+1}) - y(x_{i-1})}{2h} + q(x_i)y(x_i) + r(x_i) \\
\frac{-y(x_{i+1}) + 2y(x_i) - y(x_{i-1})}{h^2} &= -p(x_i)\frac{y(x_{i+1}) - y(x_{i-1})}{2h} - q(x_i)y(x_i) - r(x_i) \\
\frac{-y(x_{i+1}) + 2y(x_i) - y(x_{i-1})}{h^2} &+ p(x_i)\frac{y(x_{i+1}) - y(x_{i-1})}{2h} + q(x_i)y(x_i) = -r(x_i)
\end{aligned} \tag{9.25}$$

Sebelum dilanjut, saya ingin menyatakan bahwa

$$\begin{aligned}
w_{i+1} &= y(x_{i+1}) \\
w_i &= y(x_i) \\
w_{i-1} &= y(x_{i-1})
\end{aligned}$$

sehingga persamaan 9.25 di atas dapat ditulis sebagai berikut

$$\begin{aligned}
\left(\frac{-w_{i+1} + 2w_i - w_{i-1}}{h^2}\right) + p(x_i)\left(\frac{w_{i+1} - w_{i-1}}{2h}\right) + q(x_i)w_i &= -r(x_i) \\
(-w_{i+1} + 2w_i - w_{i-1}) + \frac{h}{2}p(x_i)(w_{i+1} - w_{i-1}) + h^2 q(x_i)w_i &= -h^2 r(x_i) \\
-w_{i+1} + 2w_i - w_{i-1} + \frac{h}{2}p(x_i)w_{i+1} - \frac{h}{2}p(x_i)w_{i-1} + h^2 q(x_i)w_i &= -h^2 r(x_i) \\
-w_{i-1} - \frac{h}{2}p(x_i)w_{i-1} + 2w_i + h^2 q(x_i)w_i - w_{i+1} + \frac{h}{2}p(x_i)w_{i+1} &= -h^2 r(x_i) \\
-\left(1 + \frac{h}{2}p(x_i)\right)w_{i-1} + \left(2 + h^2 q(x_i)\right)w_i - \left((1 - \frac{h}{2}p(x_i)\right)w_{i+1} &= -h^2 r(x_i)
\end{aligned} \tag{9.26}$$

Persamaan 9.26 dikenal sebagai persamaan *finite difference* 1 dimensi, dimana i=1,2,3...sampai N. Mengapa i=0 dan i=$N+1$ tidak dimasukkan? Karena yang ingin kita cari adalah w_1, w_2, w_3, dan seterusnya sampai w_N. Sementara, w_0 dan w_{N+1} biasanya **sudah diketahui** sebelumnya, yaitu w_0=α dan w_{N+1}=β. Keduanya dikenal sebagai **syarat batas** atau istilah asingnya adalah **boundary value**. Oleh karena itu, topik yang sedang dibahas ini sering juga disebut sebagai **Masalah Syarat Batas** atau **Boundary Value Problem**.

Sampai disini kita mendapatkan sistem persamaan linear yang selanjutnya dapat dinyatakan sebagai bentuk operasi matrik

$$\mathbf{Aw} = \mathbf{b} \tag{9.27}$$

dimana **A** adalah matrik tridiagonal dengan orde $N \times N$

$$\mathbf{A} = \begin{bmatrix}
2 + h^2 q(x_1) & -1 + \frac{h}{2}p(x_1) & 0 & \dots & \dots & \dots & 0 \\
-1 - \frac{h}{2}p(x_2) & 2 + h^2 q(x_2) & -1 + \frac{h}{2}p(x_2) & 0 & \dots & \dots & 0 \\
0 & -1 - \frac{h}{2}p(x_3) & 2 + h^2 q(x_3) & -1 + \frac{h}{2}p(x_3) & 0 & \dots & 0 \\
0 & 0 & -1 - \frac{h}{2}p(x_4) & 2 + h^2 q(x_4) & -1 + \frac{h}{2}p(x_4) & 0 & 0 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots \\
\dots & \dots & \dots & \dots & -1 - \frac{h}{2}p(x_{N-1}) & 2 + h^2 q(x_{N-1}) & -1 + \frac{h}{2}p(x_{N-1}) \\
0 & \dots & \dots & \dots & \dots & -1 - \frac{h}{2}p(x_N) & 2 + h^2 q(x_N)
\end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ \vdots \\ w_{N-1} \\ w_N \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} -h^2 r(x_1) + \left(1 + \frac{h}{2} p(x_1)\right) w_0 \\ -h^2 r(x_2) \\ -h^2 r(x_3) \\ -h^2 r(x_4) \\ \vdots \\ -h^2 r(x_{N-1}) \\ -h^2 r(x_N) + \left(1 - \frac{h}{2} p(x_N)\right) w_{N+1} \end{bmatrix}$$

9.3.1 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

9.3.2 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

$$\mathbf{Aw} = \mathbf{b} \tag{9.28}$$

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

$$w_1 = -0.3157 \qquad w_2 = -0.2829 \qquad w_3 = -0.2070$$

9.4 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

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y) \tag{9.29}$$

Di bidang fisika, persamaan (9.29) dikenal sebagai **Persamaan Poisson**. Jika $f(x, y)$=0, maka diperoleh persamaan yang lebih sederhana

$$\frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = 0 \tag{9.30}$$

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

$$\frac{\partial u}{\partial t}(x, t) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x, t) = 0 \tag{9.31}$$

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

$$\alpha^2 \frac{\partial^2 u}{\partial^2 x}(x, t) = \frac{\partial^2 u}{\partial t^2}(x, t) \tag{9.32}$$

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*.

9.5 PDP eliptik

Kita mulai dari persamaan aslinya

$$\frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = f(x, y) \tag{9.33}$$

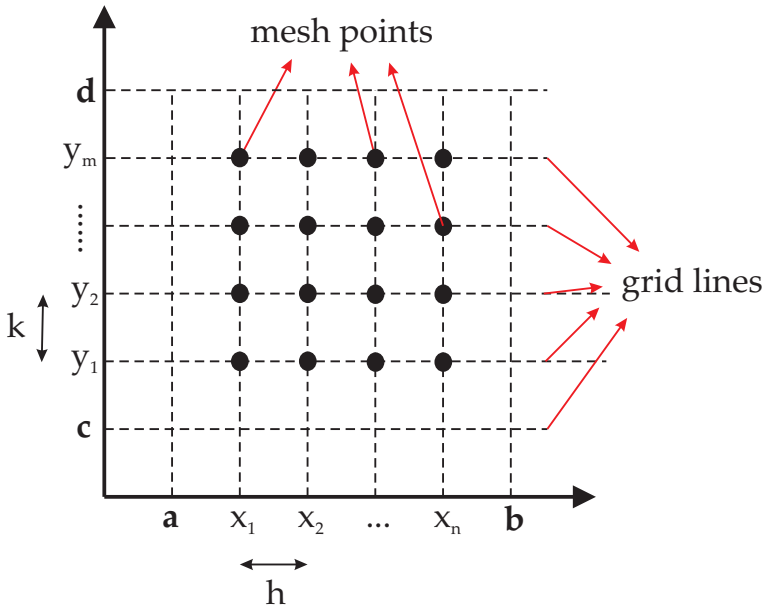
dimana $R = [(x, y)|a < x < b, c < y < d]$. Maksudnya, variasi titik-titik x berada di antara a dan b. Demikian pula dengan variasi titik-titik y, dibatasi mulai dari c sampai d (lihat Gambar 9.7). Jika h adalah jarak interval antar titik yang saling bersebelahan pada titik-titik dalam rentang horizontal a dan b, maka titik-titik variasi di antara a dan b dapat diketahui melalui rumus ini

$$x_i = a + ih, \quad \text{dimana } i = 1, 2, \ldots, n \tag{9.34}$$

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

$$y_j = c + jk, \quad \text{dimana } j = 1, 2, \ldots, m \tag{9.35}$$

dimana c adalah titik awal pada sumbu vertikal y. Perhatikan Gambar 9.7, 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 9.7: Skema *grid lines* dan *mesh points* pada aplikasi metode *Finite-Difference*

mesh points.

Turunan kedua sebagaimana yang ada pada persamaan (9.33) dapat dinyatakan dalam rumus *centered-difference* sebagai berikut

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, y_j) \tag{9.36}$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} - \frac{k^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j) \tag{9.37}$$

Metode *Finite-Difference* biasanya mengabaikan suku yang terakhir, sehingga cukup dinyatakan sebagai

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} \tag{9.38}$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} \tag{9.39}$$

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 (9.38) dan (9.39) disubstitusi ke persamaan (9.33), hasilnya adalah

$$\frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} = f(x_i, y_j) \tag{9.40}$$

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 (9.40) dinyatakan dalam notasi w, sebagai berikut

$$\begin{aligned}
\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} + \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{k^2} &= f(x_i, y_j) \\
w_{i+1,j} - 2w_{i,j} + w_{i-1,j} + \frac{h^2}{k^2}(w_{i,j+1} - 2w_{i,j} + w_{i,j-1}) &= h^2 f(x_i, y_j) \\
w_{i+1,j} - 2w_{i,j} + w_{i-1,j} + \frac{h^2}{k^2}w_{i,j+1} - 2\frac{h^2}{k^2}w_{i,j} + \frac{h^2}{k^2}w_{i,j-1} &= h^2 f(x_i, y_j) \\
-2[1 + \frac{h^2}{k^2}]w_{i,j} + (w_{i+1,j} + w_{i-1,j}) + \frac{h^2}{k^2}(w_{i,j+1} + w_{i,j-1}) &= h^2 f(x_i, y_j) \\
2[1 + \frac{h^2}{k^2}]w_{i,j} - (w_{i+1,j} + w_{i-1,j}) - \frac{h^2}{k^2}(w_{i,j+1} + w_{i,j-1}) &= -h^2 f(x_i, y_j) \qquad (9.41)
\end{aligned}$$

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 (9.41) adalah rumusan akhir metode *Finite-Difference* untuk PDP Eliptik.

9.5.1 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:

$$\frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = 0; \quad 0 < x < 0,5, \quad 0 < y < 0,5$$

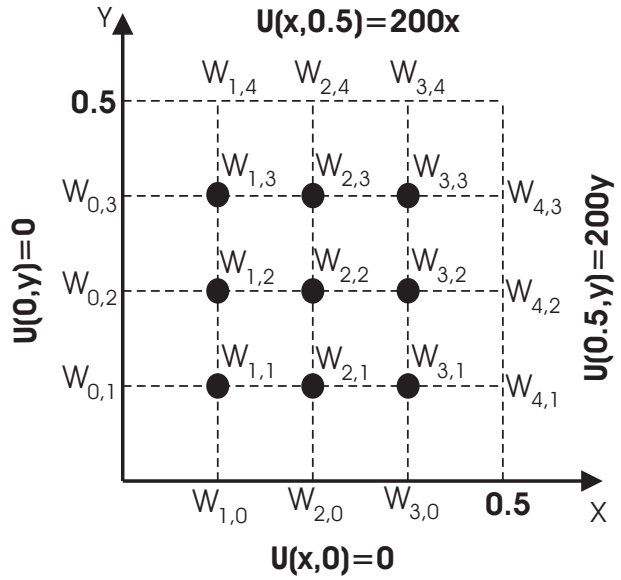
dengan syarat-syarat batas

$$u(0, y) = 0, \quad u(x, 0) = 0, \quad u(x, 0.5) = 200x, \quad u(0.5, y) = 200y$$

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 9.8). Langkah berikutnya adalah menyusun persamaan *Finite-Difference*, dimulai dari persamaan



Gambar 9.8: Susunan *grid lines* dan *mesh points* untuk mensimulasikan distribusi temperatur pada lempeng logam sesuai contoh satu

asalnya (persamaan 9.41)

$$2[1+\frac{h^2}{k^2}]w_{i,j} - (w_{i+1,j} + w_{i-1,j}) - \frac{h^2}{k^2}(w_{i,j+1} + w_{i,j-1}) = -h^2 f(x_i, y_j)$$

Karena $h = k = 0,125$ dan $f(x_i, y_j) = 0$, maka

$$4w_{i,j} - w_{i+1,j} - w_{i-1,j} - w_{i,j-1} - w_{i,j+1} = 0 \tag{9.42}$$

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 (9.42) 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.

9.5.2 *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.

9.5.3 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

$$\left| w_{ij}^{(l)} - w_{ij}^{(l-1)} \right| \le 10^{-10}$$

9.6 PDP parabolik

PDP parabolik yang kita pelajari disini adalah persamaan difusi

$$\frac{\partial u}{\partial t}(x,t) = \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t), \quad 0 < x < \ell, \quad t > 0, \tag{9.43}$$

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.

9.6.1 Metode *Forward-difference*

Solusi numerik diperoleh menggunakan **forward-difference**[3] 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 (9.43) terhadap t, dengan *time step* k, adalah

$$\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_j + k) - u(x_i, t_j)}{k} - \frac{k}{2}\frac{\partial^2 u}{\partial t^2}(x_i, \mu_j) \tag{9.44}$$

Namun, sebagaimana pendekatan *finite-difference* pada umumnya, pendekatan *forward-difference* selalu mengabaikan suku terakhir, sehingga persamaan di atas ditulis seperti ini

$$\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_j + k) - u(x_i, t_j)}{k} \tag{9.45}$$

[3]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 (9.43) terhadap x berdasarkan deret Taylor adalah

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) = \frac{u(x_i + h, t_J) - 2u(x_i, t_j) + u(x_i - h, t_J)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, t_j) \tag{9.46}$$

Pengabaian suku terakhir menjadikan persamaan di atas ditulis kembali sebagai berikut

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) = \frac{u(x_i + h, t_j) - 2u(x_i, t_j) + u(x_i - h, t_j)}{h^2} \tag{9.47}$$

Kemudian persamaan (9.45) dan (9.47) disubstitusi kedalam persamaan (9.43), maka diperoleh

$$\frac{u(x_i, t_j + k) - u(x_i, t_j)}{k} = \alpha^2 \frac{u(x_i + h, t_j) - 2u(x_i, t_j) + u(x_i - h, t_j)}{h^2} \tag{9.48}$$

atau dapat dinyatakan dalam notasi w

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0 \tag{9.49}$$

Dari sini diperoleh solusi untuk $w_{i,j+1}$, yaitu

$$w_{i,j+1} = \left(1 - \frac{2\alpha^2 k}{h^2}\right) w_{i,j} + \alpha^2 \frac{k}{h^2}(w_{i+1,j} + w_{i-1,j}) \tag{9.50}$$

jika

$$\lambda = \frac{\alpha^2 k}{h^2} \tag{9.51}$$

maka

$$(1 - 2\lambda) w_{i,j} + \lambda w_{i+1,j} + \lambda w_{i-1,j} = w_{i,j+1} \tag{9.52}$$

9.6.2 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

$$u(0,t) = u(1,t) = 0, \quad 0 < t,$$

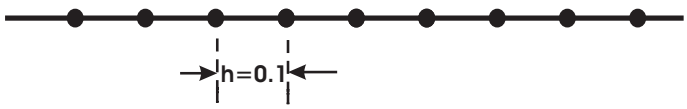
dan kondisi mula-mula

$$u(x,0) = \sin(\pi x), \quad 0 \le x \le 1,$$

Solusi analitik atas masalah ini adalah

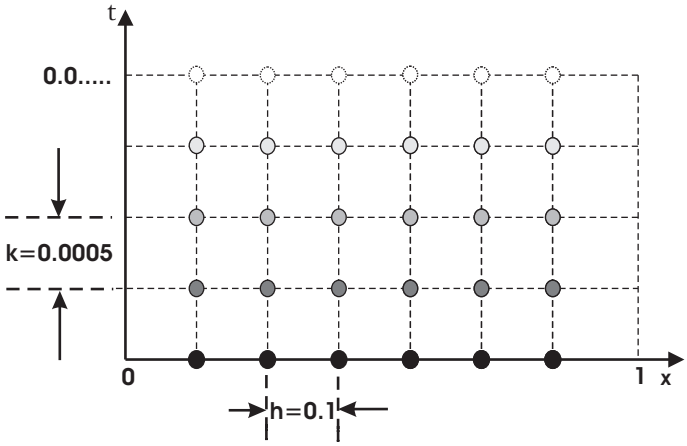
$$u(x,t) = e^{-\pi^2 t}\sin(\pi x)$$

Adapun sebaran posisi *mesh-points* dalam 1-D diperlihatkan pada Gambar 9.9. Sementara



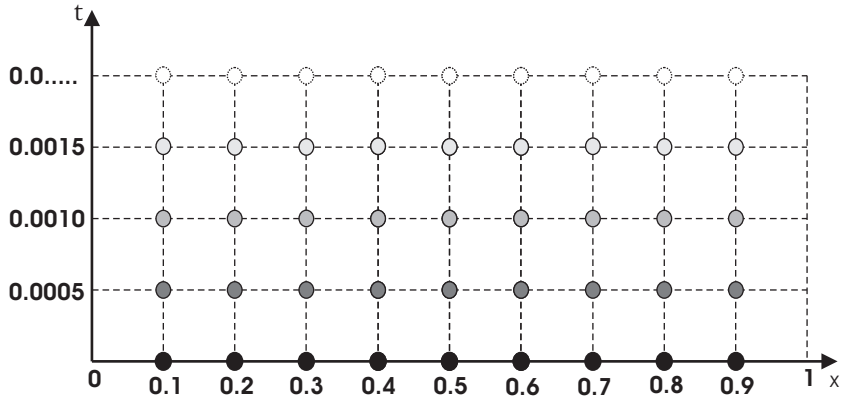
Gambar 9.9: Sebatang logam dengan posisi titik-titik simulasi (*mesh-points*) distribusi temperatur. Jarak antar titik ditentukan sebesar $h = 0,1$.

Gambar 9.10 melengkapi Gambar 9.9, dimana perubahan waktu tercatat setiap interval $k = 0,0005$. Sepintas Gambar 9.10 terlihat seolah-olah obyek yang mau disimulasikan berbentuk 2-dimensi, padahal bendanya tetap 1-dimensi yaitu hanya sebatang logam.



Gambar 9.10: Interval *mesh-points* dengan jarak $h = 0,1$ dalam interval waktu $k = 0,0005$

Selanjutnya, Gambar 9.11 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 9.11: 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 (9.51)

$$\lambda = \frac{\alpha^2 k}{h^2} = \frac{0,1}{0,0005^2} = 0,05$$

Berdasarkan persamaan (9.52), 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

$$\begin{bmatrix}
0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9 & 0,5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,5 & 0,9
\end{bmatrix}
\begin{bmatrix} w_{1,j} \\ w_{2,j} \\ w_{3,j} \\ w_{4,j} \\ w_{5,j} \\ w_{6,j} \\ w_{7,j} \\ w_{8,j} \\ w_{9,j} \end{bmatrix}
=
\begin{bmatrix} w_{1,j+1} \\ w_{2,j+1} \\ w_{3,j+1} \\ w_{4,j+1} \\ w_{5,j+1} \\ w_{6,j+1} \\ w_{7,j+1} \\ w_{8,j+1} \\ w_{9,j+1} \end{bmatrix} \tag{9.53}$$

Persamaan matriks di atas dapat direpresentasikan sebagai

$$A\mathbf{w}^{(j)} = \mathbf{w}^{(j+1)} \tag{9.54}$$

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 [4]. 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[5].

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[6]. Untuk

[4] Topik tentang perkalian matrik sudah diulas pada Bab 1

[5] karena *step time* k-nya sudah ditentukan sebesar $0,0005$

[6] 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.

9.6.2.1 *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 9.4 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 9.4: 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	

9.6.3 Metode *Backward-difference*

Kalau kita ulang lagi pelajaran yang lalu tentang *forward-difference*, kita akan dapatkan formula *forward-difference* adalah sebagai berikut (lihat persamaan (9.49))

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0$$

Sekarang, dengan sedikit modifikasi, formula *backward-difference* dinyatakan sebagai

$$\frac{w_{i,j} - w_{i,j-1}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0 \tag{9.55}$$

jika ditetapkan

$$\lambda = \frac{\alpha^2 k}{h^2}$$

maka *backward-difference* disederhanakan menjadi

$$(1 + 2\lambda)\, w_{i,j} - \lambda w_{i+1,j} - \lambda w_{i-1,j} = w_{i,j-1} \tag{9.56}$$

coba sejenak anda bandingkan dengan formula *forward-difference* dalam λ sebagaimana dinyatakan oleh persamaan (9.52)

$$(1-2\lambda)\, w_{i,j} + \lambda w_{i+1,j} + \lambda w_{i-1,j} = w_{i,j+1}$$

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 (9.51) kembali

$$\lambda = \frac{\alpha^2 k}{h^2} = \frac{0,1}{0,01^2} = 1$$

Berdasarkan persamaan (9.56), 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

$$A\mathbf{w}^{(j)} = \mathbf{w}^{(j-1)} \tag{9.57}$$

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[7]. Untuk jumlah iterasi hingga $j = 50$, perhitungannya dilakukan dalam *script* Matlab.

9.6.3.1 *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
```

[7]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 9.5: 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	

9.6.4 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

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j+1} - 2w_{i,j+1} + w_{i-1,j+1}}{h^2} = 0 \tag{9.58}$$

Jika formula ini dijumlahkan dengan formula *forward-difference*, kemudian hasilnya dibagi 2, maka akan diperoleh

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \frac{\alpha^2}{2}\left[\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} + \frac{w_{i+1,j+1} - 2w_{i,j+1} + w_{i-1,j+1}}{h^2}\right] = 0 \tag{9.59}$$

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

$$-\frac{\lambda}{2}w_{i-1,j+1} + (1+\lambda)w_{i,j+1} - \frac{\lambda}{2}w_{i+1,j+1} = \frac{\lambda}{2}w_{i-1,j} + (1-\lambda)w_{i,j} + \frac{\lambda}{2}w_{i+1,j} \tag{9.60}$$

Dalam bentuk persamaan matrik dinyatakan sebagai

$$A\mathbf{w}^{(j+1)} = B\mathbf{w}^{(j)}, \qquad \text{untuk } j = 0, 1, 2, ... \tag{9.61}$$

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 (9.60) 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 9.6: 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*.

9.7 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

$$\frac{\partial^2 u}{\partial t^2}(x,t) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t) = 0, \quad 0 < x < \ell, \quad t > 0 \tag{9.62}$$

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

$$\frac{\partial^2 u}{\partial t^2}(x_i,t_j) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x_i,t_j) = 0 \tag{9.63}$$

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 (9.63)

$$\frac{u(x_i,t_{j+1}) - 2u(x_i,t_j) + u(x_i,t_{j-1})}{k^2} - \alpha^2 \frac{u(x_{i+1},t_j) - 2u(x_i,t_j) + u(x_{i-1},t_j)}{h^2} = 0$$

maka dapat diturunkan formula finite-difference untuk PDP hiperbolik sebagai berikut

$$\frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{k^2} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0 \tag{9.64}$$

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

$$w_{i,j+1} = 2\left(1 - \lambda^2\right) w_{i,j} + \lambda^2 (w_{i+1,j} + w_{i-1,j}) - w_{i,j-1} \tag{9.65}$$

dengan $i = 1, 2, ..., m-1$ dan $j = 1, 2, ...$ Kondisi syarat batas ditentukan sebagai berikut

$$w_{0,j} = w_{m,j} = 0, \quad \text{untuk} \quad j = 1, 2, 3, ... \tag{9.66}$$

sementara kondisi awal dinyatakan

$$w_{i,0} = f(x_i), \quad \text{untuk} \quad i = 1, 2, ..., m-1 \tag{9.67}$$

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 (9.67) sementara nilai untuk $j = 1$ untuk menghitung $w_{i,2}$, harus diperoleh lewat kondisi kecepatan awal

$$\frac{\partial u}{\partial t}(x, 0) = g(x), \quad 0 \le x \le \ell \tag{9.68}$$

Salah satu cara pemecahan dengan pendekatan forward-difference adalah

$$\frac{\partial u}{\partial t}(x_i, 0) = \frac{u(x_i, t_1) - u(x_i, 0)}{k} \tag{9.69}$$

$$\begin{aligned} u(x_i, t_1) &= u(x_i, 0) + k\frac{\partial u}{\partial t}(x_i, 0) \\ &= u(x_i, 0) + kg(x_i) \end{aligned}$$

konsekuensinya

$$w_{i,1} = w_{i,0} + kg(x_i), \quad \text{untuk} \quad i = 1, 2, ..., m-1 \tag{9.70}$$

9.7.1 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 (9.62), 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 (9.65)

$$\begin{aligned} w_{i,j+1} &= 2\left(1-\lambda^2\right)w_{i,j} + \lambda^2\left(w_{i+1,j}+w_{i-1,j}\right) - w_{i,j-1} \\ w_{i,j+1} &= 2\left(1-1^2\right)w_{i,j} + 1^2\left(w_{i+1,j}+w_{i-1,j}\right) - w_{i,j-1} \\ w_{i,j+1} &= 0w_{i,j} + \left(w_{i+1,j}+w_{i-1,j}\right) - w_{i,j-1} \end{aligned}$$

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!!

9.8 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

$$u(x,t) = 2e^{-(\pi^2/4)t}\sin 2\pi x$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + k\frac{\partial u}{\partial t}\left(x_i, 0\right) + \frac{k^2}{2}\frac{\partial^2 u}{\partial t^2}\left(x_i, 0\right) + \frac{k^3}{6}\frac{\partial^3 u}{\partial t^3}\left(x_i, \hat{\mu}_i\right) \tag{9.71}$$

$$\frac{\partial^2 u}{\partial t^2}\left(x_i, 0\right) = \alpha^2\frac{\partial^2 u}{\partial x^2}\left(x_i, 0\right) = \alpha^2\frac{d^f}{dx^2}\left(x_i\right) = \alpha^2 f\text{"}\left(x_i\right) \tag{9.72}$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + kg\left(x_i\right) + \frac{\alpha^2 k^2}{2} f"\left(x_i\right) + \frac{k^3}{6}\frac{\partial^3 u}{\partial t^3}\left(x_i, \hat{\mu}_i\right) \tag{9.73}$$

$$w_{i1} = w_{i0} + kg\left(x_i\right) + \frac{\alpha^2 k^2}{2} f"\left(x_i\right) \tag{9.74}$$

$$f"\left(x_i\right) = \frac{f\left(x_{i+1}\right) - 2f\left(x_i\right) + f\left(x_{i-1}\right)}{h^2} - \frac{h^2}{12} f^{(4)}\left(\tilde{\xi}\right) \tag{9.75}$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + kg\left(x_i\right) + \frac{k^2\alpha^2}{2h^2}\left[f\left(x_{i+1}\right) - 2f\left(x_i\right) + f\left(x_{i-1}\right)h^2\right] + O\left(k^3 + h^2k^2\right) \tag{9.76}$$

$$u\left(x_i, t_1\right) = u\left(x_i, 0\right) + kg\left(x_i\right) + \frac{\lambda^2}{2}\left[f\left(x_{i+1}\right) - 2f\left(x_i\right) + f\left(x_{i-1}\right)h^2\right] + O\left(k^3 + h^2k^2\right) \tag{9.77}$$

$$= \left(1 - \lambda^2\right) f\left(x_i\right) + \frac{\lambda^2}{2} f\left(x_{i+1}\right) + \frac{\lambda^2}{2} f\left(x_{i-1}\right) + kg\left(x_i\right) + O\left(k^3 + h^2k^2\right) \tag{9.78}$$

$$w_{i,1} = \left(1 - \lambda^2\right) f\left(x_i\right) + \frac{\lambda^2}{2} f\left(x_{i+1}\right) + \frac{\lambda^2}{2} f\left(x_{i-1}\right) + kg\left(x_i\right) \tag{9.79}$$

Bab 10

Integral Numerik

Objektif :

- ▷ Mengenalkan metode Trapezoida
- ▷ Mengenalkan metode Simpson
- ▷ Mengenalkan metode Composite-Simpson
- ▷ Mengenalkan metode Adaptive Quardrature
- ▷ Mengenalkan metode Gaussian Quadrature

10.1 Metode Trapezoida

Suatu persamaan integral

$$\int_a^b f(x)dx \tag{10.1}$$

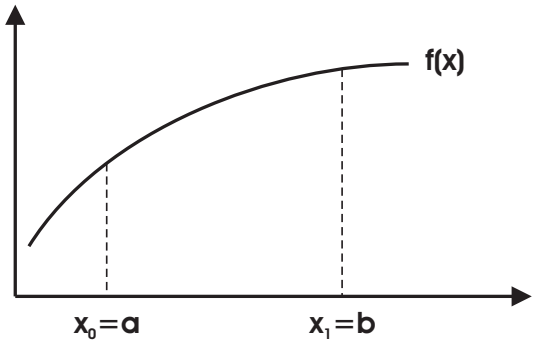
disebut *numerical quadrature*. Pendekatan numerik untuk menyelesaikan integral tersebut adalah

$$\sum_{i=0}^{n} a_i f(x_i) \tag{10.2}$$

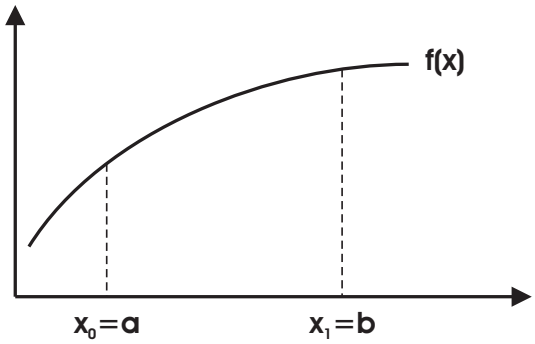
Adapun metode pendekatan yang paling dasar dalam memecahkan masalah integral secara numerik adalah metode Trapezoida yang rumusnya seperti ini

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{h^3}{12}f''(\xi) \tag{10.3}$$

dimana $x_0 = a$, $x_1 = b$ dan $h = b - a$. Karena bagian *error* pada Trapezoida adalah f'', maka pendekatan Trapezoida bekerja efektif pada fungsi-fungsi yang turunan kedua-nya bernilai nol ($f'' = 0$).



Gambar 10.1: 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 luas area integrasi, dimana luas area adalah sama dengan luas trapesium di bawah kurva $f(x)$ dalam batas-batas a dan b



Gambar 10.2: 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-batas a dan b

10.2 Metode Simpson

Metode pendekatan yang lebih baik dalam integral numerik adalah metode Simpson yang formulasinya seperti ini

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90}f^4(\xi) \tag{10.4}$$

dengan $x_0 = a$, $x_2 = b$, dan $x_1 = a + h$ dimana $h = (b - a)/2$.

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

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

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{h^3}{12}f''(\xi) \tag{10.5}$$

ketika $n = 2$: Simpson

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] - \frac{h^5}{90}f^4(\xi) \tag{10.6}$$

ketika $n = 3$: Simpson tiga-per-delapan

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] - \frac{3h^5}{80}f^4(\xi) \tag{10.7}$$

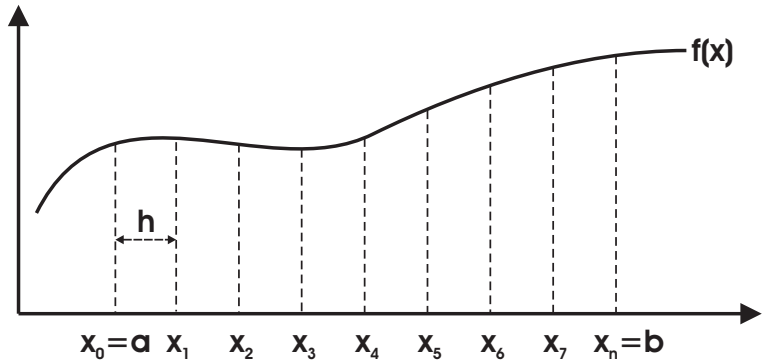
ketika $n = 4$:

$$\int_{x_0}^{x_4} f(x)dx = \frac{2h}{45}\left[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)\right] - \frac{8h^7}{945}f^6(\xi) \tag{10.8}$$

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 $n > 4$.

10.3 Metode Composite-Simpson

Persamaan (10.8) 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 (10.8).



Gambar 10.3: 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 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 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

$$\begin{aligned}\int_a^b f(x)dx &= \sum_{j=1}^{n/2}\int_{x_{2j-2}}^{x_{2j}} f(x)dx \\ &= \sum_{j=1}^{n/2}\left\{\frac{h}{3}\left[f(x_{2j-2})+4f(x_{2j-1})+f(x_{2j})\right]-\frac{h^5}{90}f^{(4)}(\xi_j)\right\} \qquad (10.9)\end{aligned}$$

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

$$\int_a^b f(x)dx=\frac{h}{3}\left[f(x_0)+2\sum_{j=1}^{(n/2)-1}f(x_{2j})+4\sum_{j=1}^{n/2}f(x_{2j-1})+f(x_n)\right]-\frac{h^5}{90}\sum_{j=1}^{n/2}f^{(4)}(\xi_j) \qquad (10.10)$$

Formula ini dikenal sebagai metode **Composite Simpson**.

10.4 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 $h=(b-a)/2$

$$\int_a^b f(x)dx=S(a,b)-\frac{h^5}{90}f^{(4)}(\mu) \qquad (10.11)$$

dengan

$$S(a,b)=\frac{h}{3}\left[f(a)+4f(a+h)+f(b)\right]$$

Langkah berikutnya adalah men

$$\begin{aligned}\int_a^b f(x)dx &= \frac{h}{6}\left[f(a)+4f\left(a+\frac{h}{2}\right)+2f\left(a+h\right)+4f\left(a+\frac{3h}{2}\right)+f(b)\right] \\ &- \left(\frac{h}{2}\right)^4\frac{(b-a)}{180}f^{(4)}(\tilde{\mu}) \qquad (10.12)\end{aligned}$$

10.5 Gaussian Quadrature

Suatu integral dapat ditransformasi kedalam bentuk Gaussian quadrature melalui formulasi berikut

$$\int_a^b f(x)dx = \int_{-1}^{1} f\left(\frac{(b-a)t+(b+a)}{2}\right)\frac{(b-a)}{2}dt \tag{10.13}$$

dimana perubahan variabel memenuhi

$$t = \frac{2x-a-b}{b-a} \Leftrightarrow x = \frac{1}{2}\left[(b-a)t+a+b\right] \tag{10.14}$$

Berikut adalah table polinomial Legendre untuk penyelesaian Gaussian quadrature

Tabel 10.1: 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

10.5.1 Contoh

Selesaikan integrasi berikut ini

$$\int_1^{1,5} e^{-x^2}dx \tag{10.15}$$

(Solusi exact integral diatas adalah: 0.1093643)

jawab:

Pertama, integral tersebut ditransformasikan kedalam Gaussian quadrature melalui persamaan (10.13)

$$\int_1^{1,5} e^{-x^2}dx = \frac{1}{4}\int_{-1}^{1} e^{\frac{-(t+5)^2}{16}}dt \tag{10.16}$$

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 $n = 3$

$$\begin{aligned}\int_1^{1,5} e^{-x^2}dx &\approx \frac{1}{4}[(0,5555555556)e^{(-(0,7745966692+5)^2/16)} + (0,8888888889)e^{(-(5)^2/16)} \\ &+ (0,5555555556)e^{(-(-0,7745966692+5)^2/16)}] = 0,1093642\end{aligned}$$

10.5.2 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 11

Mencari Akar

Objektif :

▷ Mencari akar

11.1 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

$$f(x) = f(\bar{x}) + (x - \bar{x})f'(\bar{x}) + \frac{(x - \bar{x})^2}{2}f''(\xi(x))$$

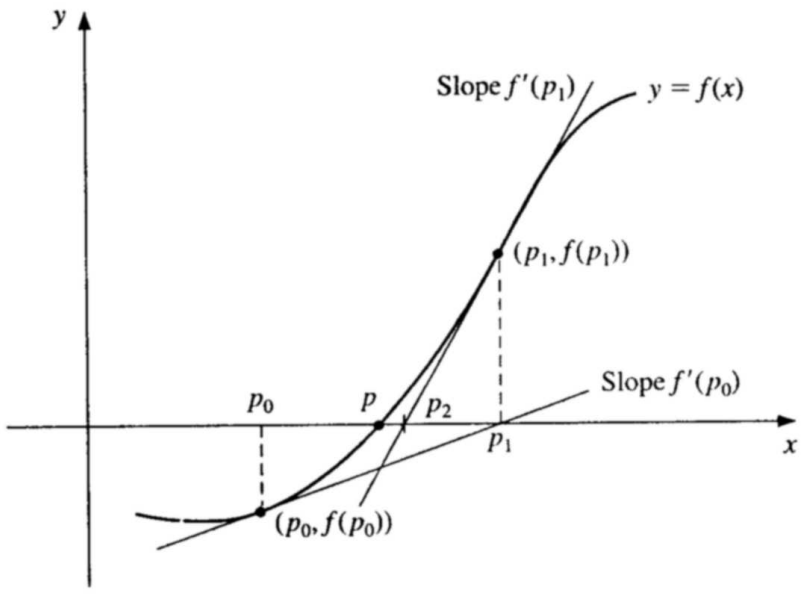
$$0 = f(\bar{x}) + (p - \bar{x})f'(\bar{x}) + \frac{(p - \bar{x})^2}{2}f''(\xi(p))$$

$$0 = f(\bar{x}) + (p - \bar{x})f'(\bar{x})$$

$$p - \bar{x} = -\frac{f(x)}{f'(\bar{x})}$$

$$p \approx \bar{x} - \frac{f(x)}{f'(\bar{x})}$$

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} \quad , \qquad n \geq 1$$



Gambar 11.1: Metode Newton

Bab 12

Metode Monte Carlo

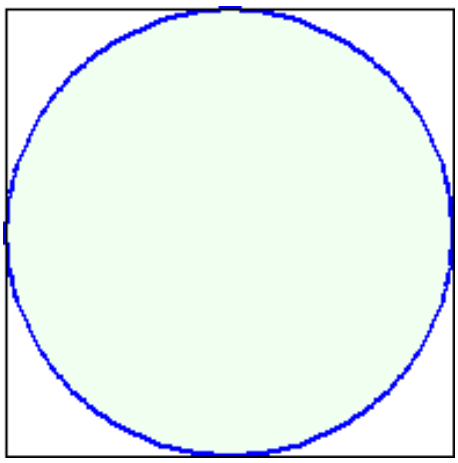
Objektif :

▷ Mengenalkan metode Monte Carlo

12.1 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

$$\rho = \frac{luas \quad lingkaran}{luas \quad bujursangkar} = \frac{\pi}{4} = 0,7853981633974483 \tag{12.1}$$



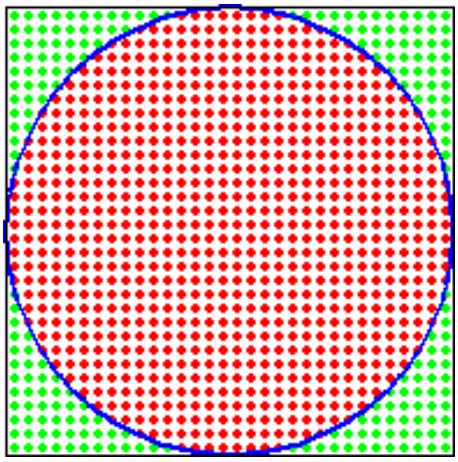
Gambar 12.1: Lingkaran dan bujursangkar

Jadi, dengan mengetahui nilai ρ, maka kita bisa menghitung luas lingkaran dengan cara

$$luas \quad lingkaran = \rho \times luas \quad bujursangkar \tag{12.2}$$

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

$$\rho = \frac{dart \quad di \quad dalam \quad lingkaran}{total \quad dart \quad di \quad dalam \quad bujursangkar} = \frac{812}{1024} = 0,79296875 \tag{12.3}$$



Gambar 12.2: Dart yang menancap pada bidang lingkaran dan bujursangkar

Dengan pendekatan ke persamaan (12.2) maka luas lingkaran adalah

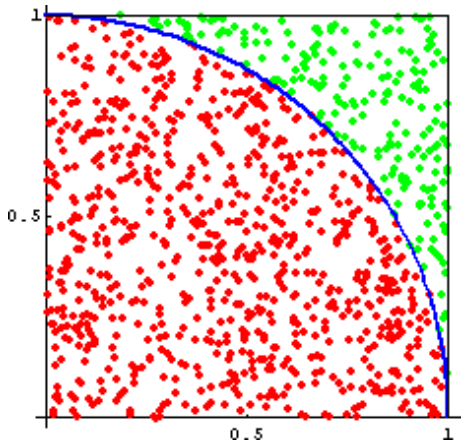
$$\begin{aligned} luas \quad lingkaran &= \rho \times luas \quad bujursangkar \\ &= 0,79296875 \times 4 \\ &= 3,171875 \end{aligned}$$

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 (12.3)

$$\rho = \frac{787}{1000} = 0,787$$

maka berdasarkan persamaan (12.1)

$$\pi = \rho \times 4 = 0,787 \times 4 = 3,148$$



Gambar 12.3: 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

$$\mathbf{R} = \{(x, y) : a \leq x \leq b \quad dan \quad 0 \leq y \leq d\} \tag{12.4}$$

dimana

$$d = maksimum \quad f(x) \quad , \quad a \leq x \leq b \tag{12.5}$$

Bab 13

Inversi

Objektif :

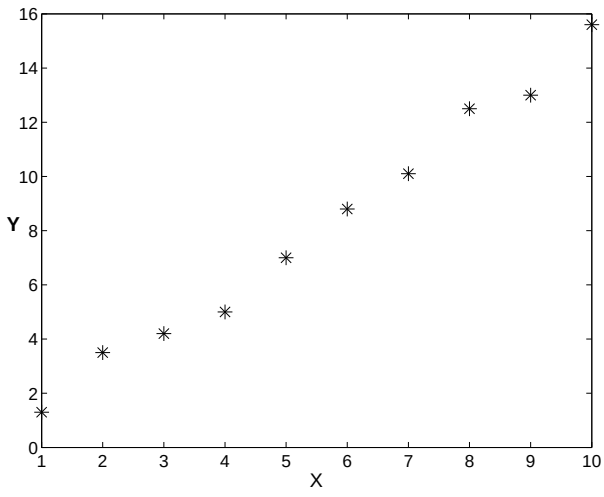
- ▷ Mengenalkan inversi linear
- ▷ Mengenalkan inversi non-linear

13.1 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

$$P(x_i) = a_1 x_i + a_0 \tag{13.1}$$

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

$$\sum_{i=1}^{m} y_i - P(x_i) = 0 \tag{13.2}$$

$$\sum_{i=1}^{m} y_i - (a_1 x_i + a_0) = 0 \tag{13.3}$$

dimana jumlah data, $m = 10$. Suku yang berada disebelah kiri dinamakan fungsi error (*error function*), yaitu

$$E(a_0, a_1) = \sum_{i=1}^{m} y_i - (a_1 x_i + a_0) \tag{13.4}$$

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

$$E(a_0, a_1) = \sum_{i=1}^{m} [y_i - (a_1 x_i + a_0)]^2 \tag{13.5}$$

Agar fungsi error bisa mencapai nilai minimum, maka syarat yang harus dipenuhi adalah:

$$\frac{\partial E(a_0, a_1)}{\partial a_i} = 0 \tag{13.6}$$

dimana $i = 0$ dan 1, karena dalam kasus ini memang cuma ada a_0 dan a_1. Maka mesti ada dua buah turunan yaitu:

$$\begin{aligned} \frac{\partial E(a_0, a_1)}{\partial a_0} = \frac{\partial}{\partial a_0} \sum_{i=1}^{m} [y_i - (a_1 x_i + a_0)]^2 &= 0 \\ 2 \sum_{i=1}^{m} (y_i - a_1 x_i - a_0)(-1) &= 0 \\ a_0 . m + a_1 \sum_{i=1}^{m} x_i &= \sum_{i=1}^{m} y_i \end{aligned} \tag{13.7}$$

dan

$$\begin{aligned}\frac{\partial E(a_0,a_1)}{\partial a_1} = \frac{\partial}{\partial a_1}\sum_{i=1}^{m}[y_i-(a_1x_i+a_0)]^2 &= 0\\ 2\sum_{i=1}^{m}(y_i-a_1x_i-a_0)(-x_i) &= 0\\ a_0\sum_{i=1}^{m}x_i+a_1\sum_{i=1}^{m}x_i^2 &= \sum_{i=1}^{m}x_iy_i \end{aligned} \tag{13.8}$$

Akhirnya persamaan (13.7) dan (13.8) dapat dicari solusinya berikut ini:

$$a_0 = \frac{\sum_{i=1}^{m}x_i^2\sum_{i=1}^{m}y_i-\sum_{i=1}^{m}x_iy_i\sum_{i=1}^{m}x_i}{m\left(\sum_{i=1}^{m}x_i^2\right)-\left(\sum_{i=1}^{m}x_i\right)^2} \tag{13.9}$$

dan

$$a_1 = \frac{m\sum_{i=1}^{m}x_iy_i-\sum_{i=1}^{m}x_i\sum_{i=1}^{m}y_i}{m\left(\sum_{i=1}^{m}x_i^2\right)-\left(\sum_{i=1}^{m}x_i\right)^2} \tag{13.10}$$

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:

$$a_0 = \frac{385(81)-55(572,4)}{10(385)-(55)^2} = -0,360 \tag{13.11}$$

dan

$$a_1 = \frac{10(572,4)-55(81)}{10(385)-(55)^2} = 1,538 \tag{13.12}$$

Jadi, fungsi pendekatan-nya, $P(x_i)$, adalah

$$P(x_i) = 1,538x_i - 0,360 \tag{13.13}$$

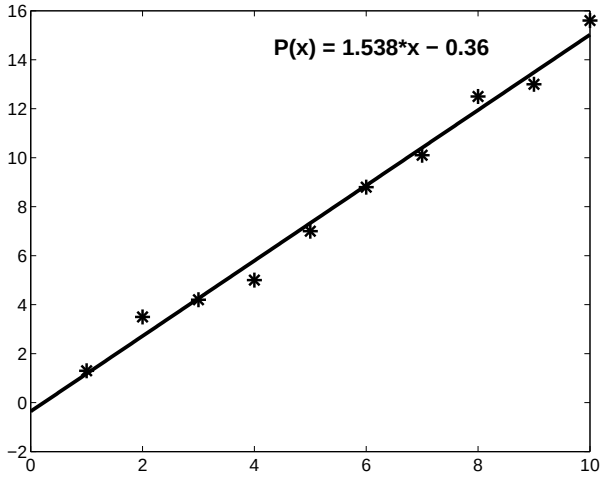
Solusi least square dengan pendekatan persamaan garis seperti ini juga dikenal dengan nama lain yaitu **regresi linear**. Sedangkan nilai a_0 dan a_1 disebut **koefisien regresi**. Gambar di bawah ini menampilkan solusi regresi linear tersebut berikut semua titik datanya

Tentu saja anda sudah bisa menduga bahwa selain regresi linear, mungkin saja terdapat regresi parabola atau quadratik dimana fungsi pendekatannya berupa persamaan parabola, yaitu:

$$P(x_i) = a_2x_i^2 + a_1x_i + a_0 \tag{13.14}$$

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:

$$P(x_i) = a_nx_i^n + a_{n-1}x_i^{n-1} + ... + a_2x_i^2 + a_1x_i + a_0 \tag{13.15}$$



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 (13.7) dan (13.8), 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:

$$\begin{bmatrix} m & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & \sum_{i=1}^{m} x_i^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} y_i \\ \sum_{i=1}^{m} x_i y_i \end{bmatrix} \tag{13.16}$$

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!

13.2 Inversi Non-Linear

Persamaan least squares linear adalah sebagai berikut:

$$[\mathbf{G}^t\mathbf{G}]\delta m = \mathbf{G}^t\delta d \tag{13.17}$$

Persamaan least squares non-linear dapat dinyatakan sebagai berikut:

$$[\mathbf{G}^t\mathbf{G} + \lambda\mathbf{I}]\delta m = \mathbf{G}^t\delta d \tag{13.18}$$

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

$$A = \frac{\partial f(m)}{\partial m} = x^m ln(x) \tag{13.19}$$

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

$$||\delta d|| = \frac{1}{N}\Sigma(d^c - d)^2 = \frac{1}{N}\Sigma\delta d^2 \tag{13.20}$$

   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

$$[\mathbf{G}^t\mathbf{G} + \lambda\mathbf{I}]\delta m = \mathbf{G}^t\delta d \tag{13.21}$$

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

$$m^1 = m^0 + \delta m \tag{13.22}$$

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