Metode Numerik Menggunakan R Untuk Teknik Lingkungan

Mohammad Rosidi

2019-08-27

Contents

Kata Pengantar

Chapter 1

Bahasa Pemrograman R

Dewasa ini tersedia banyak sekali *software* yang dapat digunakan untuk membantu kita dalam melakukan analisa data. *software* yang digunakan dapat berupa *software* berbayar atau gratis.

`R` merupakan merupakan salah satu *software* gratis yang sangat populer di Indonesia. Kemudahan penggunaan serta banyaknya besarnya dukungan komunitas membuat `R` menjadi salah satu bahasa pemrograman paling populer di dunia.

Paket yang disediakan untuk analisis statistika dan analisa numerik juga sangat lengkap dan terus bertambah setiap saat. Hal ini membuat `R` banyak digunakan oleh para analis data.

Pada *chapter* ini penulis akan memperkenalkan kepada pembaca mengenai bahasa pemrograman `R`. Mulai dari sejarah, cara instalasi sampai dengan bagaimana kita memanfaatkan fitur dasar bantuan untuk menggali lebih jauh tentang fungsi-fungsi `R`

1.1 Sejarah R

`R` Merupakan bahasa yang digunakan dalam komputasi **statistik** yang pertama kali dikembangkan oleh **Ross Ihaka** dan **Robert Gentlement** di University of Auckland New Zealand yang merupakan akronim dari nama depan kedua pembuatnya. Sebelum `R` dikenal ada `S` yang dikembangkan oleh **John Chambers** dan rekan-rekan dari **Bell Laboratories** yang memiliki fungsi yang sama untuk komputasi statistik. Hal yang membedakan antara keduanya adalah `R` merupakan sistem komputasi yang bersifat gratis.Logo `R` dapat dilihat pada Gambar 1.1.

Figure 1.1: Logo R.

`R` dapat dibilang merupakan aplikasi sistem **statistik** yang kaya. Hal ini disebabkan banyak sekali paket yang dikembangkan oleh pengembang dan komunitas untuk keperluan analisa statistik seperti *linear regression*, *clustering*, *statistical test*, dll. Selain itu, `R` juga dapat ditambahkan paket-paket lain yang dapat meningkatkan fiturnya.

Sebagai sebuah bahasa pemrograman yang banyak digunakan untuk keperluan analisa data, `R` dapat dioperasikan pada berbagai sistem operasi pada komputer. Adapun sistem operasi yang didukung antara lain: `UNIX`, `Linux`, `Windows`, dan `MacOS`.

1.2 Fitur dan Karakteristik R

`R` memiliki karakteristik yang berbeda dengan bahasa pemrograman lain seperti `C++`,`python`, dll. `R` memiliki aturan/sintaks yang berbeda dengan bahasa pemrograman yang lain yang membuatnya memiliki ciri khas tersendiri dibanding bahasa pemrograman yang lain.

Beberapa ciri dan fitur pada `R` antara lain:

1. **Bahasa `R` bersifat case sensitif**. maksudnya adalah dalam proses input `R` huruf besar dan kecil sangat diperhatikan. Sebagai contoh kita ingin melihat apakah objek A dan B pada sintaks berikut:

```
A <- "Andi"
B <- "andi"

# cek kedua objek A dan B
A == B
```

```
## [1] FALSE
```

```
# Kesimpulan : Kedua objek berbeda
```

2. **Segala sesuatu yang ada pada program `R` akan diangap sebagai objek**. konsep objek ini sama dengan bahasa pemrograma berbasis objek yang lain seperti `Java`, `C++`, `python`, dll.Perbedaannya adalah bahasa `R` relatif lebih sederhana dibandingkan bahasa pemrograman berbasis obejk yang lain.
3. **interpreted language atau script**. Bahasa `R` memungkinkan pengguna untuk melakukan kerja pada `R` tanpa perlu kompilasi kode program menjadi bahasa mesin.
4. Mendukung proses **loop**, **decision making**, dan menyediakan berbagai jenis **operstor** (aritmatika, logika, dll).
5. **Mendukung export dan import berbagai format file**, seperti:TXT, CSV, XLS, dll.
6. **Mudah ditingkatkan melalui penambahan fungsi atau paket**. Penambahan paket dapat dilakukan secara online melalui CRAN atau melalui sumber seperti github.
7. **Menyedikan berbagai fungsi untuk keperluan visualisasi data**. Visualisasi data pada `R` dapat menggunakan paket bawaan atau paket lain seperti `ggplo2`,`ggvis`, dll.

1.3 Kelebihan dan Kekurangan R

Selain karena `R` dapat digunakan secara gratis terdapat **kelebihan** lain yang ditawarkan, antara lain:

1. **Protability**. Penggunaan software dapat digunakan kapanpun tanpa terikat oleh masa berakhirnya lisensi.
2. **Multiplatform**. `R` bersifat *Multiplatform Operating Systems*, dimana *software* `R` lebih kompatibel dibanding *software* statistika lainnya. Hal in berdampak pada kemudahan dalam penyesuaian jika pengguna harus berpindah sistem operasi karena `R` baik pada sistem operasi seperti `windows` akan sama pengoperasiannya dengan yang ada di `Linux` (paket yang digunakan sama).
3. **General** dan **Cutting-edge**. Berbagai metode statistik baik metode klasik maupun baru telah diprogram kedalam `R`. Dengan demikian *software* ini dapat digunakan untuk analisis statistika dengan pendekatan klasik dan pendekatan modern.
4. **Programable**. Pengguna dapat memprogram metode baru atau mengembangakan modifikasi dari analisis statistika yang telah ada pada sistem `R`.
5. **Berbasis analisis matriks**. Bahasa `R` sangat baik digunakan untuk *programming* dengan basis matriks.

6. Fasiltas grafik yang lengkap.

Adapun kekurangan dari `R` antara lain:

1. **Point and Click GUI**. Interaksi utama dengan `R` bersifat *CLI* (*Command Line Interface*), walaupun saat ini telah dikembangkan paket yang memungkinkan kita berinteraksi dengan `R` menggunakan *GUI* (*Graphical User Interface*) sederhana menggunakan paket `R-Commander` yang memiliki fungsi yang terbatas. `R- Commander` sendiri merupakan *GUI* yang diciptakan dengan tujuan untuk keperluan pengajaran sehingga analisis statistik yang disediakan adalah yang klasik. Meskipun terbatas paket ini berguna jika kita membutuhkan analisis statistik sederhana dengan cara yang simpel.
2. **Missing statistical function**. Meskipun analisis statistika dalam `R` sudah cukup lengkap, namun tidak semua metode statistika telah diimplementasikan ke dalam `R`. Namun karena `R` merupakan *lingua franca* untuk keperluan komputasi statistika modern staan ini, dapat dikatakan ketersediaan fungsi tambahan dalam bentuk paket hanya masalah waktu saja.

1.4 RStudio

Aplikasi `R` pada dasarnya berbasis teks atau *command line* sehingga pengguna harus mengetikkan perintah-perintah tertentu dan harus hapal perintah-perintahnya. Setidaknya jika kita ingin melakukan kegiatan analisa data menggunakan `R` kita harus selalu siap dengan perintah-perintah yang hendak digunakan sehingga buku manual menjadi sesuatu yang wajib adasaat berkeja dengan `R`.

Kondisi ini sering kali membingunkan bagi pengguna pemula maupun pengguna mahir yang sudah terbiasa dengan aplikasi statistik lain seperti SAS, SPSS, Minitab, dll. Alasan itulah yang menyebabkan pengembang `R` membuat berbagai *frontend* untuk `R` yang berguna untuk memudahkan dalam pengoperasian `R`.

`RStudio` merupakan salah satu bentuk *frontend* `R` yang cukup populer dan nyaman digunakan. Selain nyaman digunakan, `RStudio` memungkinkan kita melakukan penulisan laporan menggunakan `Rmarkdown` atau `RNotebook` serta membuat berbagai bentuk project seperti shyni, dll. Pada `R` studio juga memungkinkan kita mengatur *working directory* tanpa perlu mengetikkan sintaks pada Commander, yang diperlukan hanya memilihnya di menu `RStudio`. Selain itu, kita juga dapat meng-import file berisikan data tanpa perlu mengetikkan pada Commander dengan cara memilih pada menu `Environment`.

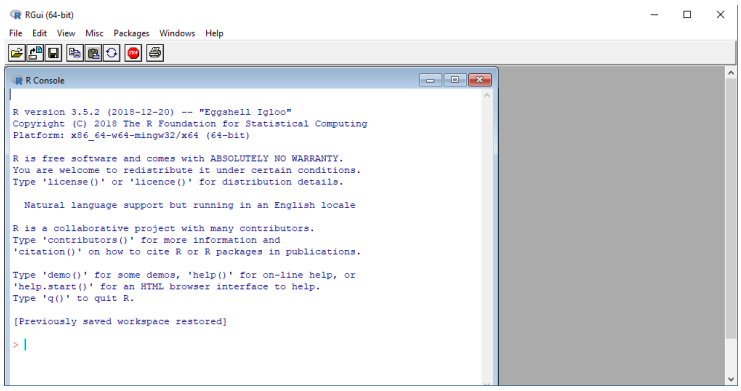

Figure 1.2: Jendela R.

1.5 Menginstall R dan RStudio

Pada tutorial ini hanya akan dijelaskan bagaimana menginstal `R` dan `RStudio` pada sistem operasi `windows`. Sebelum memulai menginstal sebaiknya pembaca mengunduh terlebih dahulu *installer* R dan RStudio.

1. Jalankan proses pemasangan dengan meng-klik *installer* aplikasi `R` dan `RStudio`.
2. Ikuti langkah proses pemasangan aplikasi yang ditampilkan dengan klik `OK` atau `Next`.
3. Apabila pemasangan telah dilakukan, jalankan aplikasi yang telah terpasang untuk menguji jika aplikasi telah berjalan dengan baik.

Jendela aplikasi yang telah terpasang ditampilkan pada Gambar 1.2 dan Gambar 1.3.

Tips: Sebaiknya install `R` terlebih dahulu sebelum `RStudio`

1.6 Working Directory

Setiap pengguna akan bekerja pada tempat khusus yang disebut sebagai *working directory*. *working directory* merupakan sebuah folder dimana `R` akan membaca dan menyimpan file kerja kita. Pada pengguna `windows`, *working directory* secara default pada saat pertama kali menginstall `R` terletak pada folder `c:\\Document`.

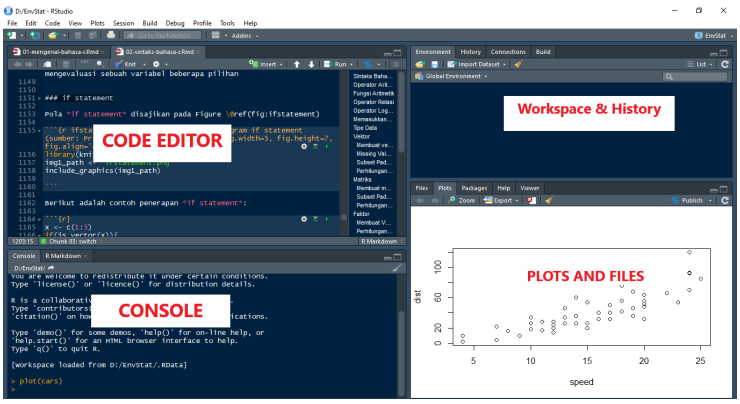

Figure 1.3: Jendela RStudio.

1.6.1 Mengubah Lokasi Working Directory

Kita dapat mengubah lokasi *working directory* berdasarkan lokasi yang kita inginkan, misalnya letak data yang akan kita olah tidak ada pada folder default atau kita ingin pekerjaan kita terkait `R` dapat berlangsung pada satu folder khusus.

Berikut adalah cara mengubah *working directory* pada `R`.

1. Buatlah folder pada drive (kita bisa membuat folder pada selain drive c) dan namai dengan nama yang kalian inginkan. Pada tutorial ini penulis menggunakan nama folder `R`.
2. Jika pengguna menggunakan `RStudio`, pada menu `RStudio` pilih **Session > Set Working Directory > Chooses Directory**. Proses tersebut ditampilkan pada Gambar 1.4
3. Pilih folder yang telah dibuat pada step 1 sebagai *working directory.

> **Penting:** Data atau file yang hendak dibaca selama proses kerja pada `R` harus selalu diletakkan pada working directory. Jika tidak maka data atau file tidak akan terbaca.

Untuk mengecek apakah proses perubahan telah terjadi, kita dapat mengeceknya dengan menjalankan perintah berikut untuk melihat lokasi *working directory* kita yang baru.

```
getwd()
```

Selain itu kita dapat mengubah *working directory* menggunakan perintah berikut:

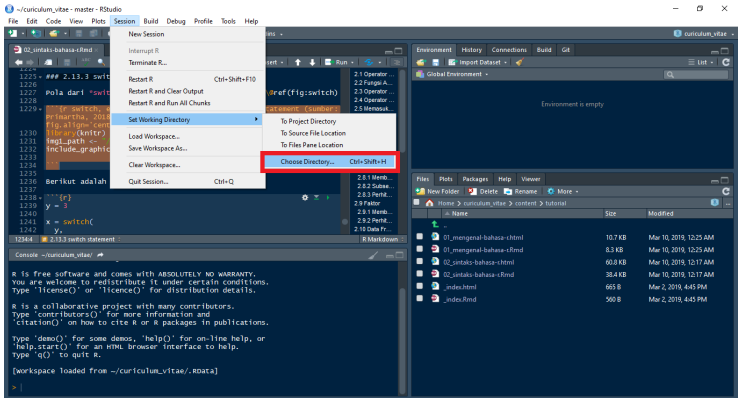

Figure 1.4: Mengubah working directory.

```
# Ubah working directori pada folder R
setwd("/Documents/R")
```

Peringatan !!!

Pada proses pengisian lokasi folder pastikan pemisah pada lokasi folder menggunakan tanda "/" bukan "\"

1.6.2 Mengubah Lokasi Working Directory Default

Pada proses yang telah penulis jelaskan sebelumnya. Proses perubahan *working directory* hanya berlaku pada saat pekerjaan tersebut dilakukan. Setelah pekerjaan selesai dan kita menjalankan kembali `R` maka *working directory* akan kembali secara default pada working directory lama.

Untuk membuat lokasi default *working directory* pindah, kita dapat melakukannya dengan memilih pada menu: **Tools > Global options > pada "General" klik pada "Browse" dan pilih lokasi working directory yang diinginkan**. Proses tersebut ditampilkan pada Gambar 1.5

1.7 Memasang dan Mengaktifkan Paket R

`R` dapat ditingkatkan fungsionalitasnya melalui paket-paket yang tersedia secara luas. Paket-paket ini dikembangkan secara spesifik oleh para pengembang sesuai dengan tujuan paketnya, seperti: `tidyverse` untuk *data science*, `pracma` untuk analisis diferensial, dll.

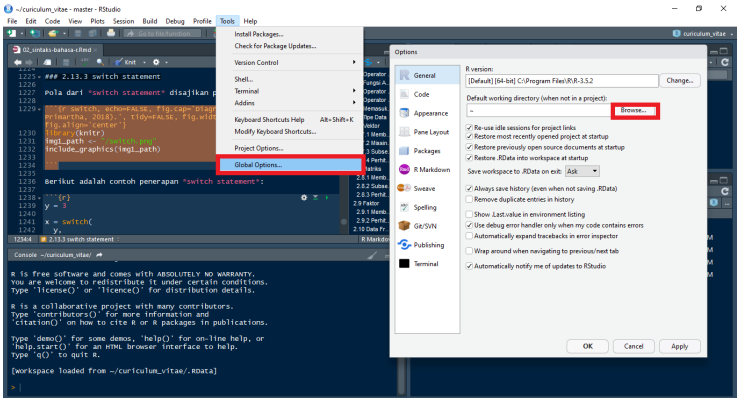

Figure 1.5: Merubah working directory melalui Global options.

Untuk menginstall paket yang kita inginkan, kita dapat menggunakan fungsi `install.packages()`. Berikut adalah contoh bagaimana cara menginstall paket `tidyverse`:

```
install.packages("tidyverse")
```

Paket yang telah diinstall tidak dapat langsung digunakan. Untuk menggunakan fungsi-fungsi yang tersedia pada paket tersebut kita perlu terlebih dahulu mengaktifkannya menggunakan fungsi `library()`. Berikut adalah contoh sintaks untuk mengaktifkan paket `tidyverse`:

```
library(tidyverse)
```

Bagaimana ingin menggunakan fungsi pada paket namun tidak ingin mengaktifkan paketnya terlebih dahulu menggunakan fungsi `library()`? Untuk melakukannya kita perlu mengetikkan nama paket dikuti oleh tanda "::" diikuti fungsi yang ingin kita gunakan. Berikut adalah contoh penggunaan fungsi `read_csv()` dari paket `readr` (salah satu paket yang terdapat pada kumpulan paket `tidyverse`) untuk membaca file `contoh.csv`:

```
readr::read_csv("contoh.csv")
```

1.8 Fasilitas Help

Agar dapat menggunakan `R` dengan secara lebih baik, pengetahuan untuk mengakses fasilitas *help* in cukup penting untuk disampaikan. Adapun cara yang dapat digunakan adalah sebagai berikut.

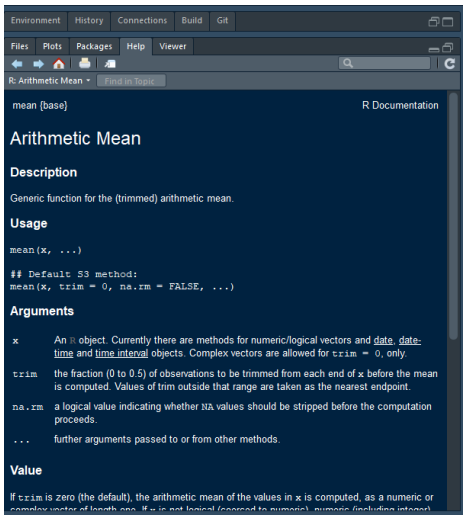

Figure 1.6: Jendela help dokumentasi fungsi mean().

1.8.1 Mencari Help dari Suatu Perintah Tertentu

Untuk memperoleh bantuan terkait suatu perintah tertentu kita dapat menggunakan fungsi `help()`. Secara umum format yang digunakan adalah sebagai berikut:

```
help(nama_perintah)
```

atau dapat juga menggunakan tanda tanya (?) pada awal `nama_perintah` seperti berikut:

```
?nama_perintah
```

Misalkan kita kebingungan terkait bagaimana cara menuliskan perintah untuk menghitung rata-rata suatu vektor. Kita dapat mengetikkan perintah berikut untuk mengakses fasilitas *help*.

```
help(mean)

#atau
?mean
```

Perintah tersebut akan memunculkan hasil berupa dokumentasi yang ditampilkan pada Gambar 1.6.

Keterangan pada jendela pada Gambar 1.6 adalah sebagia berikut:

1. Pada bagian jendela kiri atas jendela *help*, diberikan keterangan nama dari perintah yang sedang ditampilkan.
2. Selanjutnya, pada bagian atas dokumen, ditampilkan infomasi terkait nama perintah, dan nama *library* yang memuat perintah tersebut. Pada gambar diatas informasi terkait perintah dan nama *library* ditunjukkan pada teks `mean {base}` yang menunjukkan perintah `mean()` pada paket (*library*) *base* (paket bawaan `R`).
3. Setiap jendela *help* dari suatu perintah tertentu selanjutnya akan memuat bagian-bagian berikut:

- *Title*
- *Description* : deskripsi singkat tentang perintah.
- *Usage* : menampilkan sintaks perintah untuk penggunaan perintah tersebut.
- *Arguments* : keterangan mengenai *argument/input*yang diperlukan pada perintah tersebut.
- *Details* : keterangan lebih lengkap lengkap tentang perintah tersebut.
- *Value* : keterangan tentang *output* suatu perintah dapat diperoleh pada bagian ini.
- *Author(s)* : memberikan keterangan tentang *Author* dari perintah tersebut.
- *References* : seringkali referensi yang dapat digunakan untuk memperoleh keterangan lebih lanjut terhadap suatu perintah ditampilkan pada bagian ini.
- *See also*: bagian ini berisikan daftar perintah/fungsi yang berhubungan erat dengan perintah tersebut.
- *Example* : berisikan contoh-contoh penggunaan perintah tersebut.

Kita juga dapat melihat contoh penggunaan dari perintah tersebut. Untuk melakukannya kita dapat menggunakan fungsi `example()`. Fungsi tersebut akan menampilkan contoh kode penerapan dari fungsi yang kita inginkan. Secara sederhana fungsi tersebut dapat dituliskan sebagai berikut:

```
example(nama_perintah)
```

Untuk mengetahui contoh kode fungsi `mean()`, ketikkan sintaks berikut:

```
example(mean)
```

```
##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50
```

kita juga dapat mencoba kode yang dihasilkan pada console R. Berikut adalah contoh penerapannya:

```
# Menghitung rata-rata bilangan 1 sampai 10 dan 50
# membuat vektor
x <- c(0:10, 50)

# Print
x
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 50
```

```
# mean
mean(x)
```

```
## [1] 8.75
```

Pembaca dapat mencoba melakukanya sendiri dengan mengganti nilai yang telah ada serta mencoba contoh kode yang lain.

1.8.2 General Help

Kita juga dapat membaca beberapa dokumen manual yang ada pada R. Untuk melakukannya jalankan perintah berikut:

```
help.start()
```

Output yang dihasilkan berupa link pada sejumlah dokumen yang dapat kita klik. Tampilan halaman yang dihasilkan disajikan pada Gambar 1.7.

1.8.3 Fasilitas Help Lainnya

Selain yang telah penulis sebutkan sebelumnya. Kita juga dapat memanfaatkan fasilitas *help* lainnya melalui fungsi `apropos()` dan `help.search()`.

`apropos ()`: mengembalikan daftar objek, berisi pola yang pembaca cari, dengan pencocokan sebagian. Ini berguna ketika pembaca tidak ingat persis nama fungsi yang akan digunakan. Berikut adalah contoh ketika penulis ingin mengetahui fungsi yang digunakan untuk menghitung median.

```
apropos("med")
```

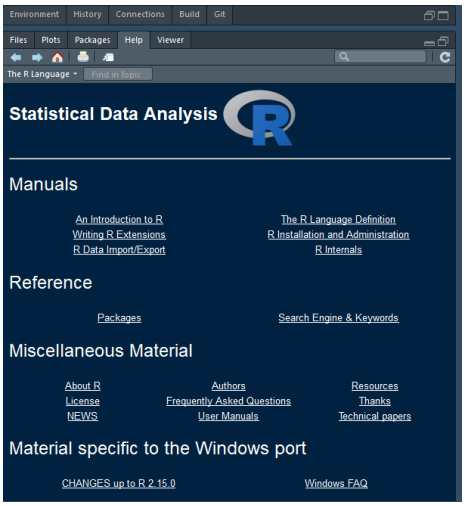

Figure 1.7: Jendela general help dokumentasi fungsi mean().

```
## [1] "elNamed"        "elNamed<-"      "median"         "median.default"
## [5] "medpolish"      "runmed"
```

List yang dihasilkan berupa fungsi-fungsi yang memiliki elemen kata "med". Berdasarkan pencaria tersebut penulis dapat mencoba menggunakan fungsi "median" untuk menghitung median.

`help.search ()` (sebagai alternatif ??): mencari dokumentasi yang cocok dengan karakter yang diberikan dengan cara yang berbeda. Ini mengembalikan daftar fungsi yang mengandung istilah yang pembaca cari dengan deskripsi singkat dari fungsi.

Berikut adalah contoh penerapan dari fungsi tersebut:

```
help.search("mean")

# atau
??mean
```

Output yang dihasilkan akan tampak seperti pada Gambar 1.8.

1.9 Referensi

1. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung

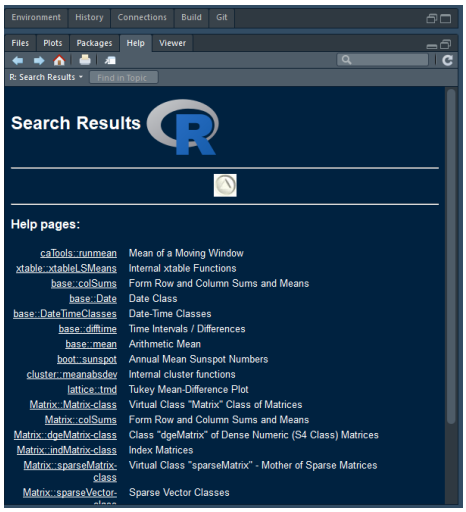

Figure 1.8: Jendela help search dokumentasi fungsi mean().

2. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta
3. STHDA. Running RStudio and Setting Up Your Working Directory - Easy R Programming .http://www.sthda.com/english/wiki/running-rstudio-and-setting-up-your-working-directory-easy-r-programming#set-your-working-directory
4. STDHA. **Getting Help With Functions In R Programming**. http://www.sthda.com/english/wiki/getting-help-with-functions-in-r-programming .
5. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.

Chapter 2

Kalkulasi Menggunakan R

Pada *Chapter* ini penulis akan menjelaskan bagaimana melakukan perhitungan menggunakan `R`. Hal-hal yang akan dibahas pada *chapter* ini antara lain:

- Operator dan fungsi dasar pada `R`
- Jenis dan struktur data
- Vektor (cara membuat dan melakukan operasi matematika pada vektor)
- Matriks (cara membuat dan melakukan operasi matematika pada matriks)

2.1 Operator Aritmatik

Proses perhitungan akan ditangani oleh fungsi khusus. `R` akan memahami urutannya secara benar. Kecuali kita secara eksplisit menetapkan yang lain. Sebagai contoh jalankan sintaks berikut:

```
2+4*2
```

```
## [1] 10
```

Bandingkan dengan sintaks berikut:

```
(2+4)*2
```

```
## [1] 12
```

> **Tips:** `R` dapat digunakan sebagai kalkulator

Berdasarkan kedua hasil tersebut dapat disimpulkan bahwa ketika kita tidak menetapkan urutan perhitungan menggunakan tanda kurung, R akan secara otomatis akan menghitung terlebih dahulu perkalian atau pembangian.

Operator aritmatika yang disediakan R disajikan pada Tabel 2.1:

Table 2.1: Operator Aritmatika R.

Simbol	**Keterangan**
+	*Addition*, untuk operasi penjumlahan
-	*Substraction*, untuk operasi pengurangan
*	*Multiplication*, untuk operasi pembagian
/	*Division*, untuk operasi pembagian
^	*Eksponentiation*, untuk operasi pemangkatan
%%	*Modulus*, Untuk mencari sisa pembagian
%/%	*Integer*, Untuk mencari bilangan bulat hasil pembagian saja dan tanpa sisa pembagian

Untuk lebih memahaminya berikut contoh sintaks penerapan operator tersebut.

```
# Addition
5+3
```

```
## [1] 8
```

```
# Substraction
5-3
```

```
## [1] 2
```

```
# Multiplication
5*3
```

```
## [1] 15
```

```
# Division
5/3
```

```
## [1] 1.666667
```

```
# Eksponetiation
5^3
```

```
## [1] 125
```

```
# Modulus
5%%3
```

```
## [1] 2
```

```
# Integer
5%/%3
```

```
## [1] 1
```

> **Tips:** Pada R tanda # berfungsi menambahkan keterangan untuk menjelaskan sebuah sintaks pada R

2.2 Fungsi Aritmetik

Selain fungsi operator aritmetik, pada R juga telah tersedia fungsi aritmetik yang lain seperti logaritmik, ekponensial, trigonometri, dll.

1. Logaritma dan eksponensial

Untuk contoh fungsi logaritmik dan eksponensial jalankan sintaks berikut:

```
log2(8) # logaritma basis 2 untuk 8
```

```
## [1] 3
```

```
log10(8) # logaritma basis 10 untuk 8
```

```
## [1] 0.90309
```

```
exp(8) # eksponensial 8
```

```
## [1] 2980.958
```

2. Fungsi trigonometri

fungsi trigonometri yang ditampilkan seperti sin,cos, tan, dll.

```
cos(x) # cos x
sin(x) # Sin x
tan(x) # Tan x
acos(x) # arc-cos x
asin(x) # arc-sin x
atan(x) #arc-tan x
```

Penting!!!

x dalam fungsi trigonometri memiliki satuan radian

Berikut adalah salah satu contoh penggunaannya:

```
cos(pi)
```

```
## [1] -1
```

Pada paket `pracma` fungsi-fungsi trigonometri dapat ditambah lagi. Fungsi-fungsi tersebut antara lain:

```
cot(x) # cotan x
csc(x) # cosecan x
sec(x) # secan x
acot(x) # arc-cotan x
acsc(x) # arc-cosecan x
asec(x) # arc-secan x
```

3. Fungsi Hiperbolik

fungsi hiperbolik yang tersedia antara lain:

```
cosh(x)
sinh(x)
tanh(x)
acosh(x)
asinh(x)
atanh(x)
```

Fungsi tersebut dapat ditambah lagi dari paket `pracma`. Fungsi-fungsi yang tersedia antara lain:

```
coth(x)
csch(x)
sech(x)
acoth(x)
acsch(x)
asech(x)
```

4. Fungsi matematik lainnya

Fungsi lainnya yang dapat digunakan adalah fungsi absolut, akar kuadrat, dll. Berikut adalah contoh sintaks penggunaan fungsi absolut dan akar kuadrat.

```
abs(-2) # nilai absolut -2
```

```
## [1] 2
```

```
sqrt(4) # akar kuadrat 4
```

```
## [1] 2
```

2.3 Operator Relasi

Operator relasi digunakan untuk membandingkan satu objek dengan objek lainnya. Operator yang disediakan R disajikan pada Tabel 2.2.

Table 2.2: Operator Relasi R.

Simbol	**Keterangan**
“>”	Lebih besar dari
“<”	Lebih Kecil dari
“==”	Sama dengan
“>=”	Lebih besar sama dengan
“<=”	Lebih kecil sama dengan
“!=”	Tidak sama dengan

Berikut adalah penerapan operator pada tabel tersebut:

```
x <- 34
y <- 35
```

```
# Operator >
x > y
```

```
## [1] FALSE
```

```
# Operator <
x < y
```

```
## [1] TRUE
```

```
# operator ==
x == y
```

```
## [1] FALSE
```

```
# Operator >=
x >= y
```

```
## [1] FALSE
```

```
# Operator <=
x <= y
```

```
## [1] TRUE
```

```
# Operator !=
x != y
```

```
## [1] TRUE
```

2.4 Operator Logika

Operator logika hanya berlaku pada vektor dengan tipe logical, numeric, atau complex. Semua angka bernilai 1 akan dianggap bernilai logika `TRUE`. Operator logika yang disediakan `R` dapat dilihat pada Tabel 2.3.

Table 2.3: Operator logika R.

Simbol	Keterangan
"&&"	Operator logika AND
"!"	Opeartor logika NOT
"&"	Operator logika AND element wise

Penerapannya terdapat pada sintaks berikut:

```
v <- c(TRUE,TRUE, FALSE)
t <- c(FALSE,FALSE,FALSE)

# Operator &&
print(v&&t)
```

```
## [1] FALSE
```

```
# Operator ||
print(v||t)
```

```
## [1] TRUE
```

```
# Operator !
print(!v)
```

```
## [1] FALSE FALSE  TRUE
```

```
# operator &
print(v&t)
```

```
## [1] FALSE FALSE FALSE
```

```
# Operator |
print(v|t)
```

```
## [1]  TRUE  TRUE FALSE
```

Penting!!!

- operator `&` dan `|` akan mengecek logika tiap elemen pada vektor secara berpesangan (sesuai urutan dari kiri ke kanan). Operator `%%` dan `||` hanya mengecek dari kiri ke kanan pada
- observasi pertama. Misal saat menggunakan && jika observasi pertama `TRUE` maka observasi pertama pada vektor lainnya akan dicek, namun jika observasi pertama `FALSE` maka proses akan segera dihentikan dan menghasilkan FALSE.

2.5 Memasukkan Nilai Kedalam Variabel

Variabel pada `R` dapat digunakan untuk menyimpan nilai. Sebagai contoh jalankan sintaks berikut:

```
# Harga sebuah lemon adalah 500 rupiah
lemon <- 500

# Atau
500 -> lemon

# dapat juga menggunakan tanda "="
lemon = 500
```

Penting!!!

1. `R` memungkinkan penggunaan `<-`,`->`, atau `=` sebagai perintah pengisi nilai variabel
2. `R` bersifat *case-sensitive*. Maksudnya adalah variabel Lemon tidak sama dengan lemon (Besar kecil huruf berpengaruh)

Untuk mengetahui nilai dari objek `lemon` kita dapat menggunakan fungsi `print()` atau mengetikkan nama objeknya secara langsung.

```
# Menggunakan fungsi print()
print(lemon)
```

```
## [1] 500
```

```
# Atau
lemon
```

```
## [1] 500
```

R akan menyimpan variabel lemon sebagai objek pada memori. Sehingga kita dapat melakukan operasi terhadap objek tersebut seperti mengalikannya atau menjumlahkannya dengan bilangan lain. Sebagai contoh jalankan sintaks berikut:

```
# Operasi perkalian terhadap objek lemon
5*lemon
```

```
## [1] 2500
```

Kita dapat juga mengubah nilai dari objek lemon dengan cara menginput nilai baru terhadap objek yang sama. R secara otomatis akan menggatikan nilai sebelumnya. Untuk lebih memahaminya jalankan sintaks berikut:

```
lemon <- 1000

# Print lemon
print(lemon)
```

```
## [1] 1000
```

Untuk lebih memahaminya berikut adalah sintaks untuk menghitung volume suatu objek.

```
# Dimensi objek
panjang <- 10
lebar <- 5
tinggi <- 5

# Menghitung volume
volume <- panjang*lebar*tinggi

# Print objek volume
print(volume)
```

```
## [1] 250
```

Untuk mengetahui objek apa saja yang telah kita buat sepanjang artikel ini kita dapang menggunakan fungsi ls().

```
ls()
```

```
##  [1] "A"         "B"         "img1_path" "lebar"     "lemon"
##  [6] "panjang"   "t"         "tinggi"    "v"         "volume"
## [11] "x"         "xm"        "y"
```

> **Catatan:** Kumpulan objek yang telah tersimpan dalam memori disebut sebagai **workspace**

Untuk menghapus objek pada memori kita dapat menggunakan fungsi `rm()`. Pada sintaks berikut penulis hendak menghapus objek `lemon` dan `volume`.

```
# Menghapus objek lemon dan volume
rm(lemon, volume)

# Tampilkan kembali objek yang tersisa
ls()
```

```
##  [1] "A"         "B"         "img1_path" "lebar"     "panjang"
##  [6] "t"         "tinggi"    "v"         "x"         "xm"
## [11] "y"
```

> **Tips:** Setiap variabel atau objek yang dibuat akan menempati sejumlah memori pada komputer sehingga jika kita bekerja dengan jumlah data yang banyak pastikan kita menghapus seluruh objek pada memori sebelum memulai kerja.

2.6 Tipe dan Struktur Data

Data pada `R` dapat dikelompokan berdasarkan beberapa tipe. Tipe data pada `R` disajikan pada Tabel 2.4.

Table 2.4: Tipe data `R`.

Tipe Data	**Contoh**	**Keterangan**
Logical	TRUE, FALSE	Nilai Boolean
Numeric	12.3, 5, 999	Segala jenis angka
Integer	23L, 97L, 3L	Bilangan integer (bilangan bulat)
Complex	2i, 3i, 9i	Bilangan kompleks
Character	‘a’, “b”, “123”	Karakter dan string
Factor	1, 0, “Merah”	Dapat berupa numerik atau string (namun pada proses akan terbaca sebagai angka)
Raw	Identik dengan “hello”	Segala jenis data yang disimpan sebagai raw bytes

Sintaks berikut adalah contoh dari tipe data pada `R`. Untuk mengetahui tipa

data suatu objek kita dapat menggunakan perintah `class()`

```
# Logical
apel <- TRUE
class(apel)
```

```
## [1] "logical"
```

```
# Numeric
x <- 2.3
class(x)
```

```
## [1] "numeric"
```

```
# Integer
y <- 2L
class(y)
```

```
## [1] "integer"
```

```
# Compleks
z <- 5+2i
class(z)
```

```
## [1] "complex"
```

```
# string
w <- "saya"
class(w)
```

```
## [1] "character"
```

```
# Raw
xy <- charToRaw("hello world")
class(xy)
```

```
## [1] "raw"
```

Keenam jenis data tersebut disebut sebagai tipe data atomik. Hal ini disebabkan karena hanya dapat menangani satu tipe data saja. Misalnya hanya numeric atau hanya integer.

Selain menggunakan fungsi `class()`, kita dapat pula menggunakan fungsi `is_numeric()`, `is.character()`, `is.logical()`, dan sebagainya berdasarkan jenis data apa yang ingin kita cek. Berbeda dengan fungsi `class()`, ouput yang dihasilkan pada fungsi seperti `is_numeric()` adalah nilai Boolean sehingga fungsi ini hanya digunakan untuk mengecek apakah jenis data pada objek sama seperti yang kita pikirkan. Sebagai contoh disajikan pada sintaks berikut:

```
data <- 25

# Cek apakah objek berisi data numerik
is.numeric(data)
```

```
## [1] TRUE
```

```
# Cek apakah objek adalah karakter
is.character(data)
```

```
## [1] FALSE
```

Kita juga dapat mengubah jenis data menjadi jenis lainnya seperti integer menjadi numerik atau sebaliknya. Fungsi yang digunakan adalah `as.numeric()` jika ingin mengubah suatu jenis data menjadi numerik. Fungsi lainnya juga dapat digunakan sesuai dengan kita ingin mengubah jenis data objek menjadi jenis data lainnya.

```
# Integer
apel <- 2L

# Ubah menjadi numerik
as.numeric(apel)
```

```
## [1] 2
```

```
# Cek
is.numeric(apel)
```

```
## [1] TRUE
```

```
# Logical
nangka <- TRUE

# Ubah logical menjadi numeric
as.numeric(nangka)
```

```
## [1] 1
```

```
# Karakter
minum <- "minum"

# ubah karakter menjadi numerik
as.numeric(minum)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA
```

> **Penting!!!**
>
> Konversi karakter menjadi numerik akan menghasilkan output NA (*not available*). R tidak mengetahui bagaimana cara merubah karakter menjadi bentuk numerik.

Berdasarkan Tabel 2, vektor karakter dapat dibuat menggunakan tanda kurung baik *double quote* (”) maupun *single quote* (’).Jika pada teks yang kita tuliskan mengandung *quote* maka kita harus menghentikannya menggunakan tanda ( ). Sbegai contoh kita ingin menuliskan ‘**My friend’s name is “Adi”**, pada sintaks akan dituliskan:

```
'My friend\`s name is "Adi"'
```

```
## [1] "My friend`s name is \"Adi\""
```

```
# Atau

"My friend's name \"Adi\""
```

```
## [1] "My friend's name \"Adi\""
```

Struktur data diklasifikasikan berdasarkan dimensi data dan tie data di dalamnya (homogen atau heterogen). Klasifikasi jenis data disajikan pada Tabel 2.5.

Table 2.5: Struktur data R.

Dimensi	Homogen	Heterogen
1d	Atomik vektor	List
2d	Matriks	Dataframe
nd	Array	

Berdasarkan Tabel tersebut dapat kita lihat bahwa objek terbagi atas dua buah struktur data yaitu homogen dan heterogen. Objek dengan struktur data homogen hanya dapat menyimpan satu tipe atau jenis data saja (numerik saja atau factor saja), sedangkan objek dengan struktur data heterogen akan dapat menyimpan berbagai jenis data.

2.7 Vektor

Vektor merupakan kombinasi berbagai nilai (numerik, karakter, logical, dan sebagainya berdasarkan jenis input data) pada objek yang sma. Pada contoh kasus berikut, pembaca akan memiliki sesuai jenis data input yaitu**vektor numerik**, **vector karakter**, **vektor logical**, dll.

2.7.1 Membuat vektor

Vektor dibuat dengan menggunakan fungsi `c()`(concatenate) seperti yang disajikan pada sintaks berikut:

```
# membuat vektor numerik
x <- c(3,3.5,4,7)
x # print vektor
```

```
## [1] 3.0 3.5 4.0 7.0
```

```
# membuat vektor karakter
y <- c("Apel", "Jeruk", "Rambutan", "Salak")
y # print vektor
```

```
## [1] "Apel"     "Jeruk"    "Rambutan" "Salak"
```

```
# membuat vektor logical
t <- c("TRUE", "FALSE", "TRUE")
t # print vektor
```

```
## [1] "TRUE"  "FALSE" "TRUE"
```

selain menginput nilai pada vektor, kita juga dapat memberi nama nilai setiap vektor menggunakan fungsi `names()`.

```r
# Membuat vektor jumlah buah yang dibeli
Jumlah <- c(5,5,6,7)
names(Jumlah) <- c("Apel", "Jeruk", "Rambutan", "Salak")

# Atau
Jumlah <- c(Apel=5, Jeruk=5, Rambutan=6, Salak=7)

# Print
Jumlah
```

```
##     Apel    Jeruk Rambutan    Salak
##        5        5        6        7
```

Penting!!!

Vektor hanya dapat memuat satu buah jenis data. Vektor hanya dapat mengandung jenis data numerik saja, karakter saja, dll.

Untuk menentukan panjang sebuah vektor kita dapat menggunakan fungsi `lenght()`.

```r
length(Jumlah)
```

```
## [1] 4
```

2.7.2 Missing Values

Seringkali nilai pada vektor kita tidak lengkap atau terdapat nilai yang hilang (*missing value*) pada vektor. *Missing value* pada `R` dilambangkan oleh `NA`(*not available*). Berikut adalah contoh vektor dengan *missing value*.

```r
Jumlah <- c(Apel=5, Jeruk=NA, Rambutan=6, Salak=7)
```

Untuk mengecek apakah dalam objek terdapat *missing value* dapat menggunakan fungsi `is.na()`. ouput dari fungsi tersebut adalah nilai Boolean. Jika terdapat *Missing value*, maka output yang dihasilkan akan memberikan nilai `TRUE`.

```r
is.na(Jumlah)
```

```
##     Apel    Jeruk Rambutan    Salak
##    FALSE     TRUE    FALSE    FALSE
```

Penting!!!

1. Selain `NA` terdapat NaN (*not a number*) sebagai *missing value8*. Nilai tersebut muncul ketika fungsi matematika yang digunakan pada proses perhitungan tidak bekerja sebagaimana mestinya. Contoh: `0/0 = NaN`
2. `is.na()` juga akan menghasilkan nilai `TRUE` pada NaN. Untuk membedakannya dengan `NA` dapat digunakan fungsi `is.nan()`.

2.7.3 Subset Pada Vektor

Subseting vector terdiri atas tiga jenis, yaitu: *positive indexing*, *Negative Indexing*, dan .

- **Positive indexing**: memilih elemen vektor berdasarkan posisinya (indeks) dalam kurung siku.

```
# Subset vektor pada urutan kedua
Jumlah[2]
```

```
## Jeruk
##    NA
```

```
# Subset vektor pada urutan 2 dan 4
Jumlah[c(2, 4)]
```

```
## Jeruk Salak
##    NA     7
```

Selain melalui urutan (indeks), kita juga dapat melakukan subset (membuat himpunan bagian) berdasarkan nama elemen vektornya.

```
Jumlah["Jeruk"]
```

```
## Jeruk
##    NA
```

Penting!!!

Indeks pada `R` dimulai dari 1. Sehingga kolom atau elemen pertama vektor dimulai dari [1]

- **Negative indexing**: mengecualikan (*exclude*) elemen vektor.

```
# mengecualikan elemen vektor 2 dan 4
Jumlah[-c(2,4)]
```

```
##     Apel Rambutan
##        5        6
```

```
# mengecualikan elemen vektor 1 sampai 3
Jumlah[-c(1:3)]
```

```
## Salak
##     7
```

- **Subset berdasarkan vektor logical**: Hanya, elemen-elemen yang nilai yang bersesuaian dalam vektor pemilihan bernilai TRUE, akan disimpan dalam subset.

  Penting!!!

  panjang vektor yang digunakan untuk subset harus sama.

```
Jumlah <- c(Apel=5, Jeruk=NA, Rambutan=6, Salak=7)

# selecting vector
merah <- c(TRUE, FALSE, TRUE, FALSE)

# Subset
Jumlah[merah==TRUE]
```

```
##     Apel Rambutan
##        5        6
```

```
# Subset untuk elemen vektor bukan missing value
Jumlah[!is.na(Jumlah)]
```

```
##     Apel Rambutan    Salak
##        5        6        7
```

2.7.4 Operasi Matematis Menggunakan Vektor

Jika pembaca melakukan operasi dengan vektor, operasi akan diterapkan ke setiap elemen vektor. Contoh disediakan pada sintaks di bawah ini:

```
pendapatan <- c(2000, 1800, 2500, 3000)
names(pendapatan) <- c("Andi", "Joni", "Lina", "Rani")
pendapatan
```

```
## Andi Joni Lina Rani
## 2000 1800 2500 3000
```

```
# Kalikan pendapatan dengan 3
pendapatan*3
```

```
## Andi Joni Lina Rani
## 6000 5400 7500 9000
```

Seperti yang dapat dilihat, `R` mengalikan setiap elemen dengan bilangan pengali.

Kita juga dapat mengalikan vektor dengan vektor lainnya.Contohnya disajikan pada sintaks berikut:

```
# membuat vektor dengan panjang
# sama dengan dengan vektor pendapatan
coefs <- c(2, 1.5, 1, 3)

# Mengalikan pendapatan dengan vektor coefs
pendapatan*coefs
```

```
## Andi Joni Lina Rani
## 4000 2700 2500 9000
```

Berdasarkan sintaks tersebut dapat terlihat bahwa operasi matematik terhadap masing-masing vektor dapat berlangsung jika panjang vektornya sama.

Berikut adalah fungsi lain yang dapat digunakan pada operasi matematika vektor.

```
max(x) # memperoleh nilai maksimum x
min(x) # memperoleh nilai minimum x
range(x) # memperoleh range vektor x
length(x) # memperoleh jumlah vektor x
sum(x) # memperoleh total penjumlahan vektor x
prod(x) # memeperoleh produk elemen vektor x
mean(x) # memperoleh nilai mean vektor x
sd(x) # standar deviasi vektor x
var(x) # varian vektor x
sort(x) # mengurutkan elemen vektor x dari yang terbesar
```

Contoh penggunaan fungsi tersebut disajikan beberapa pada sintaks berikut:

```
# Menghitung range pendapatan
range(pendapatan)
```

```
## [1] 1800 3000
```

```
# menghitung rata-rata dan standar deviasi pendapatan
mean(pendapatan)
```

```
## [1] 2325
```

```
sd(pendapatan)
```

```
## [1] 537.7422
```

2.7.5 Membuat Deret Angka

Secara sederhana vektor merupakan deret angka. Vektor bisa jadi berupa data yang kita miliki atau sengaja kita buat untuk tujuan simulasi matematika. Urutan angka-angka ini bisa memiliki interval konstan, contoh: titik waktu pada analisis reaksi kimia, atau dapat pula intervalnya bersifat acak seperti pada simulasi Monte Carlo.

2.7.5.1 *Regular Sequences*

Operator *colon* (":") dapat digunakan untuk membuat *sequence vector*. Operator tersebut berfungsi sebagai pemisah antara nilai awal dan akhir deret bilangan. Interval nilai *sequence* yang terbentuk adalah '. Berikut adalah contoh bagaimana cara membuat *sequence vector* menggunakan operator *colon*:

```
# vektor benilai 1 s/d 10
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# vektor bernilai 10 s/d -1
10:-1
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1  0 -1
```

Perlu diperhatikan bahwa dalam aplikasinya operator *colon* memiliki prioritas tinggi untuk dilakukan komputasi terlebih dahulu dibandingkan operator matematika. Perhatikan sintaks berikut:

```
n = 10

# membuat vektor bernilai 0 s/d 9
1:n-1
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

```
# membuat vektor bernilai 1 s/d 9
1:(n-1)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

Jika kita menginginkan interval antar angka selain 1, kita dapat menggunakan fungsi `seq()`. Format sintaks tersebut adalah sebagai berikut:

```
seq(from, to, by)
```

Catatan:

- **from, to**: angka awal dan akhir atau nilai maksimum dan minimum deret bilangan yang diinginkan.
- **by**: interval antar nilai

Misalkan kita akan membuat deret bilangan dari 3 sampai 8 dengan interval antar deret sebesar 0,5. Berikut adalah sintaks yang digunakan:

```
seq(from=3,to=8,by=0.5)
```

```
##  [1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
```

2.7.6 Nilai Berulang

Fungsi `rep()` dapat digunakan untuk membuat deret dengan nilai berulang. Format fungsi tersebut adalah sebagai berikut:

```
rep(x, times, each)
```

Catatan:

- **x**: nilai yang hendak dibuat berulang.
- **times**: jumlah pengulangan.
- **each**: argumen tambahan yang menentukan jumlah masing-masing elemen vektor akan dicetak.

```
# cetak angka 5 sebanyak 5 kali
rep(x=5, times=5)
```

```
## [1] 5 5 5 5 5
```

```
# cetak angka 5 dan 6 sebanyak 3 kali
rep(c(5,6), times=3)
```

```
## [1] 5 6 5 6 5 6
```

```
# cetak angka 5 dan 6 masing-masing 3 kali
rep(c(5,6), each=3)
```

```
## [1] 5 5 5 6 6 6
```

2.7.7 Deret Bilangan Acak

Deret bilangan acak biasanya banyak digunakan dalam sebuah simulasi. `R` menyediakan fungsi untuk memproduksi bilangan-bilangan acak tersebut berdasarkan distribusi tertentu. Berikut adalah tabel rangkuman nama distribusi, fungsi, dan argumen yang digunakan:

Table 2.6: Ringkasan Fungsi dan Argumen Distribusi Probabilitas.

Distribusi	Fungsi	Argumen
Beta	`rbeta(n, shape1, shape2, ncp = 0)`	`n` = jumlah observasi; `shape1,shape2` = parameter non-negatif distribusi beta; `ncp` = *non-centrality parameter*
Binomial	`rbinom(n, size, prob)`	`n`= jumlah observasi; `prob` = probabilitas sukses; `size` = jumlah percobaan
Cauchy	`rcauchy(n, location = 0, scale = 1)`	`n` = jumlah observasi; `location, scale` = parameter lokasi dan skala distribusi Cauchy
Chi-Square	`rchisq(n, df, ncp = 0)`	`n` = jumlah observasi; `df` = derajat kebebasan; `ncp` = *non-centrality parameter*

Distribusi	Fungsi	Argumen
Exponential	`rexp(n, rate = 1)`	`n` = jumlah observasi; `rate` = vektor parameter *rate*
F	`rf(n, df1, df2, ncp)`	`n` = jumlah observasi; `df1, df2` = derajat kebebasan; `ncp` = *non-centrality parameter*
Gamma	`rgamma(n, shape, rate = 1, scale = 1/rate)`	`n` = jumlah observasi; `shape, scale` = parameter *shape* dan *scale*; `rate` = alternatif lain argumen `rate`
Geometri	`rgeom(n, prob)`	`n` = jumlah observasi; `prob` = probabilitas sukses
Hipergeometri	`rhyper(nn, m, n, k)`	`nn` = jumlah observasi; `m` = jumlah bola putih dalam wadah; `n` = jumlah bola hitam dalam wadah; `k` = jumlah pengambilan
Log-normal	`rlnorm(n, meanlog = 0, sdlog = 1)`	`n` = jumlah observasi; `meanlog, sdlog` = nilai mean dan simpangan baku dalam skala logaritmik
Negatif Binomial	`rnbinom(n, size, prob, mu)`	`n` = jumlah observasi; `size` = target jumlah percobaan sukses pertama kali; `prob` = probabilitas sukses; `mu` = parameterisasi alternatif melalui mean
Normal	`rnorm(n, mean = 0, sd = 1)`	`n` = jumlah observasi; `mead, sd` = nilai mean dan simpangan baku
Poisson	`rpois(n, lambda)`	`n` = jumlah observasi; `lambda` = vektor nilai mean
Student t	`rt(n, df, ncp)`	`n` = jumlah observasi; `df` = derajat kebebasan; `ncp` = *non-centrality parameter*
Uniform	`runif(n, min = 0, max = 1)`	`n` = jumlah observasi; `min, max` = nilai maksimum dan minimum distribusi
Weibull	`rweibull(n, shape, scale = 1)`	`n` = jumlah observasi; `shape, scale` = parameter *shape* dan *scale*

Berikut adalah contoh pembuatan vektor menggunakan bilangan acak berdistribusi normal:

```
x <- 1:6
error <- rnorm(n=1, mean=0, sd=1)

# cetak x + error dengan 3 nilai signifikan
round((x+error), 3)
```

```
## [1] 0.863 1.863 2.863 3.863 4.863 5.863
```

2.8 Matriks

Matriks seperti Excel sheet yang berisi banyak baris dan kolom (kumpulan bebrapa vektor). Matriks digunakan untuk menggabungkan vektor dengan tipe yang sama, yang bisa berupa numerik, karakter, atau logis. Matriks digunakan untuk menyimpan tabel data dalam R. Baris-baris matriks pada umumnya adalah individu / pengamatan dan kolom adalah variabel.

2.8.1 Membuat matriks

Untuk membuat matriks kita dapat menggunakan fungsi `cbind()` atau `rbind()`. Berikut adalah contoh sintaks untuk membuat matriks.

```
# membuat vektor numerik
col1 <- c(5, 6, 7, 8, 9)
col2 <- c(2, 4, 5, 9, 8)
col3 <- c(7, 3, 4, 8, 7)

# menggabungkan vektor berdasarkan kolom
my_data <- cbind(col1, col2, col3)
my_data
```

```
##      col1 col2 col3
## [1,]    5    2    7
## [2,]    6    4    3
## [3,]    7    5    4
## [4,]    8    9    8
## [5,]    9    8    7
```

```
# Mengubah atau menambahkan nama baris
rownames(my_data) <- c("row1", "row2",
                       "row3", "row4",
                       "row5")
my_data
```

```
##      col1 col2 col3
## row1    5    2    7
## row2    6    4    3
## row3    7    5    4
## row4    8    9    8
## row5    9    8    7
```

Catatan:

- `cbind()`: menggabungkan objek `R` berdasarkan kolom
- `rbind()`: menggabungkan objek `R` berdasarkan baris
- `rownames()`: mengambil atau menetapkan nama-nama baris dari objek seperti-matriks
- `colnames()`: mengambil atau menetapkan nama-nama kolom dari objek seperti-matriks "'

Kita dapat melakukan tranpose (merotasi matriks sehingga kolom menjadi baris dan sebaliknya) menggunakan fungsi `t()`. Berikut adalah contoh penerapannya:

```
t(my_data)
```

```
##      row1 row2 row3 row4 row5
## col1    5    6    7    8    9
## col2    2    4    5    9    8
## col3    7    3    4    8    7
```

Selain melalui pembentukan sejumlah objek vektor, kita juga dapat membuat matriks menggunakan fungsi `matrix()`. Secara sederhana fungsi tersebut dapat dituliskan sebagai berikut:

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

Catatan:

- `data`: vektor data opsional
- `nrow`, **ncol**: jumlah baris dan kolom yang diinginkan, masing-masing.
- `byrow`: nilai logis. Jika FALSE (default) matriks diisi oleh kolom, jika tidak, matriks diisi oleh baris.
- `dimnames`: Daftar dua vektor yang memberikan nama baris dan kolom masing-masing. "'

Dalam kode `R` di bawah ini, data input memiliki panjang 6. Kita ingin membuat matriks dengan dua kolom. Kita tidak perlu menentukan jumlah baris (di sini `nrow = 3`). `R` akan menyimpulkan ini secara otomatis. Matriks diisi kolom demi kolom saat argumen `byrow = FALSE`. Jika kita ingin mengisi matriks dengan baris, gunakan `byrow = TRUE`. Berikut adalah contoh pembuatan matriks menggunakan fungsi `matrix()`.

```
data <- matrix(
           data = c(1,2,3, 11,12,13),
           nrow = 2, byrow = TRUE,
```

```
           dimnames = list(c("row1", "row2"),
                           c("C.1", "C.2", "C.3"))
           )
data
```

```
##      C.1 C.2 C.3
## row1   1   2   3
## row2  11  12  13
```

Untuk mengetahui dimensi dari suatu matriks, kita dapat menggunakan fungsi `ncol()` untuk mengetahui jumlah kolom matriks dan `nrow()` untuk mengetahui jumlah baris pada matriks. Berikut adalah contoh penerapannya:

```
# mengetahui jumlah kolom
ncol(my_data)
```

```
## [1] 3
```

```
# mengetahui jumlah baris
nrow(my_data)
```

```
## [1] 5
```

Jika ingin memperoleh ringkasan terkait dimensi matriks kita juga dapat mengunakan fungsi `dim()` untuk mengetahui jumlah baris dan kolom matriks. Berikut adalah contoh penerapannya:

```
dim(my_data) # jumlah baris dan kolom
```

```
## [1] 5 3
```

2.8.2 Subset Pada Matriks

Sama dengan vektor, subset juga dapat dilakukan pada matriks. Bedanya subset dilakukan berdasarkan baris dan kolom pada matriks.

- **Memilih baris/kolom** berdasarkan pengindeksan positif

baris atau kolom dapat diseleksi menggunakan format `data[row, col]`. Cara selesi ini sama dengan vektor, bedanya kita harus menetukan baris dan kolom dari data yang akan kita pilih. Berikut adalah contoh penerapannya:

```
# Pilih baris ke-2
my_data[2,]
```

```
## col1 col2 col3
##    6    4    3
```

```
# Pilih baris 2 sampai 4
my_data[2:4,]
```

```
##      col1 col2 col3
## row2    6    4    3
## row3    7    5    4
## row4    8    9    8
```

```
# Pilih baris 2 dan 4
my_data[c(2,4),]
```

```
##      col1 col2 col3
## row2    6    4    3
## row4    8    9    8
```

```
# Pilih baris 2 dan kolom 3
my_data[2, 3]
```

```
## [1] 3
```

- **Pilih berdasarkan nama baris/kolom**

Berikut adalah contoh subset berdasarkan nama baris atau kolom.

```
# Pilih baris 1 dan kolom 3
my_data["row1","col3"]
```

```
## [1] 7
```

```
# Pilih baris 1 sampai 4 dan kolom 3
baris <- c("row1","row2","row3")
my_data[baris, "col3"]
```

```
## row1 row2 row3
##    7    3    4
```

- **Kecualikan baris/kolom** dengan pengindeksan negatif

Sama seperti vektor pengecualian data dapat dilakukan di matriks menggunakan pengindeksan negatif. Berikut cara melakukannya:

```
# Kecualikan baris 2 dan 3 serta kolom 3
my_data[-c(2,3), -3]
```

```
##      col1 col2
## row1    5    2
## row4    8    9
## row5    9    8
```

- **Pilihan dengan logik**

Dalam kode R di bawah ini, misalkan kita ingin hanya menyimpan baris di mana col3> = 4:

```
col3 <- my_data[, "col3"]
my_data[col3 >= 4, ]
```

```
##      col1 col2 col3
## row1    5    2    7
## row3    7    5    4
## row4    8    9    8
## row5    9    8    7
```

2.8.3 Perhitungan Menggunakan Matriks

_ Kita juga dapat melakukan operasi matematika pada matriks. Pada operasi matematika pada matriks proses yang terjadi bisa lebih kompleks dibanding pada vektor, dimana kita dapat melakukan operasi untuk memperoleh gambaran data pada tiap kolom atau baris.

Berikut adalah contoh operasi matematika sederhana pada matriks:

```
# mengalikan masing-masing elemen matriks dengan 2
my_data*2
```

```
##      col1 col2 col3
## row1   10    4   14
## row2   12    8    6
## row3   14   10    8
## row4   16   18   16
## row5   18   16   14
```

```
# memperoleh nilai log basis 2 pada masing-masing elemen matriks
log2(my_data)
```

```
##          col1     col2     col3
## row1 2.321928 1.000000 2.807355
## row2 2.584963 2.000000 1.584963
## row3 2.807355 2.321928 2.000000
## row4 3.000000 3.169925 3.000000
## row5 3.169925 3.000000 2.807355
```

Seperti yang telah penulis jelaskan sebelumnya, kita juga dapat melakukan operasi matematika untuk memperoleh hasil penjumlahan elemen pada tiap baris atau kolom dengan menggunakan fungsi `rowSums()` untuk baris dan `colSums()` untuk kolom.

```
# Total pada tiap kolom
colSums(my_data)
```

```
## col1 col2 col3
##   35   28   29
```

```
# Total pada tiap baris
rowSums(my_data)
```

```
## row1 row2 row3 row4 row5
##   14   13   16   25   24
```

Jika kita tertarik untuk mencari nilai rata-rata tiap baris arau kolom kita juga dapat menggunakan fungsi `rowMeans()` atau `colMeans()`. Berikut adalah contoh penerapannya:

```
# Rata-rata tiap baris
rowMeans(my_data)
```

```
##     row1     row2     row3     row4     row5
## 4.666667 4.333333 5.333333 8.333333 8.000000
```

```
# Rata-rata tiap kolom
colMeans(my_data)
```

```
## col1 col2 col3
##  7.0  5.6  5.8
```

Kita juga dapat melakukan perhitungan statistika lainnya menggunakan fungsi `apply()`. Berikut adalah format sederhananya:

```
apply(x, MARGIN, FUN)
```

Catatan:

- `x` : data matriks
- `MARGIN` : Nilai yang dapat digunakan adalah `1` (untuk operasi pada baris) dan `2` (untuk operasi pada kolom)
- `FUN` : fungsi yang diterapkan pada baris atau kolom

untuk mengetahui fungsi (`FUN`) apa saja yang dapat diterapkan pada fungsi `apply()` jalankan sintaks bantuan berikut:

```
help(apply)
```

Berikut adalah contoh penerapannya:

```
# Rata-rata pada tiap baris
apply(my_data, 1, mean)
```

```
##     row1     row2     row3     row4     row5
## 4.666667 4.333333 5.333333 8.333333 8.000000
```

```
# Median pada tiap kolom
apply(my_data, 2, median)
```

```
## col1 col2 col3
##    7    5    7
```

Perhitungan lainnya tidak akan dibahas pada *chapter* ini. Operasi matriks lebih lengkap selanjutnya akan dibahas pada *chapter* selanjutnya.

2.9 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
2. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
3. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.

4. STHDA. **Easy R Programming Basics**. http://www.sthda.com/english/wiki/easy-r-programming-basics
5. The R Core Team. 2018. **R: A Language and Environment for Statistical Computing**. R Manuals.
6. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.

Chapter 3

Visualisasi Data

Visualisasi data merupakan bagian yang sangat penting untuk mengkomunikasikan hasil analisa yang telah kita lakukan. Selain itu, komunikasi juga membantu kita untuk memperoleh gambaran terkait data selama proses analisa data sehingga membantu kita dalam memutuskan metode analisa apa yang dapat kita terapkan pada data tersebut.

`R` memiliki library visualisasi yang sangat beragam, baik yang merupakan fungsi dasar pada `R` maupun dari sumber lain seperti ggplot dan lattice. Seluruh library visualisasi tersebut memiliki kelebihan dan kekurangannya masing-masing.

Pada *chapter* ini kita tidak akan membahas seluruh library tersebut. Kita akab berfokus pada fungsi visualisasi dasar bawaan dari `R`. kita akan mempelajari mengenai jenis visualisasi data sampai dengan melakukan kustomisasi pada parameter grafik yang kita buat.

3.1 Visualisasi Data Menggunakan Fungsi plot()

Fungsi `plot()` merupakan fungsi umum yang digunakan untuk membuat plot pada `R`. Format dasarnya adalah sebagai berikut:

```
plot(x, y, type="p")
```

Catatan:

- **x dan y**: titik koordinat plot Berupa variabel dengan panjang atau jumlah observasi yang sama.
- **type**: jenis grafik yang hendak dibuat. Nilai yang dapat dimasukkan antara lain:

- type="p" : membuat plot titik atau scatterplot. Nilai ini merupakan default pada fungsi `plot()`.
- type="l" : membuat plot garis.
- type="b" : membuat plot titik yang terhubung dengan garis.
- type="o" : membuat plot titik yang ditimpa oleh garis.
- type="h" : membuat plot garis vertikal dari titik ke garis y=0.
- type="s" : membuat fungsi tangga.
- type="n" : tidak membuat grafik plot sama sekali, kecuali plot dari axis. Dapat digunakan untuk mengatur tampilan suatu plot utama yang diikuti oleh sekelompok plot tambahan.

Untuk lebih memahaminya berikut penulis akan sajikan contoh untuk masing-masing grafik tersebut. Berikut adalah contoh sintaks dan hasil plot yang disajikan pada Gambar 3.1:

```
# membuat vektor data
x <- c(1:10); y <- x^2
```

```
# membagi jendela grafik menajdi 2 baris dan 4 kolom
par(mfrow=c(2,4))

# loop
type <- c("p","l","b","o","h","s","n")
for (i in type){
  plot(x,y, type= i,
       main= paste("type=", i))
}
```

Pada contoh selanjutnya kita akan mencoba membuat kembali data yang akan kita plotkan. Data pada contoh kali ini merupakan data suatu fungsi matematika. Berikut adalah sintaks yang digunakan:

```
set.seed(123)
x <- seq(from=0, to=10, by=0.1)
y <- x^2*exp(-x/2)*(1+rnorm(n=length(x), mean=0, sd=0.05))
```

```
par(mfrow=c(1,2),
    # mengatur margin grafik
    mar=c(4,4,1.5,1.5),
    # mengatur margin sumbu
    mex=0.8,
```

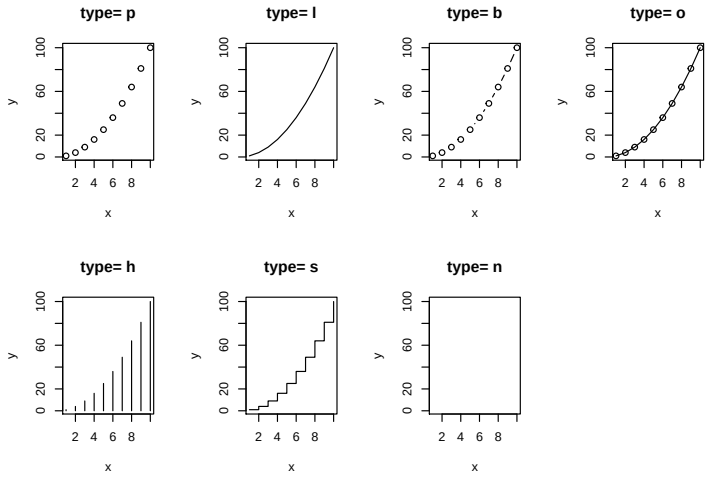

Figure 3.1: Plot berbagai jenis setting type

```
    # arah tick sumbu koordinat
    tcl=0.3)
plot(x, y, type="l")
plot(x, y, type="o")
```

Fungsi lain yang dapat digunakan untuk membuat kurva suatu persamaan matematis adalah fungsi `curve()`. Berbeda dengan fungsi `plot()` yang perlu menspesifikasi objek pada sumbu x dan y, fungsi `curve()` hanya perlu menspesifikasi objek sumbu x saja. Format fungsi `curve()` adalah sebagai berikut:

```
curve(expr, from = NULL, to = NULL, add = FALSE)
```

Catatan:

- **expr**: persamaan matematika
- **from dan to**: nilai awal dan akhir (maksimum atau minimum)
- **add**: nilai logik yang menentukan apakah kurva perlu ditambahkan kedalam kurva sebelumnya.

Berikut adalah contoh visualisasi menggunakan fungsi `curve()`:

```
par(mfrow=c(1,2),
    # mengatur margin grafik
    mar=c(4,4,1.5,1.5),
```

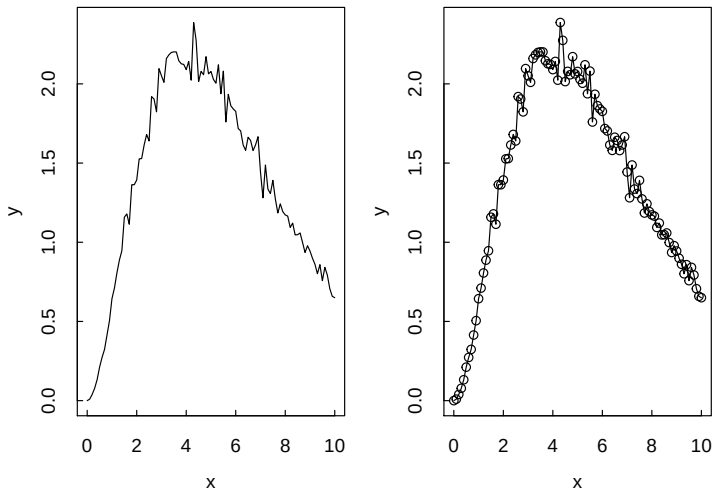

Figure 3.2: Plot fungsi matematika

```
    # mengatur margin sumbu
    mex=0.8,
    # arah tick sumbu koordinat
    tcl=0.3)

# Grafik kiri
curve(expr=x^2*exp(-x/2),
      from=0, to=10)

# Grafik kanan
plot(x, y, pch=19, cex=0.7,
     xlab="Waktu (detik)",
     ylab="Sinyal Intensitas")
curve(expr=x^2*exp(-x/2),
      from=0, to=10, add=TRUE)
```

3.2 Visualisasi Lainnya

Visualisasi lainnya yang sering digunakan antara lain: histogram, density plot, bar plot, dan box plot.

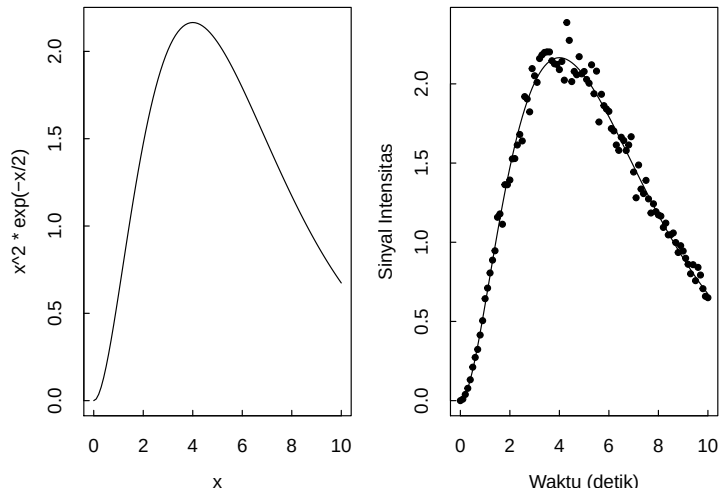

Figure 3.3: Visualisasi menggunakan fungsi curve (sebelah kiri) dan visualisasi menggunakan fungsi plot dan curve (sebelah kanan)

3.2.1 Bar Plot

Barplot pada `R` dapat dibuat menggunakan fungsi `barplot()`. Untuk lebih memahaminya berikut disajikan contoh barplot menggunakan dataset `VADeaths`. Untuk memuatnya jalankan sintaks berikut:

```
VADeaths
```

```
##       Rural Male Rural Female Urban Male Urban Female
## 50-54       11.7          8.7       15.4          8.4
## 55-59       18.1         11.7       24.3         13.6
## 60-64       26.9         20.3       37.0         19.3
## 65-69       41.0         30.9       54.6         35.1
## 70-74       66.0         54.3       71.1         50.0
```

Contoh bar plot untuk variabel `Rural Male` disajikan pada Gambar 3.4:

```
par(mfrow=c(1,2))
barplot(VADeaths[, "Rural Male"], main="a")
barplot(VADeaths[, "Rural Male"], main="b", horiz=TRUE)
```

```
par(mfrow=c(1,1))
```

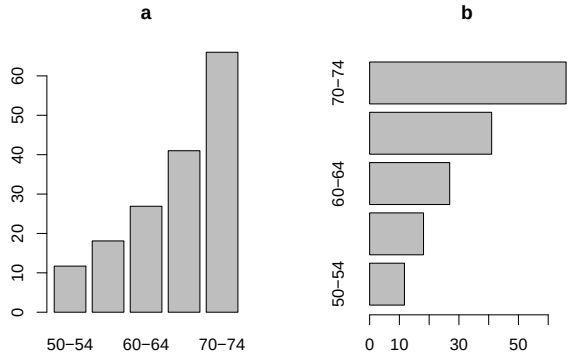

Figure 3.4: a. bar plot vertikal; b. bar plot horizontal

Kita dapat mengubah warna pada masing-masing bar, baik outline bar maupun box pada bar. Selain itu kita juga dapat mengubah nama grup yang telah dihasilkan sebelumnya. Berikut sintaks untuk melakukannya dan output yang dihasilkan pada Gambar 3.5:

```
barplot(VADeaths[, "Rural Male"],
        # ubah warna ouline menjadi steelblue
        border="steelblue",
        # ubah wana box
        col= c("grey", "yellow", "steelblue", "green", "orange"),
        # ubah nama grup dari A sampai E
        names.arg = LETTERS[1:5],
        # ubah orientasi menajadi horizontal
        horiz=TRUE)
```

Untuk bar plot dengan *multiple group*, tersedia dua pengaturan posisi yaitu *stacked bar plot*(menunjukkan proporsi penyusun pada masing-masing grup) dan *grouped bar plot*(melihat perbedaan individual pada masing-masing grup). Pada Gambar 3.6 dan Gambar 3.7 , disajikan kedua jenis bar plot tersebut.

```
# staked
barplot(VADeaths,
         col = c("lightblue", "mistyrose", "lightcyan",
                 "lavender", "cornsilk"),
        legend = rownames(VADeaths))
```

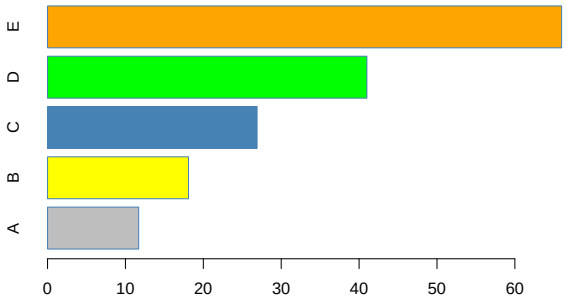

Figure 3.5: Kustomisasi bar plot

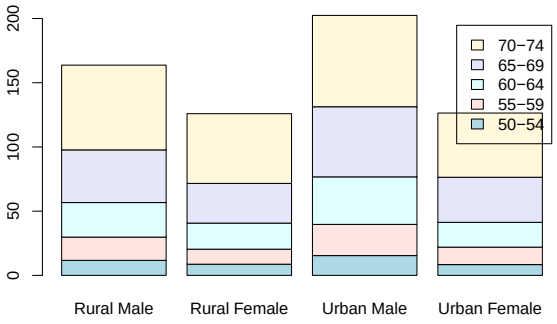

Figure 3.6: Stacked bar plot

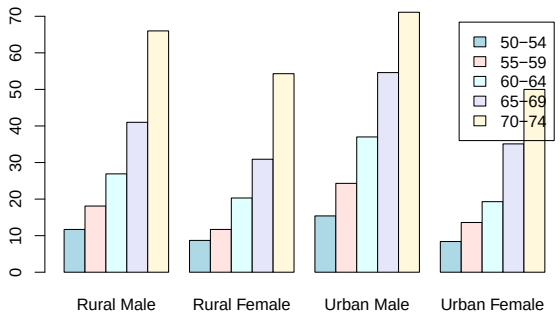

Figure 3.7: Grouped bar plot

```
# grouped
barplot(VADeaths,
         col = c("lightblue", "mistyrose", "lightcyan",
                 "lavender", "cornsilk"),
        legend = rownames(VADeaths), beside = TRUE)
```

3.2.2 Histogram dan Density Plot

Fungsi `hist()` dapat digunakan untuk membuat histogram pada `R`. Secara sederhana fungsi tersebut didefinisikan sebagai berikut:

```
hist(x, breaks="Sturges")
```

Catatan:

- **x**: vektor numerik
- **breaks**: *breakpoints* antar sel histogram.

Pada dataset `trees` akan dibuat histogram variabel `Height`. Untuk melakukannya jalankan sintaks berikut:

```
hist(trees$Height)
```

Output yang dihasilkan disajikan pada Gambar 3.8:

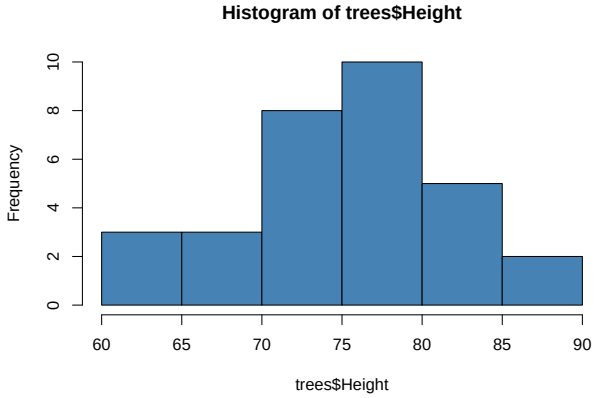

Figure 3.8: Histogram

Density plot pada `R` dapat dibuat menggunakan fungsi `density()`. Berbeda dengan fungsi `hist()`, fungsi ini tidak langsung menghasilkan grafik densitas. Fungsi `density()` hanya menghitung kernel densitas pada data. Densitas yang telah dihitung selanjutnya diplotkan menggunakan fungsi `plot()`. Berikut adalah sintaks dan output yang dihasilkan pada Gambar 3.9:

```
# menghitung kernel density
dens <- density(trees$Height)

# plot densitas dengan outline merah
plot(dens,col="red")
```

Kita juga dapat menambahkan grafik densitas pada histogram sehingga mempermudah pembacaan pada histogram. Untuk melakukannya kita perlu mengubah kernel histigram dari frekuensi menjadi density dengan menambahkan argumen `freq=FALSE` pada fungsi `hist()`. Selanjutnya tambahkan fungsi `polygon()` untuk memplotkan grafik densitas. Berikut adalah sintak dan output yang dihasilkan pada Gambar 3.10:

```
# menghitung kernel density
dens <- density(trees$Height)

# histogram
hist(trees$Height, freq=FALSE, col="steelblue")

# tambahkan density plot
polygon(dens, border="red")
```

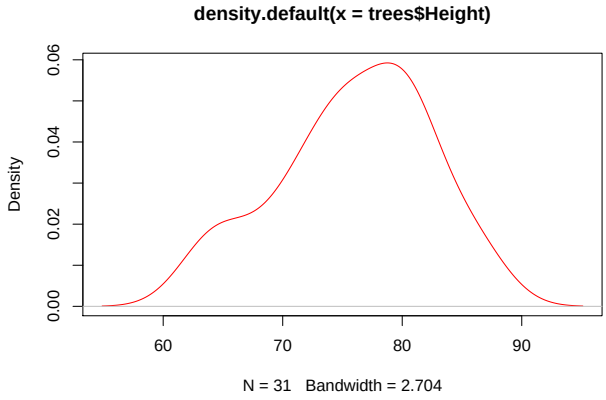

Figure 3.9: Density plot

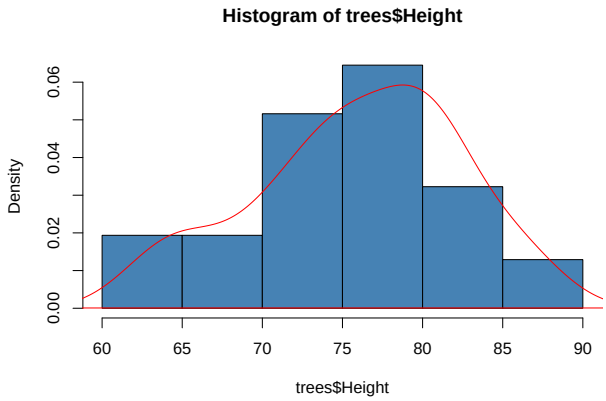

Figure 3.10: Density plot dan histogram

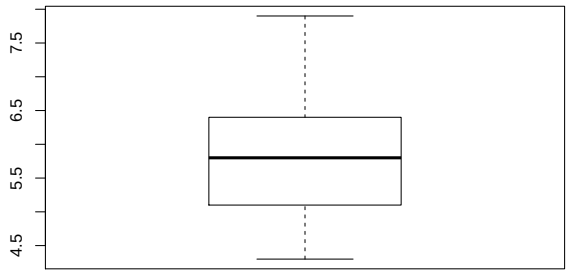

Figure 3.11: Boxplot variabel Sepal.Length

3.2.3 Box plot

Box plot pada `R` dapat dibuat menggunakan fungsi `boxplot()`. Berikut adalah sintaks untuk membuat boxplot variabel `Sepal.Lenght` pada dataset `iris` dan output yang dihasilkan pada Gambar 3.11:

```
boxplot(iris$Sepal.Length)
```

Boxplot juga dapat dibuat berdasarkan variabel factor. Hal ini berguna untuk melihat perbedaan ditribusi data pada masing-masing grup. Pada sintaks berikut dibuat boxplot berdasarkan variabel `Species`. Output yang dihasilkan disajikan pada Gambar 3.12:

```
boxplot(iris$Sepal.Length~iris$Species)
```

Kita juga dapat mengubah warna outline dan box pada boxplot. Berikut adalah contoh sintaks yang digunakan untuk melakukannya dan output yang dihasilkan disajikan pada Gambar 3.13:

```
boxplot(iris$Sepal.Length~iris$Species,
        # ubah warna outline menjadi steelblue
        border = "steelblue",
        # ubah warna box berdasarkan grup
        col= c("#999999", "#E69F00", "#56B4E9"))
```

Kita juga dapat membuat boxplot pada *multiple group*. Data yang digunakan untuk contoh tersebut adalah dataset `ToothGrowth`. Berikut adalah sintaks untuk memuat dataset tersebut:

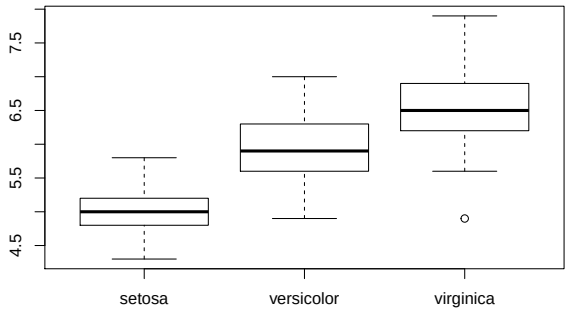

Figure 3.12: Boxplot berdasarkan variabel species

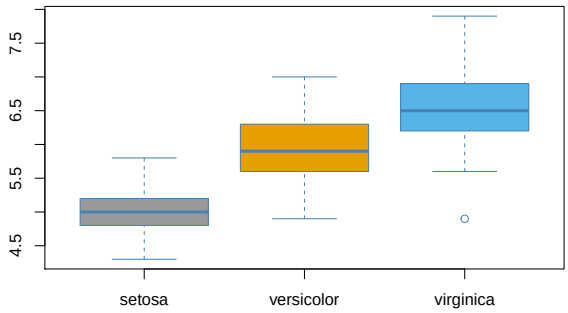

Figure 3.13: Boxplot dengan warna berdasarkan spesies

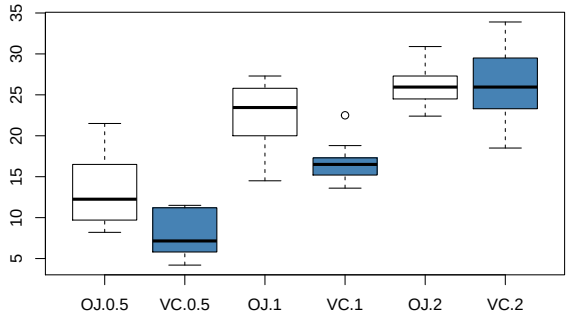

Figure 3.14: Boxplot multiple group

```
# ubah variable dose menjadi factor
ToothGrowth$dose <- as.factor(ToothGrowth$dose)

# print
head(ToothGrowth)
```

```
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5
```

Contoh sintaks dan output boxplot *multiple group* disajikan pada Gambar 3.14:

```
boxplot(len ~ supp*dose, data = ToothGrowth,
        col = c("white", "steelblue"))
```

3.3 Kustomisasi Parameter Grafik

Pada bagian ini penulis akan menjelaskan cara untuk kustomisasi parameter grafik seperti:

a. menambahkan judul, legend, teks, axis, dan garis.
b. mengubah skala axis, simbol plot, jenis garis, dan warna.

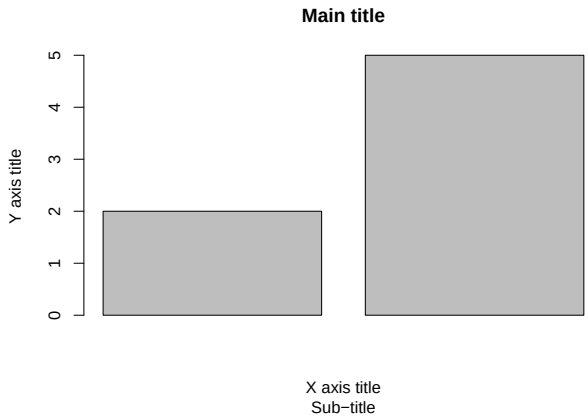

Figure 3.15: Menambahkan Judul

3.3.1 Menambahkan Judul

Pada grafik di R, kita dapat menambahkan judul dengan dua cara, yaitu: pada plot melalui parameter dan melalui fungsi plot(). Kedua cara tersebut tidak berbeda satu sama lain pada parameter input.

Untuk menambahkan judul pada plot secara langsung, kita dapat menggunakan argumen tambahan sebagai berikut:

a. **main**: teks untuk judul.
b. **xlab**: teks untuk keterangan axis X.
c. **ylab**: teks untuk keterangan axis y.
d. **sub**: teks untuk sub-judul.

Berikut contoh sintaks penerapan masing-masing argumen tersebut beserta dengan output yang dihasilkan pada Gambar 3.15:

```
# menambahkan judul
barplot(c(2,5), main="Main title",
        xlab="X axis title",
        ylab="Y axis title",
        sub="Sub-title")
```

kita juga dapat melakukan kustomisasi pada warna, *font style*, dan ukuran font judul. Untuk melakukan kustomisasi pada warna pada judul, kita dapat menambahkan argumen sebagai berikut:

a. **col.main**: warna untuk judul.

b. **col.lab**: warna untuk keterangan axis.
c. **col.sub**: warna untuk sub-judul

Untuk kustomisasi font judul, kita dapat menambahkan argumen berikut:

a. **font.main**: *font style* untuk judul.
b. **font.lab**: *font style* untuk keterangan axis.
c. **font.sub**: *font style* untuk sub-judul.

Penting!!!

Nilai yang dapat dimasukkan antara lain:

- **1**: untuk teks normal.
- **2**: untuk teks cetak tebal.
- **3**: untuk teks cetak miring.
- **4**: untuk teks cetak tebal dan miring.
- **5**: untuk font simbol.

Sedangkan untuk ukuran font, kita dapat menambahkan variabel berikut:

a. **cex.main**: ukuran teks judul.
b. **cex.lab**: ukuran teks keterangan axis.
c. **cex.sub**: ukuran teks sub-judul.

Berikut sintaks penerapan seluruh argumen tersebut beserta output yang dihasilkan pada Gambar 3.16:

```
# menambahkan judul
barplot(c(2,5),
        # menambahkan judul
        main="Main title",
        xlab="X axis title",
        ylab="Y axis title",
        sub="Sub-title",
        # kustomisasi warna font
        col.main="red",
        col.lab="blue",
        col.sub="black",
        # kustomisasi font style
        font.main=4,
        font.lab=4,
        font.sub=4,
        # kustomisasi ukuran font
        cex.main=2,
        cex.lab=1.7,
        cex.sub=1.2)
```

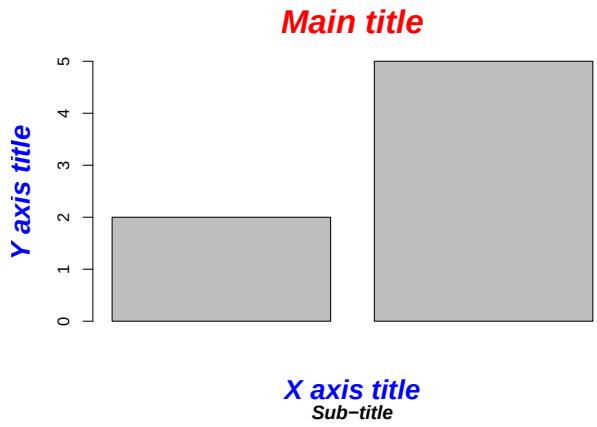

Figure 3.16: Menambahkan Judul (2)

Kita telah belajar bagaimana menambahkan judul langsung pada fungsi plot. Selain cara tersebut, telah penulis jelaskan bahwa kita dapat menambahkan judul melalui fungsi `title()`. argumen yang dimasukkan pada dasarnya tidak berbeda dengan ketika kita menambahkan judul secara langsung pada plot. Berikut adalah contoh sintaks dan output yang dihasilkan pada Gambar 3.17:

```
# menambahkan judul
barplot(c(2,5,8))

# menambahkan judul
title(main="Main title",
      xlab="X axis title",
      ylab="Y axis title",
      sub="Sub-title",
      # kustomisasi warna font
      col.main="red",
      col.lab="blue",
      col.sub="black",
      # kustomisasi font style
      font.main=4,
      font.lab=4,
      font.sub=4,
      # kustomisasi ukuran font
      cex.main=2,
      cex.lab=1.7,
      cex.sub=1.2)
```

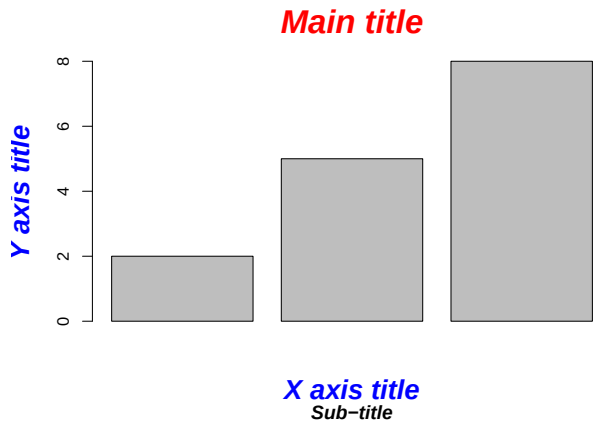

Figure 3.17: Menambahkan Judul (3)

3.3.2 Menambahkan Legend

Fungsi `legend()` pada `R` dapat digunakan untuk menambahkan legend pada grafik. Format sederhananya adalah sebagai berikut:

```
legend(x, y=NULL, legend, fill, col, bg)
```

Catatan:

- **x** dan **y**: koordinat yang digunakan untuk posisi legend.
- **legend**: teks pada legend
- **fill**: warna yang digunakan untuk mengisi box disamping teks legend.
- **col**: warna garis dan titik disamping teks legend.
- **bg**: warna latar belakang legend box.

Berikut adalah contoh sintaks dan ouput penerapan argumen disajikan pada Gambar 3.18:

```
# membuat vektor numerik
x <- c(1:10)
y <- x^2
z <- x*2

# membuat line plot
plot(x,y, type="o", col="red", lty=1)
```

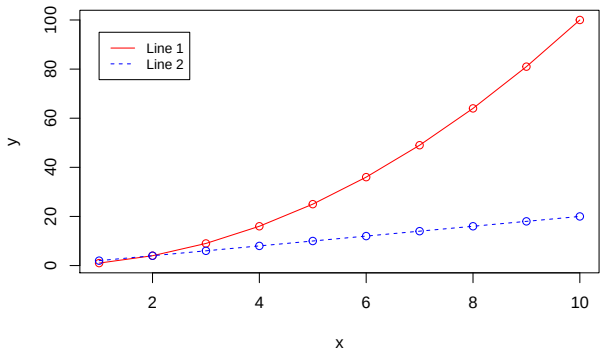

Figure 3.18: Menambahkan legend

```
# menambahkan line plot
lines(x,z, type="o", col="blue", lty=2)

# menambahkan legend
legend(1, 95, legend=c("Line 1", "Line 2"),
       col=c("red", "blue"), lty=1:2, cex=0.8)
```

Kita dapat menambahkan judul, merubah font, dan merubah warna backgroud pada legend. Argumen yang ditambahkan pada legend adalah sebagai berikut:

a. **title**: Judul legend
b. **text.font**: integer yang menunjukkan *font style* pada teks legend. Nilai yang dapat dimasukkan adalah sebagai berikut:
   - **1**: normal
   - **2**: cetak tebal
   - **3**: cetak miring
   - **4**: cetak tebal dan miring.
c. **bg**: warna background legend box.

Berikut adalah penerapan sintaks dan output yang dihasilkan pada Gambar 3.19:

```
# membuat line plot
plot(x,y, type="o", col="red", lty=1)

# menambahkan line plot
```

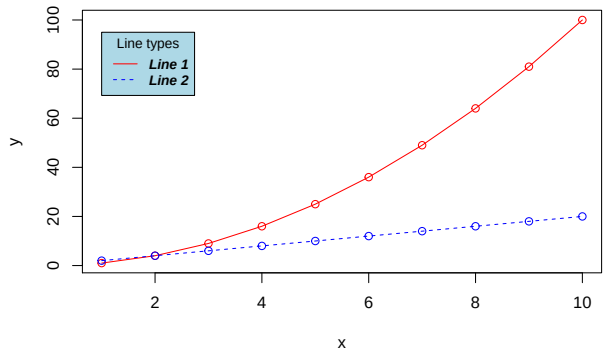

Figure 3.19: Menambahkan legend (2)

```
lines(x,z, type="o", col="blue", lty=2)

# menambahkan legend
legend(1, 95, legend=c("Line 1", "Line 2"),
       col=c("red", "blue"), lty=1:2, cex=0.8,
       title="Line types", text.font=4, bg='lightblue')
```

Kita dapat melakukan kustomisasi pada border dari legend melalui argumen `box.lty=`(jenis garis), `box.lwd=`(ukuran garis), dan `box.col=`(warna box). Berikut adalah penerapan argumen tersebut beserta output yang dihasilkan pada Gambar 3.20:

```
# membuat line plot
plot(x,y, type="o", col="red", lty=1)

# menambahkan line plot
lines(x,z, type="o", col="blue", lty=2)

# menambahkan legend
legend(1, 95, legend=c("Line 1", "Line 2"),
       col=c("red", "blue"), lty=1:2, cex=0.8,
       title="Line types", text.font=4, bg='white',
       box.lty=2, box.lwd=2, box.col="steelblue")
```

Selain menggunakan koordinat, kita juga dapat melakukan kustomisasi posisi legend menggunakan *keyword* seperti: bottomright“,”bottom“,”bottomleft“,”left“,”topleft“,”top“,”topright“,”right” and “center”. Sejumlah kustomisasi legend berdasarkan *keyword* disajikan pada Gambar 3.21:

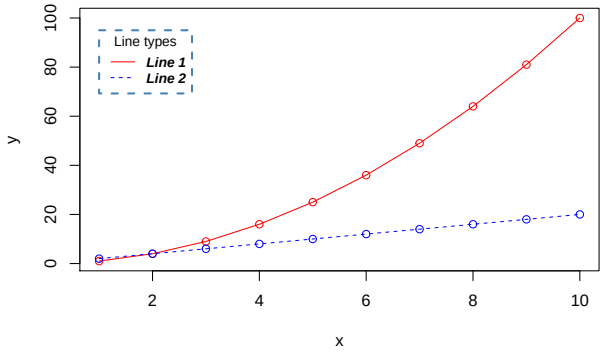

Figure 3.20: Menambahkan legend (3)

```
# plot
plot(x,y, type = "n")

# posisi kiri atas, inset =0.05
legend("topleft",
  legend = "(x,y)",
  title = "topleft, inset = .05",
  inset = 0.05)
# posisi atas
legend("top",
       legend = "(x,y)",
       title = "top")
# posisi kanan atas inset = .02
legend("topright",
       legend = "(x,y)",
       title = "topright, inset = .02",
       inset = 0.02)
# posisi kiri
legend("left",
       legend = "(x,y)",
       title = "left")
# posisi tengah
legend("center",
       legend = "(x,y)",
       title = "center")
# posisi kanan
legend("right",
```

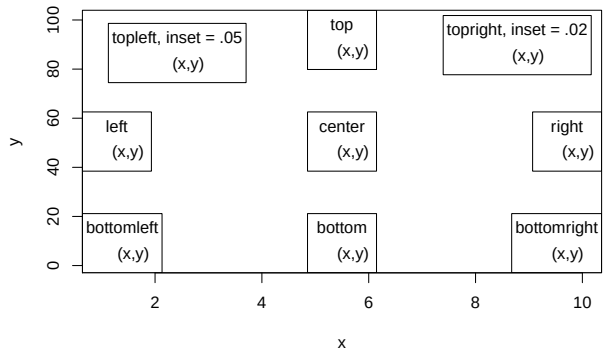

Figure 3.21: Kustomisasi posisi legend

```
       legend = "(x,y)",
       title = "right")
# posisi kiri bawah
legend("bottomleft",
       legend = "(x,y)",
       title = "bottomleft")
# posisi bawah
legend("bottom",
       legend = "(x,y)",
       title = "bottom")
# posisi kanan bawah
legend("bottomright",
       legend = "(x,y)",
       title = "bottomright")
```

3.3.3 Menambahkan Teks Pada Grafik

Teks pada grafik dapat kita tambahkan baik sebagai keterangan yang menunjukkan label suatu observasi, keterangan tambahan disekitar bingkai grafik, maupun sebuah persamaan yang ada pada bidang grafik. Untuk menambahkannya kita dapat menggunakan dua buah fungsi yaitu: `text()` dan `mtext()`.

FUngsi `text()` berguna untuk menambahkan teks di dalam bidang grafik seperti label titik observasi dan persamaan di dalam bidang grafik. Format yang digunakan adalah sebagai berikut:

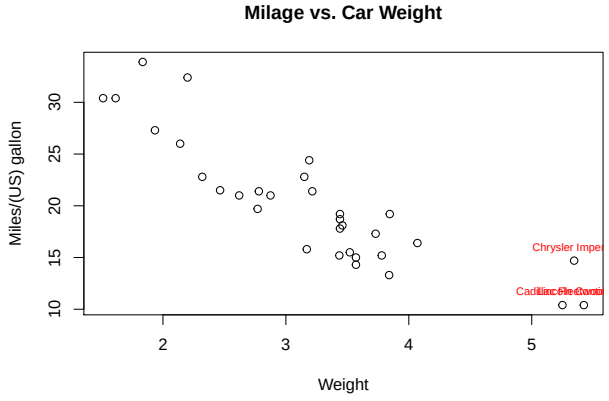

Figure 3.22: Menambahkan teks

```
text(x, y, labels)
```

Catatan:

- **x** dan **y**: vektor numerik yang menunjukkan koordinat posisi teks.
- **labels**: vektor karakter yang menunjukkan teks yang hendak ditulis.

Berikut adalah contoh sintaks untuk memberi label pada sejumlah data yang memiliki kriteria yang kita inginkan dan output yang dihasilkan pada Gambar 3.22:

```
# tandai observasi yang memiliki nilai
# mpg < 15 dan wt > 5
d <- mtcars[mtcars$wt >= 5 & mtcars$mpg <= 15, ]


# plot
plot(mtcars$wt, mtcars$mpg, main="Milage vs. Car Weight",
      xlab="Weight", ylab="Miles/(US) gallon")

# menambahkan text
text(d$wt, d$mpg,  row.names(d),
     cex=0.65, pos=3,col="red")
```

Sedangkan sintaks berikut adalah contoh bagaimana menambahkan persamaan kedalam bidang grafik dan output yang dihasilkan pada Gambar 3.23:

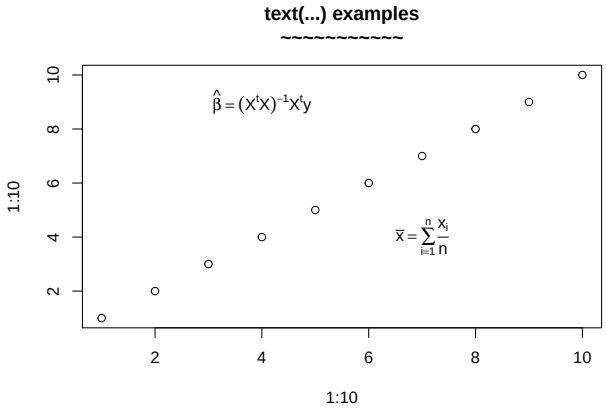

Figure 3.23: Menambahkan teks (2)

```
plot(1:10, 1:10,
     main="text(...) examples\n~~~~~~~~~~~~")
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(7, 4, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
```

Fungsi `mtext()` berguna untuk menambahkan teks pada frame sekitar bidang grafik. Format yang digunakan adalah sebagai berikut:

```
mtext(text, side=3)
```

Catatan:

- **text**: teks yang akan ditulis.
- **side**: integer yang menunjukkan lokasi teks yang akan ditulis. Nilai yang dapat dimasukkan antara lain:
- 1: bawah
- 2: kiri
- 3: atas
- 4: kanan.

Berikut adalah contoh penerapan dan output yang dihasilkan pada Gambar 3.24:

```
plot(1:10, 1:10,
     main="mtext(...) examples\n~~~~~~~~~~~~")
mtext("Magic function", side=3)
```

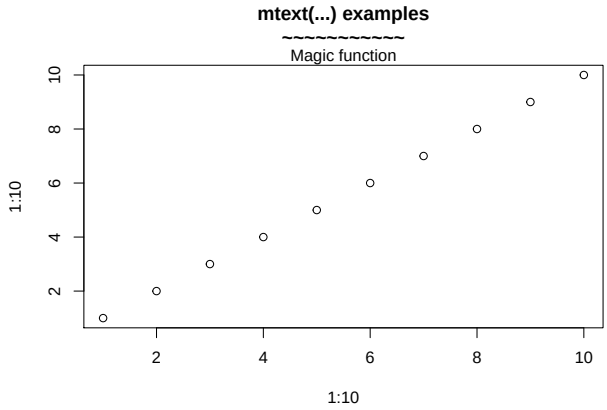

Figure 3.24: Menambahkan teks (3)

3.3.4 Menambahkan Garis Pada Plot

Fungsi `abline()` dapat digunakan untuk menamabahkan garis pada plot. Garis yang ditambahkan dapat berupa garis vertikal, horizontal, maupun garis regresi. Format yang digunakan adalah sebagi berikut:

```
abline(v=y)
```

Berikut adalah contoh sintaks bagaimana menambahkan garis pada sebuah plot dan output yang dihasilkan disajikan pada Gambar 3.25:

```
# membuat plot
plot(mtcars$wt, mtcars$mpg, main="Milage vs. Car Weight",
      xlab="Weight", ylab="Miles/(US) gallon")

# menambahkan garis vertikal di titik rata-rata weight
abline(v=mean(mtcars$wt), col="red", lwd=3, lty=2)

# menambahkan garis horizontal di titik rata-rata  mpg
abline(h=mean(mtcars$mpg), col="blue", lwd=3, lty=3)

# menambahkan garis regresi
abline(lm(mpg~wt, data=mtcars), lwd=4, lty=4)
```

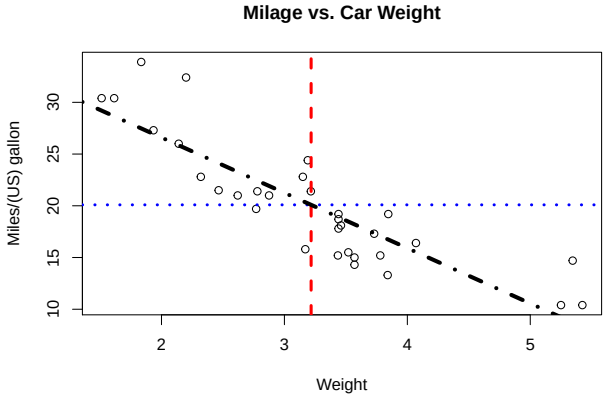

Figure 3.25: Menambahkan garis

3.3.5 Merubah Simbol plot dan Jenis Garis

Simbol plot (jenis titik) dapat diubah dengan menambahkan argumen `pch=` pada plot. Nilai yang dimasukkan pada argumen tersebut adalah integer dengan kemungkinan nilai sebagai berikut:

- pch = 0,square
- pch = 1,circle (default)
- pch = 2,triangle point up
- pch = 3,plus
- pch = 4,cross
- pch = 5,diamond
- pch = 6,triangle point down
- pch = 7,square cross
- pch = 8,star
- pch = 9,diamond plus
- pch = 10,circle plus
- pch = 11,triangles up and down
- pch = 12,square plus
- pch = 13,circle cross
- pch = 14,square and triangle down
- pch = 15, filled square
- pch = 16, filled circle
- pch = 17, filled triangle point-up
- pch = 18, filled diamond
- pch = 19, solid circle
- pch = 20,bullet (smaller circle)
- pch = 21, filled circle blue

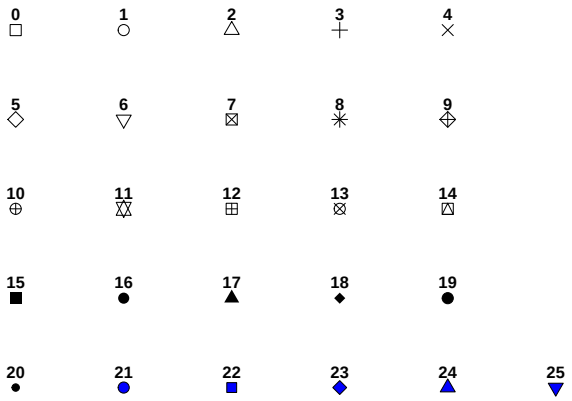

Figure 3.26: Symbol plot

- pch = 22, filled square blue
- pch = 23, filled diamond blue
- pch = 24, filled triangle point-up blue
- pch = 25, filled triangle point down blue

Untuk lebih memahami bentuk simbol tersebut, penulis akan menyajikan sintaks yang menampilkan seluruh simbol tersebut pada satu grafik. Output yang dihasilkan disajikan pada Gambar 3.26:

```
generateRPointShapes<-function(){
  # menentukan parameter plot
  oldPar<-par()
  par(font=2, mar=c(0.5,0,0,0))
  # produksi titik axis
  y=rev(c(rep(1,6),rep(2,5), rep(3,5), rep(4,5), rep(5,5)))
  x=c(rep(1:5,5),6)
  # plot seluruh titik dan label
  plot(x, y, pch = 0:25, cex=1.5, ylim=c(1,5.5), xlim=c(1,6.5),
       axes=FALSE, xlab="", ylab="", bg="blue")
  text(x, y, labels=0:25, pos=3)
  par(mar=oldPar$mar,font=oldPar$font )
}

# Print
generateRPointShapes()
```

Pada `R` kita juga dapat mengatur jenis garis yang akan ditampilkan pada plot dengan menambahkan argumen `lty=` (*line type*) pada fungsi plot. Nilai yang

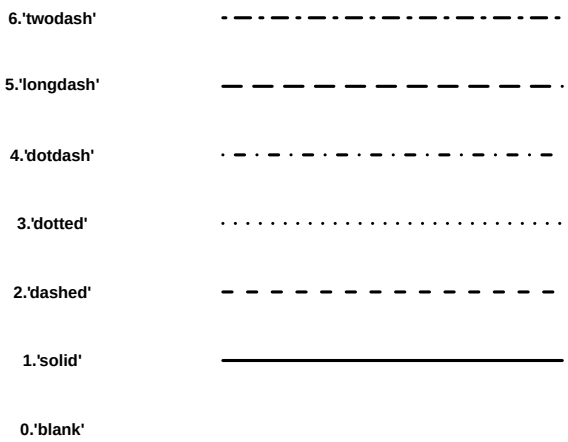

Figure 3.27: Line type

dapat dimasukkan adalah nilai integer. Keterangan masing-masing nilai tersebut adalah sebagai berikut:

- lty = 0, blank
- lty = 1, solid (default)
- lty = 2, dashed
- lty = 3, dotted
- lty = 4, dotdash
- lty = 5, longdash
- lty = 6, twodash

Untuk lebih memahaminya, pada sintaks berikut disajikan plot seluruh jenis garis tersebut beserta output yang dihasilkannya pada Gambar 3.27:

```
generateRLineTypes<-function(){
  oldPar<-par()
  par(font=2, mar=c(0,0,0,0))
  plot(1, pch="", ylim=c(0,6), xlim=c(0,0.7), axes = FALSE ,xlab="", ylab="")
  for(i in 0:6) lines(c(0.3,0.7), c(i,i), lty=i, lwd=3)
  text(rep(0.1,6), 0:6,
       labels=c("0.'blank'", "1.'solid'", "2.'dashed'", "3.'dotted'",
                "4.'dotdash'", "5.'longdash'", "6.'twodash'"))
  par(mar=oldPar$mar,font=oldPar$font )
}
generateRLineTypes()
```

3.3.6 Mengatur Axis Plot

Kita dapat melakukan pengaturan lebih jauh terhadap axis, seperti: menambahkan axis tambahan pada atas dan bawah frame, mengubah rentang nilai axis, serta kustomisasi *tick mark* pada nilai axis. Hal ini diperlukan karena fungsi grafik dasar `R` tidak dapat mengatur axis secara otomatis saat plot baru ditambahkan pada plot pertama dan rentang nilai plot baru lebih besar dibanding plot pertama, sehingga sebagian nilai plot baru tidak ditampilkan pada hasil akhir.

Untuk menambahkan axis pada `R` kita dapat menambahkan fungsi `axis()` setelah plot dilakukan. Format yang digunakan adalah sebagai berikut:

```
axis(side, at=NULL, labels=TRUE)
```

Catatan:

- **side**: nilai integer yang mengidikasikan posisi axix yang hendak ditambahkan. Nilai yang dapat dimasukkan adalah sebagai berikut:
  - 1: bawah
  - 2: kiri
  - 3: atas
  - 4: kanan.
- **at**: titik dimana *tick-mark* hendak digambarkan. Nilai yang dapat dimasukkan sama dengan `side`.
- **labels**: Teks label *tick-mark*. Dapat juga secara logis menentukan apakah anotasi harus dibuat pada *tick mark*.

Berikut contoh sintaks penerapan fungsi tersebut dan output yang dihasilkan pada Gambar 3.28:

```
# membuat vektor numerik
x <- c(1:4)
y <- x^2

# plot
plot(x, y, pch=18, col="red", type="b",
     frame=FALSE, xaxt="n") # Remove x axis

# menambahkan axis
# bawah
axis(1, 1:4, LETTERS[1:4], col.axis="blue")
# atas
axis(3, col = "darkgreen", lty = 2, lwd = 0.5)
```

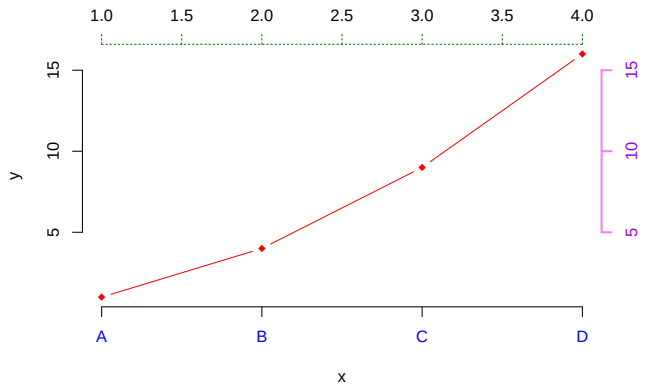

Figure 3.28: Modifikasi axis

```
# kanan
axis(4, col = "violet", col.axis = "dark violet", lwd = 2)
```

Kita dapat mengubah rentang nilai pada axis menggunakan fungsi `xlim()` dan `ylim()` yang menyatakan vektor nilai masimum dan minimum rentang. Selain itu kita dapat juga melakukan tranformasi baik pada sumbu x dan sumbu y. Berikut adalah argumen yang dapat ditambahkan pada fungsi grafik:

- **xlim**: limit nilai sumbu x dengan format: `xlim(min, max)`.
- **ylim**: limit nilai sumbu x dengan format: `ylim(min, max)`.

Untuk transformasi skala log, kita dapat menambahkan argumen berikut:

- **log=“x”**: transformasi log sumbu x.
- **log=“y”**: transformasi log sumbu y.
- **log=“xy”**: transformasi log sumbu x dan y.

Berikut adalah contoh sintaks penerapan argumen tersebut beserta output yang dihasilkan pada Gambar 3.29:

```
# membagi jendela grafik menjadi 1 baris dan 3 kolom
par(mfrow=c(1,3))

# membuat vektor numerik
x<-c(1:10); y<-x*x
```

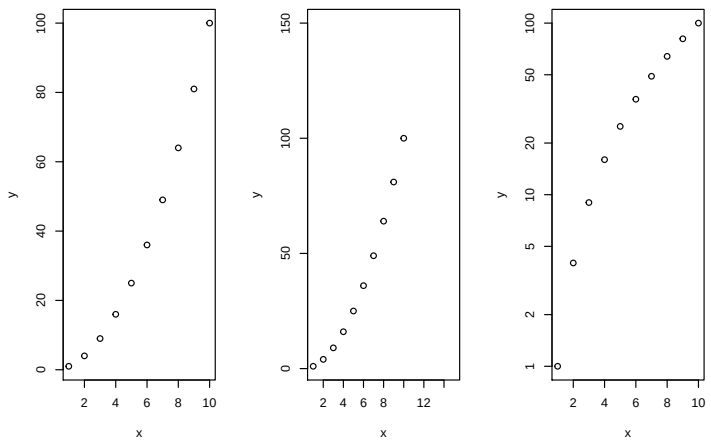

Figure 3.29: Mengubah rentang dan skala axis

```
# simple plot
plot(x, y)

# plot dengan pengaturan rentang skala
plot(x, y, xlim=c(1,15), ylim=c(1,150))

# plot dengan transformasi skala log
plot(x, y, log="y")
```

Kita dapat melakukan kustomisasi pada *tick mark*. Kustomisasi yang dapat dilakukan adalah merubah warna, *font style*, ukuran font, orientasi, serta menyembunyikan *tick mark*.

Argumen yang ditambahkan adalah sebagai berikut:

- **col.axis**: warna *tick mark*.
- **font.axis**: integer yang menunjukkan *font style*. Sama dengan pengaturan judul.
- **cex.axis**: pengaturan ukuran *tick mark*.
- **las**: mengatur orientasi *tick mark*. Nilai yang dapat dimasukkan adalah sebagai berikut:
  - **0**: paralel terhadap posisi axis (default)
  - **1**: selalu horizontal

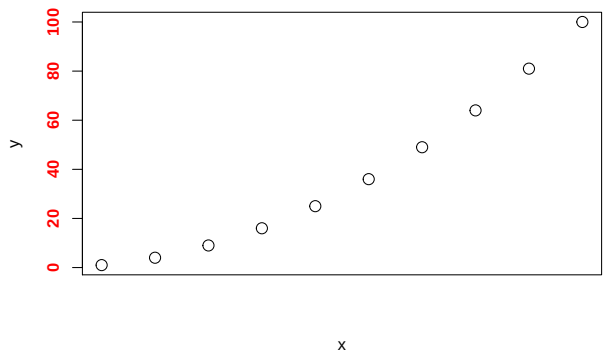

Figure 3.30: Kustomisasi tick mark

  - **2**: selalu perpendikular dengan posisi axis
  - **3**: selalu vertikal

- **xaxt** dan **yaxt**: karakter untuk menunjukkan apakah axis akan ditampilkan atau tidak. nilai dapat berupa “n”(sembunyika) dan “s”(tampilkan).

Berikut adalah contoh penerapan argumen tersebut beserta output pada Gambar 3.30:

```
# membuat vektor numerik
x<-c(1:10); y<-x*x

# plot
plot(x,y,
     # warna
     col.axis="red",
     # font style
     font.axis=2,
     # ukuran
     cex=1.5,
     # orientasi
     las=3,
     # sembunyikan sumbu x
     xaxt="n")
```

Figure 3.31: Nama warna

3.3.7 Mengatur Warna

Pada fungsi dasar R, warna dapat diatur dengan mengetikkan nama warna maupun kode hexadesimal. Selain itu kita juga dapat menamambahkan warna lain melalui library lain yang tidak dijelaskan pada chapter ini.

Untuk penggunaan warna hexadesima kita perlu mengetikkan "#" yang diukuti oleh 6 kode warna. Untuk memperlajari kode-kode dan warna yang dihasilkan, silahkan pembaca mengunjungi situs http://www.visibone.com/.

Pada sintaks berikut disajikan visualisasi nama-nama warna bawaan yang ada pada R. Output yang dihasilkan disajikan pada Gambar 3.31:

```
showCols <- function(cl=colors(), bg = "grey",
                     cex = 0.75, rot = 30) {
    m <- ceiling(sqrt(n <-length(cl)))
    length(cl) <- m*m; cm <- matrix(cl, m)
    require("grid")
    grid.newpage(); vp <- viewport(w = .92, h = .92)
    grid.rect(gp=gpar(fill=bg))
    grid.text(cm, x = col(cm)/m, y = rev(row(cm))/m, rot = rot,
              vp=vp, gp=gpar(cex = cex, col = cm))
}

# print 60 nama warna pertama
showCols(bg="gray20", cl=colors()[1:60], rot=30, cex=0.9)
```

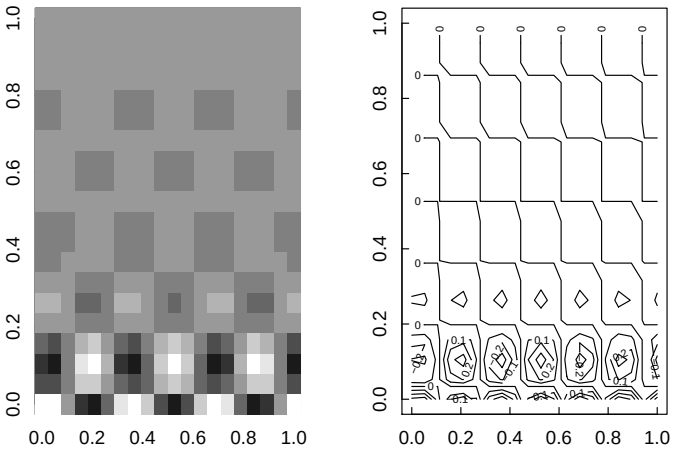

Figure 3.32: image plot (kiri) dan contour plot (kanan)

3.4 Plot Dua dan Tiga Dimensi

R dapat digunakan untuk memproduksi visualisasi pada skala 2 dan 3 dimensi. Untuk proyeksi 2 dimensi, fungsi yang digunakan adalah `image()` atau `contour()`. Untuk informasi lebih lanjut terkait fungsi tersebut pembaca dapat mengakses menu bantuan. Pada sintak berikut diberikan contoh bagaimana cara memproduksi visualisasi dua dimensi menggunakan kedua fungsi tersebut:

```
n <- 1:20
x <- sin(n)
y <- cos(n)*exp(-n/3)
z <- outer(x,y)
par(mar=c(3,3,1.5,1.5), mex=0.8, mgp=c(2,0.5,0), tcl=0.3)
par(mfrow=c(1,2))

# plot pertama
image(z, col=gray(1:10/10))

# plot kedua
contour(z)
```

```
par(mfrow=c(1,1))
```

Proyeksi 3 dimensi dapat dilakukan menggunakan fungsi `persp()`. Sudut penglihatan dapat diatur melalui argumen`theta` (sudut) dan `phi()` (rotasi).

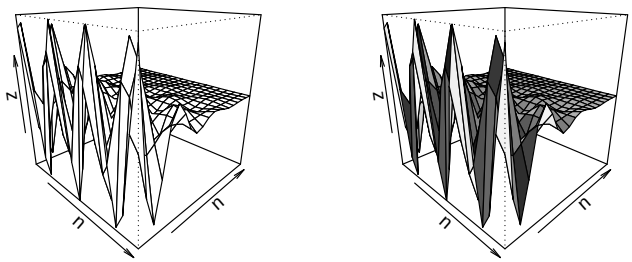

Figure 3.33: proyeksi 3 dimensi (kanan) dan proyeksi 3 dimensi dengan pewarnaan

Sintaks berikut merupakan contoh bagaimana cara menghasilkan visualisasi 3 dimensi dari data yang telah diproduksi sebelumnya:

```
par(mar=c(3,3,1.5,1.5), mex=0.8, mgp=c(2,0.5,0), tcl=0.3)
par(mfrow=c(1,2))

# plot pertama
persp(n,n,z, theta=45, phi=20)

# plot kedua
persp(n,n,z, theta=45, phi=20, shade=0.5)
```

```
par(mfrow=c(1,1))
```

3.5 Referensi

1. Maindonald, J.H. 2008. **Using R for Data Analysis and Graphics Introduction, Code and Commentary**. Centre for Mathematics and Its Applications Australian National University.
2. Scherber, C. 2007. **An introduction to statistical data analysis using R**. R_Manual Goettingen.
3. STHDA. **R Base Graphs**. http://www.sthda.com/english/wiki/r-base-graphs

4. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.

Chapter 4

Pemrograman dan Fungsi

Kita telah membahas dasar-dasar kalkulasi menggunakan `R` pada Chapter 2. Pada Chapter 4 kita akan membahas dasar pemrograman menggunakan `R`. Pada chapter ini kita juga akan membahas bagaimana kita dapat membentuk suatu fungsi menggunakan `R` untuk pekerjaan yang berulang-ulang.

4.1 Loop

Loop merupakan kode program yang berulang-ulang. *Loop* berguna saat kita ingin melakukan sebuah perintah yang perlu dijalankan berulang-ulang seperti melakukan perhitungan maupaun melakukan visualisasi terhadap banyak variabel secara serentak. Hal ini tentu saja membantu kita karena kita tidak perlu menulis sejumlah sintaks yang berulang-ulang. Kita hanya perlu mengatur *statement* berdasarkan hasil yang kita harapkan.

Pada `R` bentuk *loop* dapat bermacam-macam ("*for loop*","*while loop*", dll). `R` menyederhanakan bentuk *loop* ini dengan menyediakan sejumlah fungsi seperti `apply()`,`tapply()`, dll. Sehingga `loop` jarang sekali muncul dalam kode `R`. Sehingga `R` sering disebut sebagai *loopless loop*.

Meski *loop* jarang muncul bukan berarti kita tidak akan melakukannya. Terkadang saat kita melakukan komputasi statistik atau matematik dan belum terdapat paket yang mendukung proses tersebut, sering kali kita akan membuat sintaks sendiri berdasarkan algoritma metode tersebut. Pada algoritma tersebut sering pula terdapat *loop* yang diperlukan selama proses perhitungan. Secara sederhana diagram umum loop ditampilkan pada Gambar 4.1

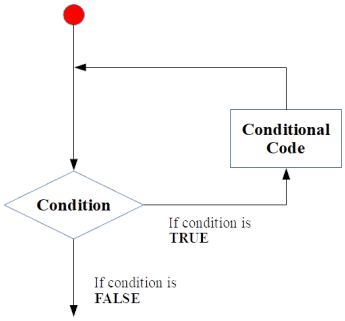

Figure 4.1: Diagram umum loop (sumber: Primartha, 2018).

4.1.1 For Loop

Mengulangi sebuah *statement* atau sekelompok *statement* sebanyak nilai yang ditentukan di awal. Jadi operasi akan terus dilakukan sampai dengan jumlah yang telah ditetapkan di awal atau dengan kata lain tes kondisi (Jika jumlah pengulangan telah cukup) hanya akan dilakukan di akhir. Secara sederhana bentuk dari *for loop* dapat dituliskan sebagai berikut:

```
for (value in vector){
  statements
}
```

Berikut adalah contoh sintaks penerapan *for loop*:

```
# Membuat vektor numerik
vektor <- c(1:5)

# loop
for(i in vektor){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Loop akan dimulai dari blok *statement* *for* sampai dengan `print(i)`. Berdasarkan *loop* pada contoh tersebut, *loop* hanya dilakukan sebanyak 5 kali sesuai dengan jumlah vektor yang ada.

4.1.2 While Loop

While loop merupakan loop yang digunakan ketika kita telah menetapkan *stop condition* sebelumnya. Blok *statement*/kode yang sama akan terus dijalankan sampai *stop condition* ini tercapai. *Stop condition* akan di cek sebelum melakukan proses *loop*. Berikut adalah pola dari *while loop* dapat dituliskan sebagai berikut:

```
while (test_expression){
  statement
}
```

Berikut adalah contoh penerapan dari *while loop*:

```
coba <- c("Contoh")
counter <- 1

# loop
while (counter<5){
  # print vektor
  print(coba)
  # tambahkan nilai counter sehingga proses terus berlangsung sampai counter = 5
  counter <- counter + 1
}
```

```
## [1] "Contoh"
## [1] "Contoh"
## [1] "Contoh"
## [1] "Contoh"
```

Loop akan dimulai dari blok *statement while* sampai dengan *counter* <- 1. *Loop* hanya akan dilakukan sepanjang nilai *counter* < 5.

4.1.3 Repeat Loop

Repeat loop akan menjalankan *statement*/kode yang sama berulang-ulang hingga *stop condition* tercapai. Berikut adalah pola dari *repeat loop*.

```
repeat {
  commands
  if(condition){
    break
  }
}
```

Berikut adalah contoh penerapan dari *repeat loop*:

```
coba <- c("contoh")
counter <- 1
repeat {
  print(coba)
  counter <- counter + 1
  if(counter < 5){
break
  }
}
```

```
## [1] "contoh"
```

Loop akan dimulai dari blok *statement while* sampai dengan *break*. *Loop* hanya akan dilakukan sepanjang nilai *counter* < 5. Hasil yang diperoleh berbeda dengan *while loop*, dimana kita memperoleh 4 buah kata "contoh". Hal ini disebabkan karena *repeat loop* melakukan pengecekan *stop condition* tidak di awal loop seperti *while loop* sehingga berapapun nilainya, selama nilainya sesuai dengan *stop condition* maka *loop* akan dihentikan. Hal ini berbeda dengan *while loop* dimana proses dilakukan berulang-ulang sampai jumlahnya mendekati *stop condition*.

4.1.4 Break

Break sebenarnya bukan bagian dari *loop*, namun sering digunakan dalam *loop*. *Break* dapat digunakan pada *loop* manakala dirasa perlu, yaitu saat kondisi yang disyaratkan pada *break* tercapai.

Berikut adalah contoh penerapan *break* pada beberapa jenis *loop*.

```
# for loop
a = c(2,4,6,8,10,12,14)
for(i in a){
  if(i>8){
    break
  }
  print(i)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
```

```
# while loop
a = 2
b = 4
while(a<7){
  print(a)
  a = a +1
  if(b+a>10){
    break
  }
}
```

```
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

```
# repeat loop
a = 1
repeat{
  print(a)
  a = a+1
  if(a>6){
    break
  }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
```

4.2 Loop Menggunakan Apply Family Function

Penggunaan loop sangat membantu kita dalam melakukan proses perhitungan berulang. Namun, metode ini tidak cukup ringkas dalam penerapannya dan perlu penulisan sintaks yang cukup panjang untuk menyelesaikan sebuah kasus yang kita inginkan. Berikut adalah sebuah sintaks yang digunakan untuk menghitung nilai mean pada suatu dataset:

```r
# subset data iris
sub_iris <- iris[,-5]
# membuat vektor untuk menyimpan hasil loop
a <- rep(NA,4)
# loop
for(i in 1:length(sub_iris)){
  a[i]<-mean(sub_iris[,i])
}
# print
a
```

```
## [1] 5.843333 3.057333 3.758000 1.199333
```

```r
class(a) # cek kelas objek
```

```
## [1] "numeric"
```

Metode alternatif lain untuk melakukan loop suatu fungsi adalah dengan menggunakan Apply function family. Metode ini memungkinkan kita untuk melakukan loop suatu fungsi tanpa perlu menuliskan sintaks loop. Berikut adalah beberapa fungsi dari apply family yang nantinya akan sering kita gunakan:

- `apply()`: fungsi generik yang mengaplikasikan fungsi kepada kolom atau baris pada matriks atau secara lebih general aplikasi dilakukan pada dimensi untuk jenis data array.
- `lapply()`: fungsi apply yang bekerja pada jenis data list dan memberikan output berupa list juga.
- `sapply()`: bentuk sederhana dari lapply yang menghasilkan output berupa matriks atau vektor.
- `vapply()`: disebut juga *verified apply* (memungkinkan untuk menghasilkan output dengan jenis data yang telah ditentukan sebelumnya).
- `tapply()`: *tagged apply* dimana dimana tag menentukan subset dari data.

4.2.1 Apply

Fungsi `apply()` bekerja dengan jenis data matrik atau array (jenis data homogen). Kita dapat melakukan spesifikasi apakah suatu fungsi hanya akan bekerja pada kolom saja, baris saja atau keduanya. Format fungsi ini adalah sebagai berikut:

```
apply(X, MARGIN, FUN, ...)
```

Catatan:

- **X**: matriks atau array
- **MARGIN**: menentukan bagaimana fungsi bekerja terhadap matriks atau array. Jika nilai yang diinputkan 1, maka fungsi akan bekerja pada masing-masing baris pada matriks. Jika nilainya 2, maka fungsi akan bekerja pada tiap kolom pada matriks.
- **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function.
- **...**: opsional argumen pada fungsi yang digunakan.

Berikut adalah contoh bagaimana aplikasi fungsi tersebut pada matriks:

```
## membuat matriks
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
x # print
```

```
##      x1 x2
## [1,]  3  4
## [2,]  3  3
## [3,]  3  2
## [4,]  3  1
## [5,]  3  2
## [6,]  3  3
## [7,]  3  4
## [8,]  3  5
```

```
class(x) # cek kelas objek
```

```
## [1] "matrix"
```

```
## menghitung mean masing-masing kolom
apply(x, MARGIN=2 ,FUN=mean, trim=0.2, na.rm=TRUE)
```

```
## x1 x2
##  3  3
```

```
## menghitung range pada masing-masing baris
## menggunakan user define function
apply(x, MARGIN=1,
      FUN=function(x){
        max(x)-min(x)
      })
```

```
## [1] 1 0 1 2 1 0 1 2
```

4.2.2 lapply

Fungsi ini melakukan loop fungsi terhadap input data berupa list. Output yang dihasilkan juga merupakan list dengan panjang list yang sama dengan yang diinputkan. Format yang digunakan adalah sebagai berikut:

```
lapply(X, FUN, ...)
```

Catatan:

- **X**: vektor, data frame atau list
- **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
- **...**: opsional argumen pada fungsi yang digunakan.

Berikut adalah contoh penerapan fungsi lapply:

```
## Membuat list
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
x # print
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $beta
## [1]  0.04978707  0.13533528  0.36787944  1.00000000  2.71828183  7.38905610
## [7] 20.08553692
##
## $logic
## [1]  TRUE FALSE FALSE  TRUE
```

```r
class(x) # cek kelas objek
```

```
## [1] "list"
```

```r
## Menghitung nilai mean pada masing-masing baris lits
lapply(x, FUN=mean)
```

```
## $a
## [1] 5.5
##
## $beta
## [1] 4.535125
##
## $logic
## [1] 0.5
```

```r
## Menghitung mean tiap kolom dataset iris
lapply(iris, FUN=mean)
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```

```
## $Sepal.Length
## [1] 5.843333
##
## $Sepal.Width
## [1] 3.057333
##
## $Petal.Length
## [1] 3.758
##
## $Petal.Width
## [1] 1.199333
##
## $Species
## [1] NA
```

```r
## Mengalikan elemen vektor dengan suatu nilai
y <- c(1:5)
lapply(y, FUN=function(x){x*5})
```

```
## [[1]]
```

```
## [1] 5
##
## [[2]]
## [1] 10
##
## [[3]]
## [1] 15
##
## [[4]]
## [1] 20
##
## [[5]]
## [1] 25
```

```
## Mengubah output menjadi vektor
unlist(lapply(y, FUN=function(x){x*5}))
```

```
## [1]  5 10 15 20 25
```

4.2.3 sapply

Fungsi `sapply()` merupakan bentuk lain dari fungsi `lapply()`. Perbedaanya terletak pada output default yang dihasilkan. Secara default `sapply()` menerima input utama berupa list (dapat pula dataframe atau vektor), namun tidak seperti `lapply()` jenis data output yang dihasilkan adalah vektor. Untuk mengubah output menjadi list perlu argumen tambahan berupa `simplify=FALSE`. Format fungsi tersebut adalah sebagai berikut:

```
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

Catatan:

- **X**: vektor, data frame atau list
- **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
- **...**: opsional argumen pada fungsi yang digunakan.
- **simplify**: logical. Jika nilainya `TRUE` maka output yang dihasilkan adalah bentuk sederhana dari vektor, matrix atau array.
- **USE.NAMES**: jika list memiliki nama pada setiap elemennya, maka nama elemen tersebut akan secara default ditampilkan.

Berikut adalah contoh penerapannya:

```
## membuat list
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))

## menghitung nilai mean setiap elemen
sapply(x, FUN=mean)
```

```
##        a     beta    logic
## 5.500000 4.535125 0.500000
```

```
## menghitung nilai mean dengan output list
sapply(x, FUN=mean, simplify=FALSE)
```

```
## $a
## [1] 5.5
##
## $beta
## [1] 4.535125
##
## $logic
## [1] 0.5
```

```
## summary objek dataframe
sapply(mtcars, FUN=summary)
```

```
##              mpg    cyl     disp       hp     drat      wt     qsec     vs
## Min.    10.40000 4.0000  71.1000  52.0000 2.760000 1.51300 14.50000 0.0000
## 1st Qu. 15.42500 4.0000 120.8250  96.5000 3.080000 2.58125 16.89250 0.0000
## Median  19.20000 6.0000 196.3000 123.0000 3.695000 3.32500 17.71000 0.0000
## Mean    20.09062 6.1875 230.7219 146.6875 3.596563 3.21725 17.84875 0.4375
## 3rd Qu. 22.80000 8.0000 326.0000 180.0000 3.920000 3.61000 18.90000 1.0000
## Max.    33.90000 8.0000 472.0000 335.0000 4.930000 5.42400 22.90000 1.0000
##              am   gear   carb
## Min.    0.00000 3.0000 1.0000
## 1st Qu. 0.00000 3.0000 2.0000
## Median  0.00000 4.0000 2.0000
## Mean    0.40625 3.6875 2.8125
## 3rd Qu. 1.00000 4.0000 4.0000
## Max.    1.00000 5.0000 8.0000
```

```
## summary objek list
a <- list(mobil=mtcars, anggrek=iris)
sapply(a, FUN=summary)
```

```
## $mobil
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000  
##
## $anggrek
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##
##
##
```

4.2.4 vapply

Funsgi ini merupakan bentuk lain dari `sapply()`. Bedanya secara kecepatan proses fungsi ini lebih cepat dari `sapply()`. Hal yang menarik dari fungsi ini kita dapat menambahkan argumen `FUN.VALUE`. pada argumen ini kita memasukkan vektor berupa output fungsi yang diinginkan. Perbedaan lainnya adalah output

yang dihasilkan hanya berupa matriks atau array. Format dari fungsi ini adalah sebagai berikut:

```
vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)
```

Catatan:

- **X**: vektor, data frame atau list
- **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
- **FUN.VALUE**: vektor, template dari return value FUN.
- **...**: opsional argumen pada fungsi yang digunakan.
- **USE.NAMES**: jika list memiliki nama pada setiap elemennya, maka nama elemen tersebut akan secara default ditampilkan.

Berikut adalah contoh penerapannya:

```
## membuat list
x <- sapply(3:9, seq)
x # print
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 1 2 3 4
##
## [[3]]
## [1] 1 2 3 4 5
##
## [[4]]
## [1] 1 2 3 4 5 6
##
## [[5]]
## [1] 1 2 3 4 5 6 7
##
## [[6]]
## [1] 1 2 3 4 5 6 7 8
##
## [[7]]
## [1] 1 2 3 4 5 6 7 8 9
```

```
## membuat ringkasan data pada tiap elemen list
vapply(x, fivenum,
       c(Min. = 0, "1st Qu." = 0,
         Median = 0, "3rd Qu." = 0, Max. = 0))
```

```
##         [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## Min.     1.0  1.0    1  1.0  1.0  1.0    1
## 1st Qu.  1.5  1.5    2  2.0  2.5  2.5    3
## Median   2.0  2.5    3  3.5  4.0  4.5    5
## 3rd Qu.  2.5  3.5    4  5.0  5.5  6.5    7
## Max.     3.0  4.0    5  6.0  7.0  8.0    9
```

```
## membuat ringkasan data pada tiap kolom dataframe
vapply(mtcars, summary,
       c(Min. = 0, "1st Qu." = 0,
         Median = 0, "3rd Qu." = 0, Max. = 0, Mean=0))
```

```
##             mpg    cyl     disp       hp     drat      wt     qsec     vs
## Min.    10.40000 4.0000  71.1000  52.0000 2.760000 1.51300 14.50000 0.0000
## 1st Qu. 15.42500 4.0000 120.8250  96.5000 3.080000 2.58125 16.89250 0.0000
## Median  19.20000 6.0000 196.3000 123.0000 3.695000 3.32500 17.71000 0.0000
## 3rd Qu. 20.09062 6.1875 230.7219 146.6875 3.596563 3.21725 17.84875 0.4375
## Max.    22.80000 8.0000 326.0000 180.0000 3.920000 3.61000 18.90000 1.0000
## Mean    33.90000 8.0000 472.0000 335.0000 4.930000 5.42400 22.90000 1.0000
##             am   gear   carb
## Min.    0.00000 3.0000 1.0000
## 1st Qu. 0.00000 3.0000 2.0000
## Median  0.00000 4.0000 2.0000
## 3rd Qu. 0.40625 3.6875 2.8125
## Max.    1.00000 4.0000 4.0000
## Mean    1.00000 5.0000 8.0000
```

4.2.5 tapply

Fungsi ini sangat berguna jika pembaca ingin menghitung suatu nilai misalnya mean berdasarkan grup data atau factor. Format fungsi ini adalah sebagi berikut:

```
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

Catatan:

- **X**: vektor, data frame atau list

- **INDEX**: list satu atau beberapa factor yang memiliki panjang sama dengan X.
- **FUN**: fungsi yang akan digunakan. Fungsi yang dapat digunakan dapat berupa fungsi dasar matematika atau statistika, serta user define function. Subset juga dimungkinkan pada fungsi ini.
- **...**: opsional argumen pada fungsi yang digunakan.
- **simplify**: logical. Jika nilainya `TRUE` maka output yang dihasilkan adalah bentuk skalar.

Berikut adalah contoh penerapannya:

```
## membuat tabel frekuensi
groups <- as.factor(rbinom(32, n = 5, prob = 0.4))

tapply(groups, groups, length)
```

```
## 12 13 16
##  2  2  1
```

```
# atau
table(groups)
```

```
## groups
## 12 13 16
##  2  2  1
```

```
## membuat tabel kontingensi
# menghitung jumlah breaks berdasarkan faktor jenis wool
# dan tensi level
tapply(X=warpbreaks$breaks, INDEX=warpbreaks[,-1], FUN=sum)
```

```
##     tension
## wool   L   M   H
##    A 401 216 221
##    B 254 259 169
```

```
# menghitung mean panjang gigi babi hutan berdasarkan
# jenis suplemen dan dosisnya
tapply(ToothGrowth$len, ToothGrowth[,-1], mean)
```

```
##     dose
## supp   0.5     1     2
##   OJ 13.23 22.70 26.06
##   VC  7.98 16.77 26.14
```

```
# menghitung mpg minimum berdasarkan jumlah silinder pada mobil
tapply(mtcars$mpg, mtcars$cyl, min, simplify=FALSE)
```

```
## $`4`
## [1] 21.4
##
## $`6`
## [1] 17.8
##
## $`8`
## [1] 10.4
```

4.3 Decision Making

Decicion Making atau sering disebut sebagai *if then else statement* merupakan bentuk percabagan yang digunakan manakala kita ingin agar program dapat melakukan pengujian terhadap syarat kondisi tertentu. Pada Tabel 4.1 disajikan daftar percabangan yang digunakan pada R.

Table 4.1: Daftar percabangan pada R.

Statement	Keterangan
if statement	*if statement* hanya terdiri atas sebuah ekspresi *Boolean*, dan diikuti satu atau lebih *statement*
if...else statement	*if else statement* terdiri atas beberapa buah ekspresi *Boolean*. Ekspressi *Boolean* berikutnya akan dijalankan jika ekspresi *Boolan sebelumnya bernilai FALSE
switch statement	*switch statement* digunakan untuk mengevaluasi sebuah variabel beberapa pilihan

4.3.1 if statement

Pola *if statement* disajikan pada Gambar 4.2

Berikut adalah contoh penerapan *if statement*:

```
x <- c(1:5)
if(is.vector(x)){
  print("x adalah sebuah vector")
}
```

```
## [1] "x adalah sebuah vector"
```

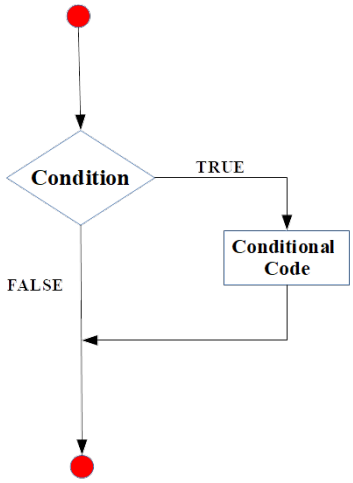

Figure 4.2: Diagram if statement (sumber: Primartha, 2018).

4.3.2 if else statement

Pola dari *if else statement* disajikan pada Gambar 4.3

Berikut adalah contoh penerapan *if else statement*:

```
x <- c("Andi","Iwan", "Adi")
if("Rina" %in% x){
  print("Rina ditemukan")
} else if("Adi" %in% x){
  print("Adi ditemukan")
} else{
  print("tidak ada yang ditemukan")
}
```

```
## [1] "Adi ditemukan"
```

4.3.3 switch statement

Pola dari *switch statement* disajikan pada Gambar 4.4

Berikut adalah contoh penerapan *switch statement*:

```
y = 3

x = switch(
```

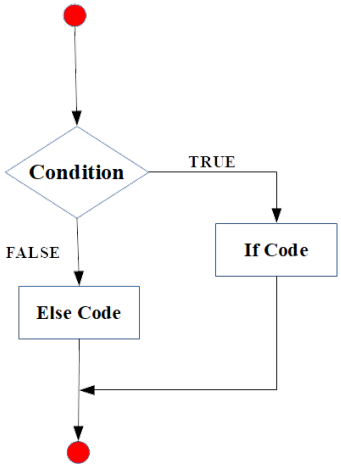

Figure 4.3: Diagram if else statement (sumber: Primartha, 2018).

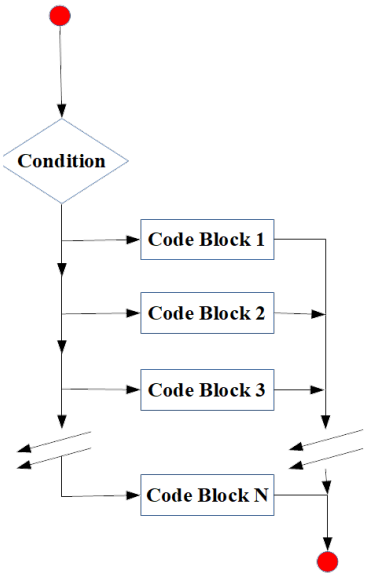

Figure 4.4: Diagram switch statement (sumber: Primartha, 2018).

```
  y,
  "Selamat Pagi",
  "Selamat Siang",
  "Selamat Sore",
  "Selamat Malam"
)

print(x)
```

```
## [1] "Selamat Sore"
```

4.4 Fungsi

Fungsi merupakan sekumpulan instruksi atau *statement* yang dapat melakukan tugas khusus. Sebagai contoh fungsi perkalian untuk menyelesaikan operasi perkalian, fungsi pemangkatan hanya untuk operasi pemangkatan, dll.

Pada `R` terdapat 2 jenis fungsi, yaitu: *build in fuction* dan *user define function*. *build in fnction* merupakan fungsi bawaan `R` saat pertama kita menginstall `R`. Contohnya adalah `mean()`, `sum()`, `ls()`, `rm()`, dll. Sedangkan *user define fuction* merupakan fungsi-fungsi yang dibuat sendiri oleh pengguna.

Fungsi-fungsi buatan pengguna haruslah dideklarasikan (dibuat) terlebih dahulu sebelum dapat dijalankan. Pola pembentukan fungsi adalah sebagai berikut:

```
function_name <- function(argument_1, argument_2, ...){
  function body
}
```

Catatan:

- **function_name** : Nama dari fungsi `R`. `R` akan menyimpan fungsi tersebut sebagai objek
- **argument_1, argument_2, ...** : *Argument* bersifat opsional (tidak wajib). *Argument* dapat digunakan untuk memberi inputan kepada fungsi
- **function body** : Merupakan inti dari fungsi. Fuction body dapat terdiri atas 0 statement (kosong) hingga banyak statement.
- **return** : Fungsi ada yang memiliki *output* atau *return value* ada juga yang tidak. Jika fungsi memiliki *return value* maka *return value* dapat diproses lebih lanjut

Berikut adalah contoh penerapan *user define function*:

```
# Fungsi tanpa argument
bilang <- function(){
  print("Hello World!!")
}

# Print
bilang()
```

```
## [1] "Hello World!!"
```

```
# Fungsi dengan argumen
tambah <- function(a,b){
  print(a+b)
}

# Print
tambah(5,3)
```

```
## [1] 8
```

```
# Fungsi dengan return value
kali <- function(a,b){
  return(a*b)
}

# Print
kali(4,3)
```

```
## [1] 12
```

4.5 Debugging

Sering kali fungsi atau sintaks yang kita tulis menghasilkan error sehingga output yang kita harapkan tidak terjadi. *Debugging* merupakan langkah untuk mengecek error yang terjadi. Untuk lebih memahami proses *debugging*, berikut penulis sajikan contoh error pada suatu fungsi dapat terjadi:

```
f1 <- function(x){
  xsq <- x^2
  xsqminus4 <- xsq - 4
```

```
  print(xsqminus4)
  log(xsqminus4-4)
}

f1(6:1)
```

```
## [1] 32 21 12  5  0 -3

## Warning in log(xsqminus4 - 4): NaNs produced

## [1] 3.332205 2.833213 2.079442 0.000000      NaN      NaN
```

Untuk mengecek error yang terjadi dari sintaks tersebut, kita dapat menggunakan fungsi `debug()`. Pembaca tinggal memasukkan nama fungsi kedalam fungsi `debug()`. Fungsi tersebut akan secara otomatis akan menampilkan hasil samping dari pengaplikasian fungsi `f1()` untuk melihat sumber atau tahapan dimana error mulai muncul.

```
debug(f1)
f1(1:6)
```

```
## debugging in: f1(1:6)
## debug at <text>#1: {
##     xsq <- x^2
##     xsqminus4 <- xsq - 4
##     print(xsqminus4)
##     log(xsqminus4 - 4)
## }
## debug at <text>#2: xsq <- x^2
## debug at <text>#3: xsqminus4 <- xsq - 4
## debug at <text>#4: print(xsqminus4)
## [1] -3  0  5 12 21 32
## debug at <text>#5: log(xsqminus4 - 4)

## Warning in log(xsqminus4 - 4): NaNs produced

## exiting from: f1(1:6)

## [1]      NaN      NaN 0.000000 2.079442 2.833213 3.332205
```

Berdasarkan hasil *debugging*, `NaN` (**missing value**) muncul pada tahapan *debug* ke-4 (pembaca dapat melakukan enter terus menerus sampai proses *debug* selesai). Hal ini disebabkan karena terdapat nilai negatif pada objek `xsqminu4-4` yang selanjutnya dilakukan transformasi logaritmik. Untuk menghentikan proses *debugging* pembaca dapat mengetikkan `undebug(f1)`.

4.6 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
2. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
3. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.

Chapter 5

Pengantar Metode Numerik

Chapter ini memberikan pengantar bagi pembaca untuk mengenal terlebih dahulu mengenai metode numerik. Pada *chapter* ini akan dibahas mengenai apa itu metode numerik, perbedaannya dengan metode analitik, dan analisis error.

5.1 Mengenal Metode Numerik

Metode numerik merupakan teknik penyelesaian permsalahn yang diformulasikan secara matematis dengan menggunakan operasi hitungan (aritmatik) yaitu operasi tambah, kurang, kali, dan bagi. Metode ini digunakan karena banyak permasalahan matematis tidak dapat diselesaikan menggunakan metode analitik. Jikapun terdapat penyelesaiannya secara analitik, proses penyelesaiaannya sering kali cukup rumit dan memakan banyak waktu sehingga tidak efisien.

Terdapat keuntungan dan kerugian terkait penggunaan metode numerik. Keuntungan dari metode ini antara lain:

1. Solusi persoalan selalu dapat diperoleh.
2. Dengan bantuan komputer, perhitungan dapat dilakukan dengan cepat serta hasil yang diperoleh dapat dibuat sedekat mungkin dengan nilai sesungguhnya.
3. Tampilan hasil perhitungan dapat disimulasikan.

Adapun kelemahan metode ini antara lain:

1. Nilai yang diperoleh berupa pendekatan atau hampiran.
2. Tanpa bantuan komputer, proses perhitungan akan berlangsung lama dan berulang-ulang.

5.1.1 Perbedaan Antara Metode Numerik dan Analitik

Perbedaan antara metode numerik dan metode analitik dapat dijelaskan sebagai berikut:

1. Solusi metode numerik selalu berbentuk angka, sedangkan solusi metode analitik dapat berbentuk fungsi matematik yang selanjutnya dapat dievaluasi untuk menghasilkan nilai dalam bentuk angka.
2. Solusi dari metode numerik berupa hampiran, sedangkan metode analitik berupa solusi sejati. Kondisi ini berakibat pada nilai error metode analitik adalah 0, sedangkan metode numerik $\neq 0$.
3. Metode analitik cocok untuk permasalahan dengan model terbatas dan sederhana, sedangkan metode numerik cocok dengan semua jenis permasalahan.

5.1.2 Tahapan Penyelesaian Menggunakan Metode Numerik

Terdapat beberapa tahapan dalam menyelesaikan suatu permasalahan dengan metode numerik. Tahapan-tahapan tersebut antara lain:

- Pemodelan

Persoalan dunia nyata dimodelkan ke dalam persamaan matematika. Persamaan matematika yang terbentuk dapat berupa persamaan linier, non-linier, dan sebagainya sesuai dengan persoalan yang dihadapi.

- Penyederhanaan Model

Model matematika yang dihasilkan dari tahap 1 mungkin saja terlalu kompleks. Semakin kompleks suatu model, semakin rumit penyelesaiaannya, sehingga model perlu disederhanakan.

Seberapa sederhana model yang akan kita buat? tergantung pada permasalahan apa yang hendak pembaca selesaikan. Model yang terlalu sederhana akan tidak cocok digunakan untuk digunakan sebagai pendekatan sistem nyata atau lingkungan yang begitu kompleks. Penyederhanaan dapat berupa asumsi sejumlah variabel yang terlibat tidak signifikan, atau asumsi kondisi reaktor (*steady* atau *non-steady*).

- Formulasi Numerik

Setelah model matematika sederhana diperoleh, tahap selanjutnya adalah memformulasikan model matematika secara numerik. Tahapan ini terdiri atas:

```
+ menentukan metode numerik yang akan dipakai bersama-sama dengan analisis galat (error) awal.
+ menyusun algoritma dari metode numerik yang dipilih.
```

- Pemrograman

Tahap selanjutnya adalah menerjemahkan algoritma ke dalam program komputer. Pada tahapan ini pembaca bisa memilih bahasa pemrograman yang pembaca kuasai.

Dalam buku ini kita hanya akan berfokus pada bahasa pemrograman `R`. Pembaca dapat menggunakan bahasa pemrograman lain selain dari buku ini. Pembaca hanya perlu memperhatikan bagaimana penulis membangun algoritma penyelesaian dan memtransfernya menjadi bentuk sintaks `R`. Dari sintaks tersebut pembaca dapat melihat bagaimana meletakakkan tiap tahapan algoritma menjadi sintaks pada bahasa pemrograman.

- Operasional

Sebelum digunakan dengan data sesungguhnya, program komputer perlu dilakukan uji coba dengan data simulasi dan dievaluasi hasilnya. jika hasil keluaran diyakini sudah sesuai, baru dioperasikan dengan data yang sesungguhnya.

- Evaluasi

Bila program sudah selesai dijalankan dengan data yang sesungguhnya, maka hasil yang diperoleh dilakukan interpretasi, meliputi analisis hasil keluaran dan membandingkannya dengan prinsip dasar dan hasil-hasil empriik untuk menaksir kualitas soluasi numerik termasuk keputusan untuk menjalankan kembali progrma dengan memperoleh hasil yang lebih baik.

5.2 Akurasi dan Presisi

Perhatikan Gambar 5.1 berikut:

Pada Gambar 5.1 terdapat 4 buah kondisi ketika kita menembakkan beberapa perluru pada sebuah sasaran. Tujuan kita disini adalah untuk menembak bagian tengah sasaran tersebut.

Pada Gambar (a) dan (c) pada Gambar 5.1 merupakan gambar yang menunjukkan seseorang telah berhasil mengenai bagian tengah sasaran tersebut dapat kita katakan pula tembakan pada kedua gambar tersebut akurat. Akurat dalam

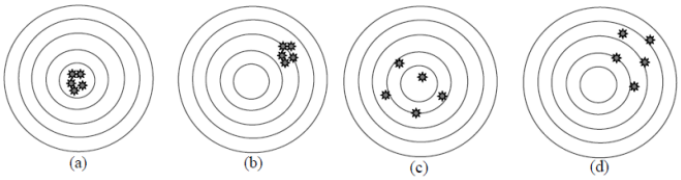

Figure 5.1: 4 ilustrasi terkait akurasi dan presisi

hal ini dapat diartikan suatu kondisi dimana kedekatan lubang peluru dengan pusat sasaran. Secara umum akurasi diartikan sebagai tingkat kedekatan pengukuran kuantitas terhadap nilai sebenarnya.

Terdapat dua buah cara untuk mengukur akurasi. Metode pengukuran akurasi antara lain: error absolut dan error relatif.

Error absolut merupakan nilai absolut dari selisih antara nilai sebenarnya x dengan nilai observasi x'. Error absolut dapat dituliskan menggunakan Persamaan (5.1).

$$\epsilon_A = |x - x'| \tag{5.1}$$

Pengukuran lain yang sering digunakan untuk mengukur akurasi adalah error relatif. Berbeda dengan error absolut, error relatif membagi selisih antara nilai sebenarnya x dan nilai observasi x' dengan nilai sebenarnya. Hasil yang diperoleh merupakan nilai tanpa satuan. Persamaan error relatif disajikan pada Persamaan (5.2).

$$\epsilon_R = \left| \frac{x - x'}{x} \right| \tag{5.2}$$

Dalam suatu pengukuran, hal lain yang perlu diperhatikan selain akurasi adalah presisi. Presisi adalah sejauh mana pengulangan pengukuran dalam kondisi yang tidak berubah mendapat hasil yang sama. Berdasarkan Gambar 5.1, Gambar (a) dan (b) menunjukkan kepresisian yang tinggi. Hal ini terlihat dari jarak antara lubang peluru yang saling berdekatan dan mengelompok.

Berdasarkan Gambar 5.1 dapat kita simpulkan bahwa dalam suatu sistem pengukuran akan terdapat 4 buah kondisi. Pengukuran akurat dan presisi (Gambar (a)), tidak akurat namun presisi (Gambar (b)), akurat namun tidak presisi (Gambar (c)), dan tidak akurat serta tidak presisi (Gambar (d)).

Dari kondisi-kondisi tersebut, akan meuncul yang dinamakan error. Dalam analisa numerik error atau kesalahan menjadi hal yang perlu diperhatikan.

5.3 Error Numerik

Kesalahan numerik merupakan error atau kesalahan yang timbul akibat adanya proses pendekatan atau hampiran. Kesalahan numerik terjadi karena tiga hal, antara lain:

- **Kesalahan bawaan (*inherent error*)**, merupakan kesalahan data yang timbul akibat adanya pengkuran, *human error* seperti kesalahan pencatatan, atau tidak memahami hukum-hukum fisik dari data yang diukur.
- **Kesalahan pembulatan (*round-off error*)**, adalah kesalahan yang terjadi karena adanya pembulatan. Contoh: 3,142857143… menjadi 3,14.
- **Kesalahan pemotongan (*truncation error*)**, adalah kesalahan yang ditimbulkan pada saat dilakukan pengurangan jumlah angka signifikan.

Kesalahan atau error dapat diukur menggunakan Persamaan (5.1) dan Persamaan (5.2) yang telah penulis jelaskan pada Chapter 5.2.

5.4 Referensi

1. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
2. Sidiq, M. Tanpa Tahun. **Materi Kuliah Metode Numerik**. Repository Universitas Dian Nuswantoro.
3. Subakti, I. 2006. **Metode Numerik**. Institut Teknologi Sepuluh Nopember.
4. Sutarno,H., Rachmatin,D. 2008. **Hands Out Metode Numerik**. Universitas Pendidikan Indonesia.

Chapter 6

Aljabar Linier

Pada *chapter* ini penulis akan menjelaskan mengenai cara untuk menyelesaikan sistem persamaan linier. Adapun yang akan dibahas pada *chapter* ini antara lain:

- operasi Vektor dan matriks
- Metode Eliminasi Gauss
- Metode Dekomposisi matriks
- Studi Kasus

6.1 Vektor dan matriks

Pada Chapter 2.7 dan Chapter 2.8 telah dijelaskan sekilas bagaimana cara melakukan operasi pada vektor dan matriks. Pada *chapter* ini, penulis akan menambahkan operasi-operasi lain yang dapat dilakukan pada vektor dan matriks. Dasar-dasar operasi ini selanjutnya akan digunakan sebagai dasar menyusun algoritma penyelesaian sistem persamaan linier.

6.1.1 Operasi Vektor

Misalkan saja diberikan vektor u dan v yang ditunjukkan pada Persamaan (6.1).

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} dan\ v\ = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{6.1}$$

Jika kita menambahkan atau mengurangkan nilai elemen vektor dengan suatu skalar (konstanta yang hanya memiliki besaran), maka operasi penjumlahan/pengurangan akan dilakukan pada setiap elemen vektor.

$$u \pm x = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \pm x = \begin{bmatrix} u_1 \pm x \\ u_2 \pm x \\ \vdots \\ u_n \pm x \end{bmatrix} \tag{6.2}$$

Jika kita melakukan penjumlahan pada vektor u dan v, maka operasi akan terjadi pada masing-masing elemen dengan indeks yang sama.

$$u \pm v = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \pm v \ = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 \pm v_1 \\ u_2 \pm v_2 \\ \vdots \\ u_n \pm v_n \end{bmatrix} \tag{6.3}$$

Untuk lebih memahami operasi tersebut, berikut penulis berikan contoh penerapannya pada R:

```
u <- seq(1,5)
v <- seq(6,10)

# penjumlahan
u+v
```

```
## [1]  7  9 11 13 15
```

```
# penguranga
u-v
```

```
## [1] -5 -5 -5 -5 -5
```

Bagaimana jika kita melakukan operasi dua vektor, dimaana salah satu vektor memiliki penjang yang berbeda?. Untuk memnjawab hal tersebut, perhatikan sintaks berikut:

```
x <- seq(1,2)
u+x
```

```
## Warning in u + x: longer object length is not a multiple of shorter object
## length
```

```
## [1] 2 4 4 6 6
```

Berdasarkan contoh tersebut, `R` akan mengeluarkan peringatan yang menunjukkan operasi dilakukan pada vektor dengan panjang berbeda. `R` akan tetap melakukan perhitungan dengan menjumlahkan kembali vektor u yang belum dijumlahkan dengan vektor x sampai seluruh elemen vektor u dilakukan operasi penjumlahan.

Operasi lain yang dapat dilakukan pada vektor adalah menghitung *inner product* dan panjang vektor. Inner product dihitung menggunakan Persamaan (6.4).

$$u.v = \sum_{i=1}^{n} u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \tag{6.4}$$

Panjang vektor atau vektor yang telah dinormalisasi dihitung menggunakan Persamaan (6.5)

$$|u| = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2} \tag{6.5}$$

Berikut adalah contoh bagaimana cara menghitung `inner product` dan panjang vektor menggunakan `R`:

```
# inner product
u%*%v
```

```
##      [,1]
## [1,]  130
```

```
# panjang vektor u
sqrt(sum(u*u))
```

```
## [1] 7.416198
```

6.1.2 Operasi matriks

Misalkan kita memiliki 2 buah matriks A dan B.

$$A = \begin{bmatrix} a_{1.1} & a_{1.2} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & \cdots & a_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & \cdots & a_{m.n} \end{bmatrix} dan\ B = \begin{bmatrix} b_{1.1} & b_{1.2} & \cdots & b_{1.n} \\ b_{2.1} & b_{2.2} & \cdots & b_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m.1} & b_{m.2} & \cdots & b_{m.n} \end{bmatrix} \tag{6.6}$$

Jika salah satu matriks tersebut dijumlahkan atau dikurangkan dengan skalar.

$$A \pm x = \begin{bmatrix} a_{1.1} & a_{1.2} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & \cdots & a_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & \cdots & a_{m.n} \end{bmatrix} \pm x = \begin{bmatrix} a_{1.1} \pm x & a_{1.2} \pm x & \cdots & a_{1.n} \pm x \\ a_{2.1} \pm x & a_{2.2} \pm x & \cdots & a_{2.n} \pm x \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} \pm x & a_{m.2} \pm x & \cdots & a_{m.n} \pm x \end{bmatrix} \tag{6.7}$$

Jika kedua matriks A dan B saling dijumlahkan atau dikurangkan. Perlu diperhatikan bahwa penjumlahan dua buah matriks hanya dapat dilakukan pada matriks dengan ukuran yang seragam.

$$\begin{aligned} A \pm B &= \begin{bmatrix} a_{1.1} & a_{1.2} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & \cdots & a_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & \cdots & a_{m.n} \end{bmatrix} \pm \begin{bmatrix} b_{1.1} & b_{1.2} & \cdots & b_{1.n} \\ b_{2.1} & b_{2.2} & \cdots & b_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m.1} & b_{m.2} & \cdots & b_{m.n} \end{bmatrix} \\ &= \begin{bmatrix} a_{1.1} \pm b_{1.1} & a_{1.2} \pm b_{1.2} & \cdots & a_{1.n} \pm b_{1.n} \\ a_{2.1} \pm b_{2.1} & a_{2.2} \pm b_{2.2} & \cdots & a_{2.n} \pm b_{2.n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m.1} \pm b_{m.1} & a_{m.2} \pm b_{m.2} & \cdots & a_{m.n} \pm b_{m.n} \end{bmatrix} \end{aligned} \tag{6.8}$$

Untuk lebih memahaminya, berikut disajikan contoh operasi penjumlahan pada matriks:

```
A <- matrix(1:9,3)
B <- matrix(10:18,3)
C <- matrix(1:6,3)

# penjumlahan dengan skalar
A+1
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    8
## [2,]    3    6    9
## [3,]    4    7   10
```

```
# penjumlahan A+B
A+B
```

```
##      [,1] [,2] [,3]
## [1,]   11   17   23
## [2,]   13   19   25
## [3,]   15   21   27
```

```
# penjumlahan
A+C
```

Operasi pehitungan lain yang penting pada matriks adalah operasi perkalian matriks. Perlu diperhatikan bahwa untuk perkalian matriks, jumlah kolom matriks sebelah kiri harus sama dengan jumlah baris pada matriks sebelah kanan. Perkalian antara dua matriks disajikan pada Persamaan (6.9).

$$A_{m.n} \times B_{n.r} = AB_{m.r} \tag{6.9}$$

Pada R perkalian matriks dilakukan menggunakan operator %*%. Berikut adalah contoh perkalian matriks pada R:

```
# Perkalian matriks
A%*%B
```

```
##      [,1] [,2] [,3]
## [1,]  138  174  210
## [2,]  171  216  261
## [3,]  204  258  312
```

6.2 Operasi Baris Elementer

Terdapat tiga buah operasi dasar pada baris matriksoperasi baris elementer. Ketiga operasi ini akan menjadi dasar operasi *sub-chapter* selanjutnya. Ketiga operasi dasar tersebut antara lain:

1. ***Row Scalling***. Mengalikan baris matriks dengan konstanta bukan nol.
2. ***Row Swaping***. Menukar urutan baris pada sebuah matriks (contoh: menukar baris 1 dengan baris 2 dan sebaliknya).
3. ***Row Replacement***. Baris matriks diganti dengan hasil penjumlahan atau pengurangan baris matriks tersebut dengan baris matriks lainnya, dimana baris matriks lainnya yang akan dijumlahkan/dikurangkan dengan matriks tersebut telah dilakukan proses *row scalling*. Luaran yang diperoleh pada umumnya adalah nilai nol pada baris matriks awal atau akhir.

Ketiga proses tersebut akan terjadi secara berulang, khusunya jika kita hendak mengerjakan sistem persamaan linier menggunakan algoritma eliminasi Gauss. Untuk mempermudah proses tersebut, kita dapat membuat masing-masing fungsi untuk masing-masing operasi tersebut. Algoritma fungsi-fungsi tersebut selanjutnya menjadi dasar penyusunan algoritma fungsi-fungsi eliminasi Gauss dan dekomposisi matriks yang akan dijelaskan pada *chapter* selanjutnya.

Fungsi *row scalling* pada `R` dapat dituliskan pada sintaks berikut:

```
scale_row <- function(m, row, k){
 m[row, ] <- m[row, ]*k
 return(m)
}
```

Berikut adalah contoh penerapannya:

```
# membuat matriks A
(A <- matrix(1:15, nrow=5))
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
# lakukan scaling pada row 2 dengan nilai 10
scale_row(m=A, row=2, 10)
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]   20   70  120
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

> **Catatan:** Untuk menyimpan hasil perhitungan, simpan proses perhitungan dalam sebuah objek (lihat Chapter 2.5).

Row swapping merupakan proses yang berulang, kita perlu menyimpan terlebih dahulu baris matriks pertama kedalam sebuah objek. Baris matriks pertama selanjutnya diganti dengan baris matriks kedua, sedangkan baris matriks kedua selanjutnya akan diganti dengan baris matriks pertama yang telah terlebih dahulu disimpan dalam sebuah objek. Fungsi *row swapping* pada `R` dapat dituliskan pada sintaks berikut:

```
swap_row <- function(m, row1, row2){
  row_tmp <- m[row1, ]
  m[row1, ] <- m[row2, ]
  m[row2, ] <- row_tmp
  return(m)
}
```

Berikut merupakan contoh penerapan fungsi `swap_row()`:

```
# pertukarkan baris 2 dengan baris 5
swap_row(m=A, row1=2, row2=5)
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    5   10   15
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    2    7   12
```

Pada proses *row replacement*, proses perhitungan dilakukan dengan melakukan penjumlahan suatu baris matriks dengan baris matriks lainnya dengan terlebih dahulu melakukan *row scalling* terhadap matriks lainnya. Berikut adalah fungsi `replace_row()` yang ditulis pada `R`:

```
replace_row <- function(m, row1, row2, k){
  m[row2, ] <- m[row2, ] + m[row1, ]*k
  return(m)
}
```

Berikut adalah contoh penerapan fungsi `replace_row()`:

```
replace_row(m=A, row1=1, row2=3, k=-3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    0  -10  -20
## [4,]    4    9   14
## [5,]    5   10   15
```

6.3 Eliminasi Gauss

Pada *sub-chapter* ini kita akan menggunakan operasi baris elementer yang telah dijelaskan pada Chapter 2.5. Terdapat dua topik yang akan dibahas pada *sub-chapter* ini, yaitu: *row echelon form* termasuk *reduced row echelon form* dan matriks tridiagonal.

Eliminasi Gauss merupakan sebuah cara untuk mencari penyelesaian sistem persamaan linier. Ide dasar dari eliminasi Gauss adalah melakukan operasi matematika pada baris matriks (lihat Chapter 2.5) dan melanjutkannya sampai hanya tersisa satu variabel saja. Kita dapat melakukan lebih dari satu operasi baris elementer pada proses elmininasi ini (contoh: mengalikan sebuah baris dengan konstanta dan menjumlahkan hasilnya pada baris lain).

6.3.1 *Row Echelon Form*

Sebuah matriks merupakan *row echelon form* jika matriks tersebut memenuhi beberapa kondisi:

1. Angka bukan nol pertama dari kiri (*leading coefficient*) selalu di sebelah kanan angka bukan nol pertama pada baris di atasnya.
2. Baris yang terdiri dari semua nol ada di bagian bawah matriks.

Misalkan terdapat persamaan linier seperti yang ditunjukkan pada Persamaan (6.10).

$$\begin{aligned} a_{1.1}x_1 + a_{1.2}x_2 + a_{1.3}x_3 + \cdots + a_{1.n}x_n &= b_1 \\ a_{2.1}x_1 + a_{2.2}x_2 + a_{2.3}x_3 + \cdots + a_{2.n}x_n &= b_2 \\ a_{3.1}x_1 + a_{3.2}x_2 + a_{3.3}x_3 + \cdots + a_{3.n}x_n &= b_3 \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{m.1}x_1 + a_{m.2}x_2 + a_{m.3}x_3 + \cdots + a_{m.n}x_n &= b_n \end{aligned} \tag{6.10}$$

dimana $a_{i.j}$ untuk $i = 1$ sampai dengan m dan $j = 1$ sampai dengan n merupakan koefisien persamaan linier. x_i untuk $i = 1$ sampai dengan n merupakan variabel bebas pada sistem persamaan linier.

Persamaan linier pada Persamaan (6.10) dapat dinyatakan ke dalam bentuk matriks pada Persamaan (6.11).

$$\begin{bmatrix} a_{1.1} & a_{1.2} & a_{1.3} & \cdots & a_{1.n} \\ a_{2.1} & a_{2.2} & a_{2.3} & \cdots & a_{2.n} \\ a_{3.1} & a_{3.2} & a_{3.3} & \cdots & a_{3.n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m.1} & a_{m.2} & a_{m.3} & \cdots & a_{m.n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.11}$$

$$AX = B \tag{6.12}$$

dimana:

- matriks A merupakan matriks koefisien / Jacobian
- vaktor X merupakan vaktor variabel
- vektor B merupakan vektor konstanta

matriks pada Persamaan (6.11) dapat diubah menjadi *augmented matrix*, yaitu: perluasan matriks A dengan menambahkan vektor B pada kolom terakhirnya.

$$\begin{bmatrix} a_{1.1} & a_{1.2} & a_{1.3} & \cdots & a_{1.n} & b_1 \\ a_{2.1} & a_{2.2} & a_{2.3} & \cdots & a_{2.n} & b_2 \\ a_{3.1} & a_{3.2} & a_{3.3} & \cdots & a_{3.n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{m.1} & a_{m.2} & a_{m.3} & \cdots & a_{m.n} & b_n \end{bmatrix} \tag{6.13}$$

$$A = [A|B] \tag{6.14}$$

Teorema 6.1 (spltheorem)**.** *Suatu sistem persamaan linier mempunyai penyelesaian tunggal bila memenuhi syarat-syarat sebagai berikut:*

- ukuran persamaan linier simultan bujursangkar (jumlah persamaan sama dengan jumlah variabel bebas).
- sistem persamaan linier *non-homogen* di mana minimal ada satu nilai vektor konstanta B tidak nol atau terdapat $b_n \neq 0$.
- Determinan dari matriks koefisiensistem persamaan linier tidak sama dengan nol.

Untuk memperoleh penyelesaian sistem persamaan linier, Persamaan (6.13) perlu dilakukan operasi baris elementer. Hasil operasi baris dasar akan menghasilkan matriks *row echelon form* yang disajikan pada Persamaan (6.15).

$$\begin{bmatrix} a_{1.1} & a_{1.2} & a_{1.3} & \cdots & a_{1.n} & b_1 \\ a_{2.1} & a_{2.2} & a_{2.3} & \cdots & a_{2.n} & b_2 \\ a_{3.1} & a_{3.2} & a_{3.3} & \cdots & a_{3.n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{m.1} & a_{m.2} & a_{m.3} & \cdots & a_{m.n} & b_n \end{bmatrix} \Longrightarrow \begin{bmatrix} c_{1.1} & c_{1.2} & c_{1.3} & \cdots & c_{1.n} & d_1 \\ 0 & c_{2.2} & c_{2.3} & \cdots & c_{2.n} & d_2 \\ 0 & 0 & c_{3.3} & \cdots & c_{3.n} & d_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & c_{m.n} & d_n \end{bmatrix} \tag{6.15}$$

Sehingga penyelesaian sistem persamaan linier dapat diperoleh menggunakan Persamaan (6.16).

$$\begin{aligned} x_n &= \tfrac{d_n}{c_{m.n}} \\ x_{n-1} &= \tfrac{1}{C_{m-1.n-1}} \left(d_{n-1} - c_{m-1.n} x_n\right) \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ x_2 &= \tfrac{1}{c_{2.2}} \left(d_2 - c_{2.3} x_3 - c_{2.4} x_4 - \cdots - c_{2.n} x_n\right) \\ x_1 &= \tfrac{1}{c_{1.1}} \left(d_1 - c_{1.2} x_2 - c_{1.3} x_3 - \cdots - c_{1.n} x_n\right) \end{aligned} \tag{6.16}$$

Contoh 6.1. Selesaikan sistem persamaan berikut:

$$\begin{aligned} x_1 + x_2 + x_3 &= 6 \\ x_1 + 2x_2 - x_3 &= 2 \\ 2x_1 + x_2 + 2x_3 &= 10 \end{aligned}$$

Jawab:

Augmented matrix sistem persamaan linier tersebut adalah sebagai berikut:

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

Operasi baris elementer selanjutnya dilakukan pada matriks tersebut. Pada langkah pertama, baris ke-2 dikurangkan dengan baris ke-1 ($B_2 - B_1$) dan baris ke-3 dikurangkan oleh dua kali baris ke-1 ($B_3 - 2B_1$).

$$\begin{bmatrix} 1 & 1 & 1 & 6 \\ 1 & 2 & -1 & 2 \\ 2 & 1 & 2 & 10 \end{bmatrix} \begin{matrix} B_2 - B_1 \\ \Longrightarrow \\ B_3 - 2B_1 \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 6 \\ 0 & 1 & -2 & -4 \\ 0 & -1 & 0 & -2 \end{bmatrix}$$

Hasil dari langkah pertama tersebut, selanjutnya menjadi input dari langkah selanjutnya. Pada langkah selanjutnya operasi baris elementer kembali dilanjutkan. Baris ke-3 dikurangkan denganbaris ke-2 ($B_3 - B_2$).

$$\begin{bmatrix} 1 & 1 & 1 & 6 \\ 0 & 1 & -2 & -4 \\ 0 & -1 & 0 & -2 \end{bmatrix} \begin{matrix} B_3 - B_2 \\ \Longrightarrow \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 6 \\ 0 & 1 & -2 & -4 \\ 0 & 0 & -2 & -6 \end{bmatrix}$$

Setelah diperoleh matriks *row echelon form* selanjutnya penyelesaian persamaan dapat dikerjakan menggunakan Persamaan (6.16).

$$x_3 = \frac{-6}{-2} = 3$$
$$x_2 = \frac{1}{1}(-4 - (2)\,3) = 2$$
$$x_1 = \frac{1}{1}(6 - 2 - 3) = 1$$

Algoritma *Row Echelon Form*

1. Masukkan matriks A, dan vektor B beserta ukurannya n
2. Buat *augmented matrix* $[A|B]$ namakan dengan A
3. Untuk baris ke-i dimana $i = 1$ s/d n, perhatikan apakah nilai $a_{i,j}$ sama dengan nol.

- **Bila iya**, lakukan *row swapping* antara baris ke-i dan baris ke-$i + k \leq n$, dimana $a_{i+k,j}$ tidak sama dengan nol. Bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian.
- **Bila tidak**, lanjutkan.

4. Untuk baris ke-j, dimana $j = i+1$ s/d n, lakukan operasi baris elementer:

- Hitung $c = \frac{a_{j,i}}{a_{i,i}}$
- untuk kolom k, dimana $k = 1$ s/d $n + 1$, hitung $a_{j,k} = a_{j,k} - c.a_{i,k}$

5. Hitung akar, untuk $i = n$ s/d 1 (bergerak dari baris pertama) menggunakan Persamaan (6.16).

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi pada `R` untuk menyelesaikan sistem persamaan linier menggunakan matriks *row echelon form*. Fungsi yang akan dibentuk hanya sampai pada algoritma ke-4. Proses substitusi akan dilakukan secara manual. Berikut adalah sintaks yang digunakan:

```
ref_matrix <- function(a){
  m <- nrow(a)
  n <- ncol(a)
  piv <- 1

# cek elemen diagonal apakah bernilai nol
  for(row_curr in 1:m){
    if(piv <= n){
      i <- row_curr
      while(a[i, piv] == 0 && i < m){
        i <- i+1
        if(i > m){
          i <- row_curr
          piv <- piv+1
```

```
          if(piv > n)
            return(a)
        }
      }

# jika diagonal bernilai nol, lakukan row swapping
    if(i != row_curr)
      a <- swap_row(a, i, row_curr)

# proses triangulasi untuk membentuk matriks segitiga atas
    for(j in row_curr:m)
      if(j != row_curr){
        c <- a[j, piv]/a[row_curr, piv]
        a <- replace_row(a, row_curr, j, -c)
      }
    piv <- piv+1
    }
  }
  return(a)
}
```

Dengan menggunakan fungsi `ref_matrix()`, kita dapat membentuk matriks *row echelon form* pada Contoh 6.1.

```
am <- c(1,1,2,
        1,2,1,
        1,-1,2,
        6,2,10)
(m <- matrix(am, nrow=3))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    6
## [2,]    1    2   -1    2
## [3,]    2    1    2   10
```

```
ref_matrix(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    6
## [2,]    0    1   -2   -4
## [3,]    0    0   -2   -6
```

matriks yang diperoleh selanjutnya dapat diselesaikan menggunakan Persamaan (6.16).

Contoh 6.2. Dengan menggunakan fungsi `ref_matrix()`, buatlah matriks *row echelon form* dari sistem persamaan linier berikut:

$$\begin{aligned} 2x_1 + x_2 - x_3 &= 1 \\ 3x_1 + 2x_2 - 2x_3 &= 1 \\ x_1 - 5x_2 + 4x_3 &= 3 \end{aligned}$$

Jawab:

Augmented matrix dari sistem persamaan tersebut adalah sebagai berikut:

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

Penyelesaian matriks tersebut adalah sebagai berikut:

```
(m <- matrix(c(2,3,1,
               1,2,-5,
               -1,-2,4,
               1,1,3), nrow=3))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    1   -1    1
## [2,]    3    2   -2    1
## [3,]    1   -5    4    3
```

```
ref_matrix(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2  1.0 -1.0  1.0
## [2,]    0  0.5 -0.5 -0.5
## [3,]    0  0.0 -1.0 -3.0
```

Proses lebih lanjut akan menghasilkan penyelesaian sebagai berikut:

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

6.3.2 Eliminasi Gauss-Jordan

Berbeda dengan metode eliminasi Gauss yang telah dijelaskan pada Chapter 6.3.1, metode eliminasi Gauss-Jordan membentuk matriks menjadi bentuk *reduced row echelon form*. Metode ini merupakan pengembangan metode eliminasi Gauss, dimana matriks sebelah kiri *augmented matrix* diubah menjadi matriks diagonal (lihat Persamaan (6.17)).

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,n} & b_n \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & d_1 \\ 0 & 1 & 0 & \cdots & 0 & d_2 \\ 0 & 0 & 1 & \cdots & 0 & d_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & \cdots & 1 & d_n \end{bmatrix} \tag{6.17}$$

Sehingga penyelesaian persamaan linier tersebut adalah nilai $d_1, d_2, d3, \ldots, d_n$ dan atau:

$$\begin{gathered} x_1 = d_1 \\ x_2 = d_2 \\ x_3 = d_3 \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ x_n = d_n \end{gathered} \tag{6.18}$$

Contoh 6.3. Selesaikan sistem persamaan berikut:

$$\begin{aligned} x_1 + x_2 &= 3 \\ 2x_1 + 4x_2 &= 8 \end{aligned}$$

Jawab:

Augmented matrix dari persamaan linier tersebut adalah sebagai berikut:

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 4 & 8 \end{bmatrix}$$

Operasi baris elementer selanjutnya dilakukan pada matriks tersebut.

$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 4 & 8 \end{bmatrix} \overset{B_2 - 2B_1}{\Longrightarrow} \begin{bmatrix} 1 & 1 & 3 \\ 0 & 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 0 & 2 & 2 \end{bmatrix} \overset{\frac{B_2}{2}}{\Longrightarrow} \begin{bmatrix} 1 & 1 & 3 \\ 0 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 3 \\ 0 & 1 & 1 \end{bmatrix} B_1 - B_2 \Longrightarrow \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

Penyelesaian persamaan linier tersebut adalah sebagai berikut:

$$x_1 = 2 \; dan \; x_2 = 1$$

Algoritma Metode Eliminasi Gauss-Jordan

1. Masukkan matriks A dan vektor B beserta ukurannya n
2. Buat *augmented matrix* $[A|B]$ namakan dengan A
3. Untuk baris ke-i dimana $i = 1$ s/d n

- Perhatikan apakah nilai $a_{i.i}$ sama dengan nol:
  - **Bila ya**: pertukakan baris ke-i dan baris ke-$i+k \leq n$, dimana $a_{i+k.i}$ tidak sama dengan nol, bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian.
  - **Bila tidak**: lanjutkan
- Jadikan nilai diagonalnya menjadi satu dengan cara untuk setiap kolom k dimana $k = 1$ s/d $n+1$, hitung $a_{i.k} = \frac{a_{i.k}}{a_{i.i}}$

4. Untuk baris ke-j, dimana $j = i+1$ s/d n. Lakukan operasi baris elementer untuk kolom k dimana $k = 1$ s/d n.

- Hitung $c = a_{j.i}$
- Hitung $a_{j.k} = a_{j.k} - c.a_{i.k}$

5. Penyelesaian untuk $i = n$ s/d 1 disajikan pada Persamaan (6.18).

Dari algoritma tersebut, kita dapat membangun sebuah fungsi menggunakan `R`. Fungsi tersebut adalah sebagai berikut:

```
gauss_jordan <- function (a){
    m <- nrow (a)
    n <- ncol (a)
    piv <- 1

# cek elemen diagonal utama apakah bernilai nol
    for(row_curr in 1:m){
        if(piv <= n){
```

```
            i <- row_curr
            while(a[i, piv] == 0 && i < m){
                i <- i + 1
                if(i > m){
                    i <- row_curr
                    piv <- piv + 1
                    if(piv > n)
                        return (a)
                }
            }

# jika diagonal utama bernilai nol,lakukan row swapping
            if(i != row_curr)
                a <- swap_row(a, i, row_curr)

# proses pembentukan matriks reduced row echelon form
            piv_val <- a[row_curr , piv]
            a <- scale_row (a, row_curr , 1 / piv_val)
            for(j in 1: m){
                if(j != row_curr){
                    k <- a[j, piv]/a[row_curr, piv]
                    a <- replace_row (a, row_curr, j, -k)
                }
            }
            piv <- piv + 1
        }
    }
    return (a)
}
```

Dengan menggunakan fungsi `gauss_jordan()`, sistem persamaan linier pada Contoh 6.3:

```
(m <- matrix(c(1,2,1,4,3,8), nrow=2))
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    3
## [2,]    2    4    8
```

```
gauss_jordan(m)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    2
## [2,]    0    1    1
```

Contoh 6.4. Dengan menggunakan fungsi `gauss_jordan()`, carilah penyelesaian sistem persamaan linier pada Contoh 6.1 dan Contoh 6.2:

Jawab:

Untuk Contoh 6.1:

```
am <- c(1,1,2,
        1,2,1,
        1,-1,2,
        6,2,10)
m <- matrix(am, nrow=3)

gauss_jordan(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    1
## [2,]    0    1    0    2
## [3,]    0    0    1    3
```

Untuk Contoh 6.2:

```
m <- matrix(c(2,3,1,1,2,-5,
              -1,-2,4,1,1,3),
            nrow=3)
gauss_jordan(m)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    1
## [2,]    0    1    0    2
## [3,]    0    0    1    3
```

6.3.3 Matrik Tridiagonal

Metode eliminasi Gauss merupakan metode yang sederhana untuk digunakan khususnya jika semua koefisien bukan nol berkumpul pada diagonal utama dan beberapa diagonal sekitarnya. Suatu sistem yang bersifat demikian disebut sebagai *banded* dan banyaknya diagonal yang memuat koefisien bukan nol disebut sebagai *bandwidth*. Contoh khusus yang sering dijumpai adalah matriks tridiagonal yang memiliki *bandwidth* tiga.

Proses eliminasi untuk matriks tridiagonal bersifat trivial karena dengan membentuk sebuah subdiagonal tambahan, proses substitusi mundur segera dapat dilakukan. Bentuk matriks tridiagonal disajikan pada Persamaan (6.19).

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & 0 \\ 0 & a_{3,2} & a_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.19}$$

Penyelesaian persamaan tersebut disajikan pada Persamaan (6.20).

$$x_n = \frac{b_n}{a_{m.n}};\ x_i = \frac{b_i - a_{i,j+1}x_{i+1}}{a_{i,j}} \tag{6.20}$$

dimana $i = n-1, n-2, \ldots, 1$.

Pada beberapa *textbook*, diagonal matriks sering dilambangkan dengan l(diagonal bawah), d(diagonal tengah), dan u (diagonal atas). Bentuk matriksnya disajikan pada Persamaan (6.21).

$$\begin{bmatrix} d_1 & u_2 & 0 & \cdots & 0 \\ l_2 & d_2 & u_3 & \cdots & 0 \\ 0 & l_3 & d_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.21}$$

Algoritma Penyelesaian Matrik Tridiagonal

1. Bentuk sistem persamaan linier menjadi matriks pada Persamaan (6.21).
2. Lakukan *foward sweep*. Setiap elemen diagonal l dieliminasi menggunakan reduksi baris.

- Untuk $i = 1$
  - Hitung $u_1 = \frac{u_1}{d_1}$
  - Hitung $b_1 = \frac{b_1}{d_1}$
- Untuk $i = 2$ s/d $n-1$
  - Hitung $u_i = \frac{u_i}{d_i - l_i \times u_{i-1}}$
  - Hitung $b_i = \frac{b_i - l_i \times u_{i-1}}{d_i - l_i \times u_{i-1}}$
- Hitung $b_n = \frac{b_n - l_n \times u_{n-1}}{d_n - l_n \times u_{n-1}}$

3. Lakukan *backward sweep*. Setiap elemen diagonal u dilakukan eliminasi.

- Untuk $i = n - 1$ s/d 1
  - Hitung $x_n = b_i - u_i \times x_{i+1}$
- Hitung $x_n = b_n$

Berdasarkan algoritma tersebut, kita dapat membangun sebuah fungsi pada `R`. Fungsi penyelesaian matriks tridiagonal disajikan sebagai berikut:

```
tridiagmatrix <- function (L, D, U, b){
  n <- length (D)
  L <- c(NA , L)

  ## forward sweep
  U[1] <- U[1] / D[1]
  b[1] <- b[1] / D[1]
  for(i in 2:(n - 1)){
      U[i] <- U[i] / (D[i] - L[i] * U[i - 1])
      b[i] <- (b[i] - L[i] * b[i - 1]) /
      (D[i] - L[i] * U[i - 1])
  }
  b[n] <- (b[n] - L[n] * b[n - 1])/(D[n] - L[n] * U[n - 1])

  ## backward sweep
  x <- rep.int (0, n)
  x[n] <- b[n]
  for(i in (n - 1) :1)
      x[i] <- b[i] - U[i] * x[i + 1]
  return (x)
}
```

Contoh 6.5. Selesaikan sistem persamaan berikut menggunakan fungsi `tridiagmatrix()` dan fungsi `gauss_jordan()`!

$$\begin{aligned} 3x_1 + 4x_2 &= 20 \\ 4x_1 + 5x_2 - 2x_3 &= 28 \\ 2x_2 + 5x_3 - 3x_4 &= 18 \\ 3x_3 + 5x_4 &= 18 \end{aligned}$$

Jawab:

Langkah pertama untuk menyelesaikannya, kita harus merubah persamaan tersebut kedalam bentuk matriks

$$\begin{bmatrix} 3 & 4 & 0 & 0 \\ 4 & 5 & 2 & 0 \\ 0 & 2 & 5 & 3 \\ 0 & 0 & 3 & 5 \end{bmatrix} x = \begin{bmatrix} 20 \\ 28 \\ 18 \\ 18 \end{bmatrix}$$

Untuk menyelesaikan persamaan tersebut menggunakan fungsi `tridiagmatrix()`, kita perlu membentuk vektor diagonal l, d, u, dan b.

```
l <- u <- c(4, 2, 3); d <- c(3, 5, 5, 5)
b <- c(20, 28, 18, 18)
```

Setelah terbentuk, vektor tersebut dapat langsung dimasukkan ke dalam fungsi `tridiagmatrix()`.

```
tridiagmatrix(L=l, D=d, U=u, b=b)
```

```
## [1] 4 2 1 3
```

Untuk menyelesaikannya menggunakan fungsi `gauss_jordan()`, kita perlu membentuk *augmented matrix*-nya terlebih dahulu.

```
m <- matrix(c(3,4,0,0,4,5,2,0,
              0,2,5,3,0,0,3,5,
              20,28,18,18), nrow=4)
gauss_jordan(m)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    4
## [2,]    0    1    0    0    2
## [3,]    0    0    1    0    1
## [4,]    0    0    0    1    3
```

6.3.4 Penyelesaian Sistem Persamaan Linier Menggunakan Fungsi `solve()`

`R` menyediakan fungsi bawaan `solve()` untuk menyelesaiakan sistem persamaan linier. Format fungsi `solve()` adalah sebagai berikut:

```
solve(a,b)
```

Catatan:

- **a**: matriks koefisien atau matriks segiempat
- **b**: vektor konstanta

Berikut adalah contoh penerapan fungsi `solve()` pada sistem persamaan linier yang disajikan pada Contoh 6.2:

```
# memecah matriks m menjadi matriks koefisien dan vektor konstanta
a <- matrix(c(2,3,1,1,2,-5,-1,-2,4),nrow=3)
b <- c(1,1,3)

solve(a,b)
```

```
## [1] 1 2 3
```

Jika kita hanya memasukkan matriks persegi, maka output yang akan dihasilkan adalah invers dari matriks yang kita masukkan.

```
solve(a)
```

```
##      [,1] [,2]          [,3]
## [1,]    2   -1  7.401487e-17
## [2,]   14   -9 -1.000000e+00
## [3,]   17  -11 -1.000000e+00
```

Jika kita mengalikan invers dengan matriks semula, maka akan dihasilkan output berupa matriks identitas.

```
a%*%solve(a)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

6.3.5 Penyelesaian Sistem Persamaan Linier Menggunakan Fungsi 'Solve.tridiag()'

Penyelesaian matriks tridiagonal selain menggunakan fungsi `solve()`, juga dapat menggunakan fungsi `Solve.tridiag()` dari paket `limSolve`. Untuk menginstall dan mengaktifkan paket tersebut, jalankan sintaks berikut:

```
install.packages("limSolve")
```

```
library(limSolve)
```

Fungsi `Solve.tridiag()` memiliki format sebagai berikut:

```
Solve.tridiag ( diam1, dia, diap1, B=rep(0,times=length(dia)))
```

Catatan:

- **diam1**: vektor bukan nol di bawah diagonal matriks
- **dia**: vektor bukan nol pada diagonal matriks
- **diap1**: vektor bukan nol di atas diagonal matriks
- **B**: vektor konstanta

Untuk memahami penerapannya, kita akan menggunakan kembali matriks yang ada pada Contoh 6.5.

```
l <- u <- c(4, 2, 3); d <- c(3, 5, 5, 5)
b <- c(20, 28, 18, 18)
Solve.tridiag(diam1=l, dia=d, diap1=u, B=b)
```

```
##      [,1]
## [1,]    4
## [2,]    2
## [3,]    1
## [4,]    3
```

6.4 Dekomposisi Matriks

Seringkali kita diminta untuk memperoleh nilai penyelesaian suatu persamaan linier $Ax = B$, dimana nilai vektor B yang selalu berubah-ubah. Penggunaan metode eliminasi Gauss mengharuskan untuk menyelesaikan sistem persamaan linier $Ax = B$ secara terpisah untuk setiap perubahan vektor B. Untuk menghindari pekerjaan eliminasi yang selalu berulang-ulang, faktorisasi menjadi suatu hal yang dapat dilakukan untuk mempersingkat prosesnya. Faktorisasi atau dekomposisi matriks merupakan suatu algoritma untuk memecah matriks A, hasil pemecahan ini selanjutnya digunakan untuk memperoleh penyelesaian sistem persamaan linier melalui perkalian antara vektor B dan hasil faktorisasi matriks A.

6.4.1 Dekomposisi LU

Misalkan kita memiliki persamaan linier seperti yang ditunjukkan oleh Persamaan (6.12). Pada metode dekomposisi LU, matriks A difaktorkan menjadi matriks L dan matriks U, dimana ukuran kedua matriks tersebut harus sama dengan ukuran matriks A atau dapat kita tuliskan bahwa hasil perkalian kedua matriks tersebut akan menghasilkan matriks A.

$$A = LU \tag{6.22}$$

Sehingga Persamaan (6.12) akan menjadi Persamaan (6.23).

$$LUx = b \tag{6.23}$$

Langkah penyelesaian sistem persamaan linier, diawali dengan menghadirkan vektor t yang ditunjukkan pada Persamaan (6.24).

$$Ux = t \tag{6.24}$$

Langkah pada Persamaan (6.24) tidak dimaksudkan untuk menghitung vektor t, melainkan untuk menghitung vektor x. Vektor t diperoleh dengan menggunakan Persamaan (6.25).

$$Lx = t \tag{6.25}$$

Kita dapat menyelesaikan sistem persamaan yang ditunjukkan pada Persamaan (6.24) dan Persamaan (6.25) menggunakan berbagai algoritma penyelesaian yang telah dibahas sebelumnya. Namun, karena matriks L merupakan matriks segitiga bawah dengan nilai nol berada pada bagian atas diagonal utama, penyelesaian t mengambil langkah yang lebih sedikit. Kondisi ini sama dengan kondisi penyelesaian matriks tridiagonal, dimana kita memanfaatkan sejumlah jalan pintas penyelesaiaannya guna mempercepat komputasi. Matriks segitia bawah L akan berupa matriks persegi dengan ukuran m, di mana m merupakan jumlah baris matriks A. Persamaan (6.25) dalam bentuk matriks akan terlihat seperti Persamaan (6.26).

$$Lt = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{2,1} & 1 & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{m,1} & l_{m,2} & l_{m,3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \cdots \\ t_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_n \end{bmatrix} \tag{6.26}$$

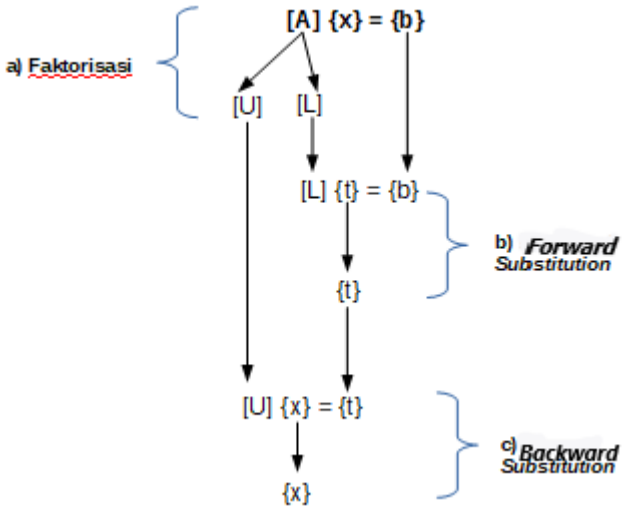

Figure 6.1: Tahapan dekomposisi LU.

Berdasarkan Persamaan (6.26), diketahui nilai $t_1 = b_1$. Nilai ini selanjutnya dapat digunakan untuk melakukan proses substitusi guna memperoleh seluruh nilai vektor t. Proses ini disebut sebagai *foward substitution*. Proses substitusi dapat dituliskan menggunakan Persamaan (6.27).

$$t_i = b_i - \sum_{j=1}^{i-1} l_{i,j} t_i \tag{6.27}$$

Seteleh nilai vektor t dihitung, kita dapat menghitung nilai x pada Persamaan (6.28).

$$Ux = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \cdots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \cdots & u_{2,n} \\ 0 & 0 & u_{3,3} & \cdots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \cdots \\ t_n \end{bmatrix} \tag{6.28}$$

Jika diperhatikan, kita dapat mengetahui mengetahui nilai $x_n = \frac{t_n}{u_{m,n}}$. Nilai tersebut selanjutnya dapat digunakan untuk melakukan proses susbtitusi pada nilai lainnya. Proses substitusi ini disebut sebagai *backward substitution*. Proses dekomposisi atau faktorisasi LU digambarkan pada Gambar 6.1.

Dekomposisi LU didasarkan pada operasi baris elementer. Pertama, kita perlu menemukan matriks segitiga atas yang sesuai dengan matriks A. Solusi untuk melakukan dekomposisi bisa jadi tak terhingga, namun solusi yang paling sederhana adalah mengubah matriks A menjadi matriks *row echelon form*. Kedua, L harus menjadi matriks segitiga bawah yang mereduksi ke-l dengan mengikuti operasi baris yang sama yag menghasilkan U. Kita dapat menggunakan algoritma Doolittle untuk menghasilkan L, di mana nilai setiap entri dalam matriks segitiga bawah merupakan pengali yang digunakan untuk menghilangkan entri yang sesuai untuk setiap proses *row replacement*.

Pada praktiknya, proses eliminasi Gauss untuk memperoleh matriks U kadang menghasilkan nol di kolom pivotnya. Kondisi tersebut mengharuskan kita untuk melakukan proses *row swapping* atau pertukaran baris (biasanya dengan baris bawahnya) untuk pivot bukan nol. Jika proses tersebut berhasil dilakukan bisa jadi matriks A mungkin setara dengan matriks LU, tetapi tidak sama dalam hal urutan nilai pada tiap barisnya. Agar kita dapat memperoleh hasil yang sama (matriks A sama dengan matriks LU), diperlukan matriks ketiga, P. Matriks ini merupakan matriks identitas dengan ukuran sama dengan matriks A. Jika pertukaran baris dilakukan selama proses pembentukan matriks U, maka pertukaran baris yang sama juga akan diimplemenntasikan pada matriks P. oleh karena itu, dalam praktiknya matriks $A = PLU$ dan perkalian dengan matriks P berfungsi untuk mengembalikan urutan baris.

Contoh 6.6. Selesaikan sistem persamaan linier berikut menggunakan faktorisasi LU

$$\begin{aligned} x_1 + x_2 + 3x_4 &= 4 \\ 2x_1 + x_2 - x_3 + x_4 &= 1 \\ 3x_1 - x_2 - x_3 + 2x_4 &= -3 \\ -x_1 - 2x_2 + 3x_3 - x_4 &= 4 \end{aligned}$$

Jawab:

Nayatakan sistem persamaan tersebut ke dalam bentuk matriks $Ax = b$.

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

Lakukan operasi baris elementer pada matriks A untuk memperoleh matriks U. Urutan operasi baris elementer yang dilakukan adalah sebagai berikut:

- $(B_2 - 2B_1) \rightarrow B_2 \rightarrow l_{2,1} = 2$,
- $(B_3 - 3B_1) \rightarrow B_3 \rightarrow l_{3,1} = 3$,

- $(B_4 + B_1) \rightarrow B_4 \rightarrow l_{4,1} = -1$,
- $(B_3 - 4B_2) \rightarrow B_3 \rightarrow l_{3,2} = 4$,
- $(B_4 + 3B_2) \rightarrow B_4 \rightarrow l_{4,2} = -3$,
- $l_{4,3} = 0$

Simpan pengali tiap tahapan pada masing-masing elemen matriks L. Hasil operasi tersebut akan menghasilkan matriks triangular U.

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

Untuk matriks L sebagai berikut:

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

Karena pada proses operasi baris elementer tidak terdapat operasi pertukaran baris, maka matriks P tidak mengalami perubahan:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Lakukan operasi *forward substitution* menggunakan Persamaan (6.26).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ -3 \\ 4 \end{bmatrix}$$

Berdasarkan hasil perhitungan diperoleh nilai vektor t.

$$t_1 = 4, t_2 = 1, t_3 = -3, t_4 = 4$$

Operasi terakhir yang perlu dilakukan untuk memperoleh nilai x adalah dengan melakukan *backward substitution* menggunakan nilai vektor t yang telah dihitung.

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

Berdasarkan hasil perhitungan diperoleh nilai x sebagai berikut:

$$x_1 = -1, x_2 = 2, x_3 = 0, x_4 = 1$$

Algoritma Dekomposisi LU

1. Masukkan matriks A, dan vektor B beserta ukurannya n
2. Lakukan langkah poin ke-4 s/d poin 5 untuk meperoleh matriks U.
3. Untuk baris ke-i di mana $i = 1$ s/d n, perhatikan apakah nilai $a_{i,j}$ sama dengan nol.

- **Bila iya**, lakukan *row swapping* antara baris ke-i dan baris ke-$i + k \leq n$, dimana $a_{i+k,j}$ tidak sama dengan nol. Bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian.
- **Bila tidak**, lanjutkan.

5. Untuk baris ke-j, dimana $j = i+1$ s/d n, lakukan operasi baris elementer:

- Hitung $c = \frac{a_{j,i}}{a_{i,i}}$
- untuk kolom k, dimana $k = 1$ s/d $n + 1$, hitung $a_{j,k} = a_{j,k} - c_i.a_{i,k}$

6. Lakukan langkah poin ke-7 s/d poin 9 untuk memperoleh matriks L
7. Untuk diagonal matriks L isikan dengan nilai 1 dan elemen di atas diagonal dengan nilai nol.
8. Untuk elemen di bawah diagonal isikan dengan faktor pengali operasi baris elementer matriks U.
9. Lakukan proses *forward substitution* menggunakan Persamaan (6.27) untuk memperoleh nilai vektor t.
10. Lakukan *backward substituion* menggunakan Persamaan (6.16).

Berdasarkan algoritma tersebut, kita dapat menyusun algoritma faktorisasi LU menggunakan `R`. Berikut adalah sintaks yang digunakan:

```
lu_solve <- function(a, b=NULL){
    m <- nrow(a)
    n <- ncol(a)
    piv <- 1
```

```
# membentuk matriks identitas P dan L
    P <- L <- diag(n)

# cek elemen diagonal utama apakah bernilai nol
    for(row_curr in 1:m){
        if(piv <= n){
            i <- row_curr
            while(a[i, piv] == 0 && i < m){
                i <- i + 1
                if(i > m){
                    i <- row_curr
                    piv <- piv + 1
                    if(piv > n)
                        return(list(P = P, L = L, U = a))
                }
            }

# jika elemen diagonal utama bernilai nol,lakukan row swapping
            if(i != row_curr){
                a <- swap_row(a, i, row_curr)
                P <- swap_row(P, i, row_curr)
            }

  # pembentukan matriks L dan U
            for(j in row_curr:m)
                if(j != row_curr){
                    k <- a[j, piv]/a[row_curr, piv]

  # matriks U
                    a <- replace_row(a, row_curr, j, -k)

  # pengisian elemen matriks L
                    L[j, piv] <- k
                }
            piv <- piv + 1
        }
    }

# penyelesaian persamaan linier
    if(is.null(b)){
      return(list(P = P, L = L, U = a))
    }else{

      # forward substitution
      t <- forwardsolve(L, b)
```

```
      # backward substitution
      x <- backsolve(a, t)
      return(list(P = P, L = L, U = a, result=x))
     }
}
```

Kita dapat menyelesaikan sistem persamaan linier pada Contoh 6.6 menggunakan fungsi yang telah kita buat.

```
# membuat matriks a dan vektor b
a <- matrix(c(1,2,3,-1,1,1,-1,2,
              0,-1,-1,3,3,1,2,-1),
            nrow=4)
b <- c(4,1,-3,4)

# penyelesaian
decomp<-lu_solve(a,b)
```

Untuk membentuk kembali matriks A, kita dapat mengalikan matriks L, U, dan P.

```
decomp$L%*%decomp$U%*%decomp$P
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    0    3
## [2,]    2    1   -1    1
## [3,]    3   -1   -1    2
## [4,]   -1    2    3   -1
```

Contoh 6.7. Lakukan dekomposisi LU pada matriks berikut dan lakukan pengecekan apakah perkalian hasil dekomposisi matriks akan menghasilkan matriks semula!

$$\begin{bmatrix} 0 & 1 & -1 \\ 1 & 5 & 9 \\ 7 & -1 & -5 \end{bmatrix}$$

Jawab:

Lakukan proses dekomposisi menggunakan fungsi `lu_solve()`.

```
# membentuk matriks a
(A <- matrix(c(0, 1, 7, 1, 5, -1, -2, 9, -5), 3))
```

```
##      [,1] [,2] [,3]
## [1,]    0    1   -2
## [2,]    1    5    9
## [3,]    7   -1   -5
```

```
# dekomposisi lu
decomp<-lu_solve(A)
```

Lakukan pengecekan apakah matriks hasil dekomposisi akan menghasilkan matriks A.

```
decomp$P %*% decomp$L %*% decomp$U
```

```
##      [,1] [,2] [,3]
## [1,]    0    1   -2
## [2,]    1    5    9
## [3,]    7   -1   -5
```

Fungsi `lu()` pada paket `Matrix` dapat digunakan untuk melakukan dekomposisi LU. Untuk meggunakan fungsi tersebut, kita harus menginstall dan mengaktifkan paket `Matrix`.

```
install.packages("Matrix")
```

```
library(Matrix)
```

Untuk dapat menggunakannya kita hanya perlu menginputkan matriks kedalam fungsi tersebut. Berikut adalah contoh penerapannya:

```
# membuat matriks a
a <- Matrix(round(rnorm(9),2), nrow=3)

# dekomposisi
lum <- lu(a)
lum
```

```
## 'MatrixFactorization' of Formal class 'denseLU' [package "Matrix"] with 4 slots
##   ..@ x       : num [1:9] -0.95 0.705 0.589 0.53 -0.604 ...
##   ..@ perm    : int [1:3] 2 3 3
```

```
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ Dim     : int [1:2] 3 3
```

Untuk menampilkan hasil dekomposisi, jalankan fungsi `expand()`.

```
decomp <- expand(lum)
decomp
```

```
## $L
## 3 x 3 Matrix of class "dtrMatrix" (unitriangular)
##      [,1]       [,2]       [,3]
## [1,]  1.0000000          .          .
## [2,]  0.7052632  1.0000000          .
## [3,]  0.5894737 -0.2278591  1.0000000
##
## $U
## 3 x 3 Matrix of class "dtrMatrix"
##      [,1]       [,2]       [,3]
## [1,] -0.9500000  0.5300000  1.7600000
## [2,]          . -0.6037895 -0.7512632
## [3,]          .          .  0.1913441
##
## $P
## 3 x 3 sparse Matrix of class "pMatrix"
##
## [1,] . . |
## [2,] | . .
## [3,] . | .
```

6.4.2 Dekomposisi Cholesky

Dekomposisi Cholesky memberikan faktorisasi matriks alternatif sehingga $A = LL^T$, di mana L^T merupakan transpose konjugat dari matriks L. Dalam kasus ini, penulis hanya bekerja dengan matriks rill dengan nilai rill dan bagian imajiner nol. Jadi untuk tujuan *sub-chapter* ini, matriks L^T hanyalah transpose dari matriks L.

Seperti dekomposisi LU, dekomposisi Cholesky dapat digunakan untuk menyelesaikan sistem persamaan linier. Kelebihannya, Menemukan dekomposisi Cholesky jauh lebih cepat daripada dekomposisi LU. Namun, dekomposisi ini hanya terbatas pada matriks tertentu saja. Dekomposisi Cholesky hanya dapat digunakan pada matriks definit positif dan simetris. Matriks simeteris merupakan matriks yang nilai di atas dan di bawah diagonalnya simetris atau

sama; secara matematis, untuk semua i dan j pada matriks A, $a_{i;j} = a_{j;i}$. Definit positif berarti bahwa setiap entri pivot (nilai elemen diagonal utama) selelu bernilai positif. Selain itu, untuk matriks definit positif, hubungan $xAx > 0$ untuk semua vektor, x.

Karena L transpose dari matriks L, maka $l^T_{i,j} = l_{j,i}$ untuk semua nilai i dan j. Tanpa kendala (*constraint*) ini, dekomposisi Cholesky akan mirip dekomposisi LU. Tetapi dengan kendala ini, nilai elemen matriks L dan L^T harus dipilih dengan cermat sehingga hubungan $A = LL^T$ berlaku. Bentuk dekomposisi Cholesky disajikan pada Persamaan (6.29).

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & a_{m,3} & \cdots & a_{m,m} \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{m,1} & l_{m,2} & l_{m,3} & \cdots & l_{m,m} \end{bmatrix} \begin{bmatrix} l_{1,1} & l_{1,2} & l_{1,3} & \cdots & l_{1,m} \\ 0 & l_{2,2} & l_{2,3} & \cdots & l_{2,m} \\ 0 & 0 & l_{3,3} & \cdots & l_{3,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & l_{m,m} \end{bmatrix} \tag{6.29}$$

Untuk setiap elemen matriks A memiliki hubungan yang dituliskan pada Persamaan (6.30).

$$a_{i,j} = \sum_{k=1}^{m} L_{i,k} L_{k,j} \tag{6.30}$$

Berdasarkan Persamaan (6.29), sejumlah nilai elemen $L_{i,k}$ dan $L_{k,j}$ adalah nol. Nilai tiap elemen diagonal utama yang tidak bernilai nol dihitung menggunakan Persamaan (6.31).

$$l_{i,i} = \sqrt{\left(a_{i,i} - \sum_{k=1}^{i-1} l^2_{i,k} \right)} \tag{6.31}$$

Elemen diagonal dihitung menggunakan Persamaan (6.32)

$$l_{i,j} = \frac{1}{l_{i,i}} \left(a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} l_{j,k} \right) \tag{6.32}$$

Algoritma Dekomposisi Cholesky

1. Masukkan matriks A, dan vektor B beserta ukurannya n.
2. Untuk elemen matriks L, hitung menggunakan Persamaan (6.32).

3. Untuk nilai diagonal utama matriks L, hitung menggunakan Persamaan (6.31).
4. Untuk memperoleh matriks L^T, lakukan transpose pada matriks L.
5. Untuk memperoleh nilai x,

- Hitung vektor t menggunakan Persamaan (6.25).
- Hitung vektor x menggunakan Persamaan (6.24), dimana matriks $U = L^T$.

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi pada `R` untuk melakukan dekomposisi Cholesky. Fungsi tersebut disajikan pada sintaks berikut:

```
cholesky_solve <- function(a, b=NULL){
    m <- nrow(a)

# membentuk matriks L dengan elemen nol
    L = diag(0,m)

# Perhitungan elemen matriks L
    for(i in 1:m){
        for(k in 1:i){
            p_sum <- 0
            for(j in 1:k)
                p_sum <- p_sum + L[j,i]*L[j,k]

# Pehitungan elemen diagonal utama
            if(i==k)
                L[k,i]<-sqrt(a[i,i]-p_sum)
            else
                L[k,i]<-(a[k,i]-p_sum)/L[k,k]
        }
    }

# Perhitungan elemn matriks L*
    tL <- t(L)

# penyelesaian persamaan linier
    if(is.null(b)){
      return(list(L = L, tL = tL, a = a))
    }else{

      # forward substitution
      t <- forwardsolve(L, b)
```

```
    # backward substitution
    x <- backsolve(tL, t)
    return(list(L = L, tL = tL, a = a, result=x))
   }
}
```

Contoh 6.8. Dengan menggunakan fungsi `cholesky_solve()`, lakukan dekomposisi pada matriks berikut! Lakukan pengecekan pada hasil dekomposisi apakah hasil kali matriks dekomposisi akan menghasilkan matriks semula!

$$\begin{bmatrix} 9 & -3 & 6 \\ -3 & 17 & -10 \\ 6 & -10 & 12 \end{bmatrix}$$

Jawab:

Dekomposisi Cholesky menggunakan fungsi `cholesky_solve()`, disajikan pada sintaks berikut:

```
a <- matrix(c(9,-3,6,-3,17,-10,6,-10,12),3)

# dekomposisi Cholesky
(decomp<-cholesky_solve(a))
```

```
## $L
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    0    4   -2
## [3,]    0    0    2
##
## $tL
##      [,1] [,2] [,3]
## [1,]    3    0    0
## [2,]   -1    4    0
## [3,]    2   -2    2
##
## $a
##      [,1] [,2] [,3]
## [1,]    9   -3    6
## [2,]   -3   17  -10
## [3,]    6  -10   12
```

```
# mengecek hasil dekomposisi
decomp$tL %*% decomp$L
```

```
##      [,1] [,2] [,3]
## [1,]    9   -3    6
## [2,]   -3   17  -10
## [3,]    6  -10   12
```

Fungsi lain yang dapat digunakan untuk melakukan dekomposisi Cholesky adalah menggunakan fungsi `chol()` pada paket `Matrix`. Pada fungsi tersebut, kita hanya perlu menginputkan objek matrik kedalamnya. Berikut adalah contoh penerapan fungsi tersebut menggunakan matriks pada Contoh 6.8.

```
chol(a)
```

```
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    0    4   -2
## [3,]    0    0    2
```

Penting!!!

Fungsi `chol()` hanya menampilkan matriks L^T. Untuk menampilkan matriks L, kita perlu melakukan transpose

6.4.3 Dekomposisi Lainnya

Terdapat beberapa algoritma lain yang telah dikembangkan untuk melakukan dekomposisi matriks. Pada buku ini hanya akan dijelaskan secara singkat terkait fungsi yang digunakan dalam melakukan dekomposisi matriks. Algoritma yang akan dijelaskan pada *sub-chapter* ini antara lain: QR, *singular value decomposition* (SVD), dan dekomposisi eigen. Untuk algoritma lainnya, pembaca dapat membaca buku terkait atau mengecek dokumentasinya pada paket `base`.

6.4.3.1 Dekomposisi QR

Dekomposisi QR merupakan dekomposisi yang penting dalam menyelesaikan sistem persamaan linier. Dekomposisi ini juga berperan penting untuk menghitung koefisien regresi dan pengaplikasian algoritma Newton-Raphson.

Untuk memperoleh informasi terkait dekomposisi ini, pembaca dapat mengetikkan sintaks berikut pada `R`:

```
?qr
```

Berikut merupakan contoh penerapan fungsi `qr()` untuk menyelesaikan sistem persamaan linier:

```
# membuat matriks A dan B
set.seed(123)
A <- matrix((1:12)+rnorm(12), nrow=4)
b <- 2:5

# dekomposisi matriks A
qr(A)
```

```
## $qr
##            [,1]        [,2]       [,3]
## [1,] -6.3777985 -12.1257372 -19.850120
## [2,]  0.2774974  -6.3105459  -7.939245
## [3,]  0.7147777  -0.6461294   2.351193
## [4,]  0.6382310  -0.5653624   0.276672
##
## $rank
## [1] 3
##
## $qraux
## [1] 1.068915 1.512720 1.960964
##
## $pivot
## [1] 1 2 3
##
## attr(,"class")
## [1] "qr"
```

```
# memperoleh penyelesaian SPL
qr.solve(A,b)
```

```
## [1]  0.3045952 -0.1111081  0.3236862
```

6.4.3.2 *Singular Value Decomposition*

Singular value decomposition (SVD) merupakan algoritma faktorisasi matriks yang mendekomposisi matriks segiempat menjadi matriks UDV_H, dimana D merupakan mmatriks diagonal non negatif, U dan V merupakan matriks *unitary*, dan V_H merupakan matriks tanspose konjugat dari matriks V. Algoritma ini banyak digunakan dalam analisis *principal component*.

Pada R, SVD dapat dilakukan menggunakan fungsi svd() dari paket base. Berikut adalah sintaks untuk memperoleh informasi terkait fungsi tersebut:

```
?svd
```

Berikut adalah contoh penerapan fungsi svd():

```
# dekomposisi matriks A
svd(A)
```

```
## $d
## [1] 26.094305  2.727495  1.329585
##
## $u
##             [,1]       [,2]       [,3]
## [1,] -0.3684647 -0.5661362  0.6650563
## [2,] -0.4706958 -0.5703414 -0.6113237
## [3,] -0.5740273  0.4324050 -0.2589679
## [4,] -0.5596176  0.4089332  0.3419343
##
## $v
##             [,1]        [,2]       [,3]
## [1,] -0.2257101  0.87169376 -0.4349769
## [2,] -0.5201583 -0.48536069 -0.7027520
## [3,] -0.8237052  0.06763865  0.5629696
```

6.4.3.3 Dekomposisi Eigen

Proses umum yang digunakan untuk menemukan nilai eigen dan vektor eigen suatu matriks segiempat dapat dilihat sebagai proses dari dekomposisi eigen. Proses ini akan mendekomposisi matriks menjadi VDV^{-1}, dimana D merupakan matriks diagonal yang terbentuk dari nilai eigen, dan V merupakan vektor eigen. Proses dekomposisi ini akan berguna bagi pembaca yang ingin mempelajari *principal component analysis*.

Fungsi eigen() pada paket base dapat digunakan untuk melakukan dekomposisi eigen. Untuk mempelajari lebih jauh terkait fungsi ini, pambaca dapat menjalankan sintaks berikut:

```
?eigen
```

Berikut adalah contoh sintaks untuk melakukan dekomposisi eigen:

```
A <- matrix(c(2,-1,0,-1,2,-1,0,-1,2), nrow=3)

# dekomposisi matriks A
eigen(A)
```

```
## eigen() decomposition
## $values
## [1] 3.4142136 2.0000000 0.5857864
##
## $vectors
##            [,1]          [,2]      [,3]
## [1,] -0.5000000 -7.071068e-01 0.5000000
## [2,]  0.7071068  1.099065e-15 0.7071068
## [3,] -0.5000000  7.071068e-01 0.5000000
```

6.5 Metode Iterasi

Pada Chapter 6.5 kita akan membahas penyelesaian persamaan linier dengan menggunakan metode iterasi. Terdapat dua metode iterasi yang akan dibahas yaitu iterasi Jacobi dan Gauss-Seidel.

Metode iterasi dimulai dengan estimasi nilai akhir. Setelah menerapkan beberapa perlakuan pada nilai estimasi, hasil perlakuan selanjutnya menjadi nilai estimasi untuk iterasi berikutnya. Proses tersebut akan berlangsung secara terus-menerus hingga ambang batas dipenuhi. Nilai ambang batas dapat berupa jumlah iterasi maksimum atau selisih antara nilai estimasi baru dan estimasi semula lebih kecil dari suatu nilai toleransi yang ditetapkan.

Jumlah kuadrat merupakan metode yang sering digunakan untuk mengecek apakah selisih nilai estimasi baru terhadap estimasi lama lebih kecil dari nilai toleransi yang ditetapkan. Persamaan (6.33) menampilkan hubungan antara jumlah kuadrat dan nilai toleransi pada proses iterasi.

$$\sqrt{\sum_{i=1}^{n} \left(x_i^{n+1} - x_i^n\right)^2} < t_0 \tag{6.33}$$

dimana x^n merupakan iterasi ke-n dari algoritma dan t_0 merupakan nilai toleransi maksimum yang diterima.

6.5.1 Iterasi Jacobi

Untuk menyelesaikan matriks menggunakan metode iterasi, kita dapat mulai dengan premis terdapat matriks A dan vektor x dan b, sehingga $Ax = b$. Den-

gan menggunakan metode Jacobi, pertama-tama kita dapat amati bahwa terdapat matriks R dan D yang memiliki hubungan $A = R + D$. Berdasarkan kedua hubungan tersebut, dapat diturunkan operasi matriks melalui persamaan berikut:

$$Ax = b \tag{6.34}$$

$$Rx + Dx = b \tag{6.35}$$

$$Dx = b - Rx \tag{6.36}$$

$$x = D^{-1}(b - Rx) \tag{6.37}$$

Persamaan (6.37) merupakan persamaan yang dapat kita gunakan untuk memperoleh nilai x. Jika kita menulis kembali persamaan tersebut, maka kita akan memperoleh persamaan yang digunakan sebagai acuan iterasi Jacobi.

$$x^{n+1} = D^{-1}(b - Rx^n) \tag{6.38}$$

dimana D merupakan matriks diagonal dengan nilai elemen diagonal berupa diagonal utama matriks A. Invers dari matriks D secara sederhana sebagai matriks diagonal sama dengan satu dibagi dengan elemen diagonal utama matriks A. Matriks R identik dengan matriks A. Namun, diagonal utamanya bernilai nol. Suatu iterasi dikatakan konvergen jika jumlah kuadrat dari vektor $x^{(n+1)}$ dan vektor $x^{(n)}$ semakin mengecil.

Suatu persamaan linier yang hendak diselesaikan dengan menggunakan metode iterasi Jacobi harus memenuhi syarat nilai elemen diagonal utama matriks harus lebih dominan. Maksudnya adalah nilai absolut diagonal utama matriks harus lebih besar dari jumlah nilai absolut elemen matriks lainnya pada satu kolom.

Contoh 6.9. Selesaikan sistem persamaan berikut menggunakan iterasi Jacobi!

$$\begin{bmatrix} 5 & 2 & 3 \\ 2 & 7 & 4 \\ 1 & 3 & 8 \end{bmatrix} x = \begin{bmatrix} 40 \\ 39 \\ 55 \end{bmatrix}$$

Jawab:

Berdasarkan matriks A (matriks koefisien), kita dapat memastikan bahwa matriks tersebut memiliki nilai dominan pada elemen diagonal utama. Sebagai contoh:

$$|5| > |2| + |1| \quad (kolom\ 1)$$

$$|7| > |2| + |3| \quad (kolom\ 2)$$

Untuk mempermudah proses iterasi, kita akan menggunakan bantuan `R` untuk melakukan komputasi. Langkah pertama yang perlu dilakukan adalah menyiapkan matriks A, vektor b, dan vektor x (nilai taksiran awal).

```
(A <- matrix(c(5,2,1,2,7,3,3,4,8), 3))
```

```
##      [,1] [,2] [,3]
## [1,]    5    2    3
## [2,]    2    7    4
## [3,]    1    3    8
```

```
(b <- c(40,39,55))
```

```
## [1] 40 39 55
```

```
(x <- rep(0,3))
```

```
## [1] 0 0 0
```

Langkah selanjutnya adalah memperoleh invers matriks D.

```
(Dinv <- diag(1/diag(A)))
```

```
##      [,1]      [,2]  [,3]
## [1,]  0.2 0.0000000 0.000
## [2,]  0.0 0.1428571 0.000
## [3,]  0.0 0.0000000 0.125
```

Persiapan terakhir sebelum iterasi dilakukan adalah menyiapkan matriks R.

```
(R<-A-diag(diag(A)))
```

```
##      [,1] [,2] [,3]
## [1,]    0    2    3
## [2,]    2    0    4
## [3,]    1    3    0
```

Iterasi selanjutnya dilakukan menggunakan Persamaan (6.38).

iterasi 1

```
(x1 <- Dinv %*% (b-R%*%x))
```

```
##          [,1]
## [1,] 8.000000
## [2,] 5.571429
## [3,] 6.875000
```

iterasi 2

```
(x2 <- Dinv %*% (b-R%*%x1))
```

```
##            [,1]
## [1,]  1.6464286
## [2,] -0.6428571
## [3,]  3.7857143
```

iterasi 3

```
(x3 <- Dinv %*% (b-R%*%x2))
```

```
##          [,1]
## [1,] 5.985714
## [2,] 2.937755
## [3,] 6.910268
```

Selama proses iterasi,jumlah akar jumlah kuadrat dihitung. Sebagai contoh berikut disajikan akar jumlah kuadrat pada iterasi ke-3:

```
sqrt(sum(x3-x2)^2)
```

```
## [1] 11.04445
```

Selama proses iterasi nilai tersebut terus mengecil. Iterasi dihantikan jika nilai akar jumlah kuadrat tersebut lebih kecil dari nilai toleransi. Pada contoh ini digunakan nilai toleransi 10^{-7}.

Proses iterasi berlangsung sampai dengan iterasi ke-62 dengan nilai x akhir sebagai berikut:

$$x = \begin{bmatrix} 4 \\ 1 \\ 6 \end{bmatrix}$$

Algoritma Iterasi Jacobi

1. Masukkan matriks A, dan vektor B beserta ukurannya n.
2. Hitung invers matriks D, dimana nilai invernya merupakan matriks diagonal dari satu per diagonal utama matriks A.
3. Hitung matriks R, dimana R merupakan selisih matriks A dikurangi dengan matriks diagonal dengan entri dari diagonal utama matriks A.
4. Tetapkan vektor x estimasi.
5. Tetapkan nilai toleransi maksimum yang dapat diterima.
6. Lakukan iterasi menggunakan Persamaan (6.38).
7. Hitung akar jumlah kuadrat dari vektor x^{n+1} dan vektor x^n.
8. Jadikan nilai x^{n+1} sebagai nilai taksiran x untuk iterasi berikutnya.
9. Hentikan proses iterasi jika telah memenuhi syarat yang ditampilkan pada Persamaan (6.33).

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi sebuah fungsi untuk melakukan iterasi Jacobi. Berikut sintaks yang digunakan:

```
jacobi <- function(a, b, tol=1e-7, maxiter=100){
  n <- length(b)
  iter <- 0

  Dinv <- diag(1/diag(a))
  R <- a-diag(diag(a))
  x <- rep(0,n)
  x_new <- rep(tol, n)

  while(sqrt(sum(x_new-x)^2)>tol){
            if(iter>maxiter){
              warning("iterasi maksimum tercapai")
              break
            }
            x <- x_new
            x_new <- Dinv %*% (b - R %*% x)
            iter <- iter+1
  }
  return(list(X = x_new, iter=iter))

}
```

Berikut adalah penerpan fungsi `jacobi()` tersebut:

```
jacobi(A,b)
```

```
## $X
##      [,1]
## [1,]    4
## [2,]    1
## [3,]    6
##
## $iter
## [1] 62
```

Contoh 6.10. Selesaikan sistem persamaan berikut menggunakan fungsi `jacobi()`

$$\begin{bmatrix} 27 & 6 & -1 \\ 6 & 15 & 2 \\ 1 & 1 & 54 \end{bmatrix} x = \begin{bmatrix} 85 \\ 72 \\ 110 \end{bmatrix}$$

Jawab:

Matriks A (matriks koefisien) berdasarkan sistem persamaan linier tersebut telah memenuhi syarat dari algoritma Jacobi (nilai diagonal utama dominan dibanding nilai lainnya pada satu kolom). Penyelesaian sistem persamaan tersebut, sebagai berikut:

```
A <- matrix(c(27,6,1,6,15,1,-1,2,54), 3)
b <- c(85,72,110)

jacobi(A,b)
```

```
## $X
##          [,1]
## [1,] 2.425476
## [2,] 3.573016
## [3,] 1.925954
##
## $iter
## [1] 17
```

Nilai vektor x sesungguhnya dapat diperoleh menggunakan fungsi `solve()`.

```
solve(A,b)
```

```
## [1] 2.425476 3.573016 1.925954
```

Berdasarkan hasil perhitungan, vektor x hasil iterasi memiliki nilai identik dengan nilai penyelesaian yang sebenarnya.

Perlu diperhatikan dalam penggunaan fungsi `jacobi()` syarat utama matriks haruslah terpenuhi, seperti: nilai diagonal matriks A lebih besar dari nilai elemen lainnya pada satu kolom. Selain itu, nilai diagonal matriks D tidak boleh sama dengan nol agar inver matriks D dapat diperoleh. Jika syarat-syarat tersebut terpenuhi, maka metode Jacobi dapat diterapkan. Jika tidak terpenuhi, maka penyelesaian yang konvergen mungkin masih dapat diperoleh meskipun penulis tidak dapat menjamin hal tersebut dapat terjadi.

6.5.2 Iterasi Gauss-Seidel

Metode iterasi Gauss-Seidel melakukan dekomposisi pada matriks A menjadi matriks segitiga atas U dan matriks segitiga bawah L. Dekomposisi ini tidak sama dengan dekomposisi LU pada Chapter 6.4.1. Matriks U pada metode Gauss-Seidel merupakan elemen (entri) matriks A pada bagian atas diagonal utama, sedangkan matriks L merupakan elemen diagonal utama dan bagian bawah diagonal utama matriks A. Elemen selain yang penulis sebutkan pada kedua matriks tersebut akan bernilai nol. Persamaan iterasi Gauss-Seidel ditampilkan pada Persamaan (6.39).

$$x^{n+1} = L^{-1}\left(b - Ux^{n}\right) \tag{6.39}$$

Syarat agar suatu sistem persamaan linier dapat diselesaikan menggunakan metode Gauss-Seidel adalah matriks harus memiliki nilai diagonal utama yang dominan. Maksudnya, nilai absolut diagonal utama lebih besar dari jumlah nilai absolut elemen lainnya dalam satu kolom. Jika syarat ini tidak terpenuhi maka metode ini tidak akan memperoleh penyelesaian yang konvergen.

Contoh 6.11. Selesaikan sistem persamaan pada Contoh 6.10 menggunakan iterasi Gauss-Seidel!

Jawab:

Kita akan kembali menggunakan bantuan `R` untuk melakukan kalkulasi pada proses iterasi Gauss-Seidel. Kita telah melakukan pengecekan pada sistem persamaan linier pada contoh tersebut dan menghasilkan kesimpulan bahwa persamaan linier tersebut dapat diselesaikan dengan metode Gauss-Seidel. Langkah selanjutnya adalah membentuk matriks L dan U.

```
# membentuk matriks U dan L dari matriks A
(L <- U <- A)
```

```
##      [,1] [,2] [,3]
## [1,]   27    6   -1
## [2,]    6   15    2
## [3,]    1    1   54
```

```
# membentuk matriks L dari entri bagian bawah diagonal utama matriks A
L[upper.tri(A, diag=FALSE)]<-0
L
```

```
##      [,1] [,2] [,3]
## [1,]   27    0    0
## [2,]    6   15    0
## [3,]    1    1   54
```

```
# membentuk matriks U dari entri bagian atas diagonal utama matriks A
U[lower.tri(A, diag=TRUE)]<-0
U
```

```
##      [,1] [,2] [,3]
## [1,]    0    6   -1
## [2,]    0    0    2
## [3,]    0    0    0
```

Selanjutya lakukan invers terhadap matriks L menggunakn fungsi `solve()`.

```
(Linv <- solve(L))
```

```
##               [,1]         [,2]       [,3]
## [1,]  0.0370370370  0.000000000 0.00000000
## [2,] -0.0148148148  0.066666667 0.00000000
## [3,] -0.0004115226 -0.001234568 0.01851852
```

Tetapkan nilai estimasi awal dan nilai toleransi yang dikehendaki. Nilai toleransi pada proses ini ditetapkan sebesar 10^-7.

```
# tebakan awal nilai x
(x <- rep(0, length(b)))
```

```
## [1] 0 0 0
```

Lakukan iterasi menggunakan Persamaan (6.39).

Iterasi 1

```
(x1 <- Linv %*% (b - U %*% x))
```

```
##          [,1]
## [1,] 3.148148
## [2,] 3.540741
## [3,] 1.913169
```

```
# akar jumlah kuadrat
sqrt(sum(x1-x)^2)
```

```
## [1] 8.602058
```

Iterasi 2

```
(x2 <- Linv %*% (b - U %*% x1))
```

```
##          [,1]
## [1,] 2.432175
## [2,] 3.572041
## [3,] 1.925848
```

```
# akar jumlah kuadrat
sqrt(sum(x2-x1)^2)
```

```
## [1] 0.6719939
```

Iterasi terus dilakukan sampai dengan nilai akar jumlah kuadrat lebih kecil dari nilai toleransi. Setelah iterasi ke-7 diperoleh nilai vektor x sebesar:

$$x = \begin{bmatrix} 2,425476 \\ 3,573016 \\ 1,925954 \end{bmatrix}$$

Algoritma Iterasi Gauss-Seidel

1. Masukkan matriks A, dan vektor B beserta ukurannya n.

2. Lakukan dekomposisi LU, dimana matriks L merupakan matriks segitiga bawah dengan nilai entri diagonal utama matriks A dan bagian bawah diagonalnya dan matriks U merupakan matriks segitiga atas dengan entri berasal dari elemen atas diagonal utama matriks A. Isi elemen lain yang tidak disebut pada kedua matriks tersebut dengan nol.
3. Tetapkan vektor x estimasi.
4. Tetapkan nilai toleransi maksimum yang dapat diterima.
5. Lakukan iterasi menggunakan Persamaan (6.39).
6. Hitung akar jumlah kuadrat dari vektor x^{n+1} dan vektor x^n.
7. Jadikan nilai x^{n+1} sebagai nilai taksiran x untuk iterasi berikutnya.
8. Hentikan proses iterasi jika telah memenuhi syarat yang ditampilkan pada Persamaan (6.33).

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi sebuah fungsi untuk melakukan iterasi Gauss-Seidel. Berikut sintaks yang digunakan:

```
gauss_seidel <- function(a, b, tol=1e-7, maxiter=100){
  n <- length(b)
  iter <- 0


  L <- U <- a
  L[upper.tri(a, diag=FALSE)] <- 0
  U[lower.tri(a, diag=TRUE)] <- 0
  Linv <- solve(L)

  x <- rep(0,n)
  x_new <- rep(tol, n)

  while(sqrt(sum(x_new-x)^2)>tol){
            if(iter>maxiter){
              warning("iterasi maksimum tercapai")
              break
            }
            x <- x_new
            x_new <- Linv %*% (b - U %*% x)
            iter <- iter+1
  }
      return(list(X = x_new, iter=iter))
}
```

Contoh 6.12. Selesaikan sistem persamaan pada Contoh 6.10 menggunakan fungsi `gauss_seidel()`!

Jawab:

Penyelesaiansistem persamaan linier tersebut menggunakan fungsi `gauss_seidel()` disajikan pada sintaks berikut:

```
gauss_seidel(A,b)
```

```
## $X
##           [,1]
## [1,] 2.425476
## [2,] 3.573016
## [3,] 1.925954
##
## $iter
## [1] 7
```

6.6 Studi Kasus

Aljabar linier banyak diaplikasikan baik dalam bidang *engineering*, fisika, sampai dengan statistika. Pada *sub-chapter* ini penulis akan menjelaskan penerapan aljabar linier pada metode kuadrat terkecil dan aliran massa dalam reaktor. Untuk penerapan lainnya pembaca dapat membaca buku lainnya terkait aljabar linier.

6.6.1 Metode Kuadrat Terkecil

Metode kuadrat terkecil merupakan salah satu aplikasi penerapan aljabar linier yang paling populer. Intuisi dibalik metode ini adalah bagaimana kita meminimalkan jarak antara sejumlah titik dengan garis regresi. Misalkan kita menggambarkan scatterplot antara dua buah variabel. Pola yang terbentuk dari plot tersebut adalah terjadi korelasi positif antara variabel pada sumbu x dan sumbu y. Kita ingin menggambarkan garis regresi terbaik yang dapat menangkap seluruh pola tersebut. Garis regresi terbaik terjadi ketika jumlah kuadrat jarak antara titik observasi dan garis regresi yang terbentuk seminimal mungkin.

Untuk lebih memahaminya kita akan melakukan latihan menggunakan dataset `trees` yang berisi data hasil pengukuran kayu dari pohon yang ditebang. Pada dataset ini terdapat 31 observasi dan 3 buah kolom. Keterangan dari ketiga buah kolom tersebut adalah sebagai berikut:

- `Girth`: diameter pohon dalam satuan *inch*.
- `Height`: tinggi pohon dalam satuan *feet*.
- `Volume`: volume kayu dalam satuan *cubic feet*.

Untuk mengecek 6 observasi pertama dan struktur data, jalankan sintaks berikut:

```
head(trees)
```

```
##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2
## 4  10.5     72   16.4
## 5  10.7     81   18.8
## 6  10.8     83   19.7
```

```
str(trees)
```

```
## 'data.frame':    31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

Scatterplot matriks sangat bagus untuk mengecek korelasi antar variabel dalam dataset tersebut. Berikut adalah sintaks untuk membuatnya:

Kita ingin membuat sebuah model linier untuk memprediksi `Volume` kayu berdasarkan variabel `Girth` dan `Heiht` atau volume sebagia fungsi dari variabel `Girth` dan `Heiht`. Kita dapat menuliskan relasi antara variabel volume sebagai fungsi dari variabel `Girth` dan `Heiht` menggunakan Persamaan (6.40).

$$Volume = \beta_{girth} Girth + \beta_{height} Height + \beta_0 \tag{6.40}$$

dimana β_0 merupakan intersep persamaan regresi linier dan nilai β lainnya merupakan koefisien dari variabel `Girth` dan `Heiht`. Variabel `Volume` disebut sebagai variabel respon, sedangkan variabel `Girth` dan `Heiht` disebut sebagai variabel prediktor.

Metode kuadrat terkecil berusaha memperoleh seluruh koefisien variabel dan intersep dari persamaan regresi linier. Berdasarkan yang telah penulis jelaskan garis regresi terbaik adalah garis yang memiliki nilai kuadrat terkecil jarak antara titik observasi dan garis regresi. Dasar dari metode kuadrat terkecil merupakan persamaan yang relatif sederhana yang ditunjukkan pada Persamaan (6.41).

$$A^T A = A^T b \tag{6.41}$$

LUfig-1.bb

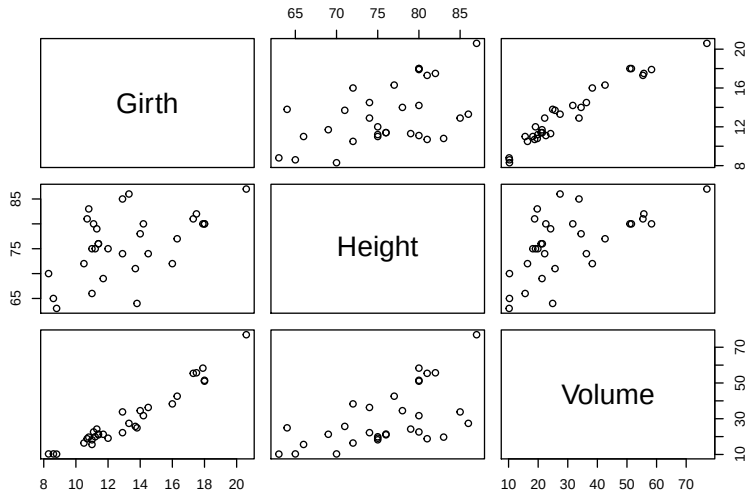

Figure 6.2: Scatterplot matriks dataset trees
(#fig:trees, LUfig)

dimana b merupakan vektor dari variabel respon (`Volume`) dan matrik A merupakan matriks variabel prediktor (variabel `Girth` dan `Heiht`).

Untuk menginputkan intercept kedalam persamaan linier kita perlu menmabhakan satu kolom di awal matriks A yang berisi nilai 1. Berikut adalah sintaks yang digunakan untuk membentuk matriks A:

```
# membentuk matriks A
pred <- cbind(intercept=1, Girth=trees$Girth, Height=trees$Height)
head(A)
```

```
##      [,1] [,2] [,3]
## [1,]   27    6   -1
## [2,]    6   15    2
## [3,]    1    1   54
```

Langkah selanjutnya adalah membentuk matriks b. Berikut adalah sintaks yang digunakan:

```
resp<- trees$Volume
head(resp)
```

```
## [1] 10.3 10.3 10.2 16.4 18.8 19.7
```

Untuk memperoleh koefisien β, kita dapat mencarinya dengan cara menyelesaikan Persamaan (6.41). Berikut adalah sintaks yang digunakan:

```
A <- t(pred) %*% pred
b <- t(pred) %*% resp

Ab <- cbind(A,b)
(x <- gauss_jordan(Ab))
```

```
##           intercept Girth Height
## intercept         1     0      0 -57.9876589
## Girth             0     1      0   4.7081605
## Height            0     0      1   0.3392512
```

Berdasarkan hasil yang diperoleh, persamaan linier yang terbentuk disajikan pada Persamaan (6.42).

$$Volume = 4.7081605Girth + 0.3392512Height - 57.9876589 \tag{6.42}$$

Pembaca juga dapat menggunakan fungsi lain untuk memperoleh nilai koefisien tersebut, seperti: `lu_solve()`dan`solve()`. untuk fungsi `jacobi()` dan `gauss_seidel()`, kita harus pastikan syarat-syarat terkait metode tersebut. Berikut adalah contoh penyelesaian menggunakan sintaks lainnya:

```
# metode LU
lu_solve(A,b)
```

```
## $P
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
##
## $L
##           [,1]     [,2] [,3]
## [1,]  1.00000 0.000000    0
## [2,] 13.24839 1.000000    0
## [3,] 76.00000 1.054369    1
##
## $U
##           intercept    Girth    Height
```

```
## intercept        31 410.7000 2356.0000
## Girth             0 295.4374  311.5000
## Height            0   0.0000  889.5641
##
## $result
##              [,1]
## [1,] -57.9876589
## [2,]   4.7081605
## [3,]   0.3392512
```

```
# fungsi solve()
solve(A,b)
```

```
##                   [,1]
## intercept -57.9876589
## Girth       4.7081605
## Height      0.3392512
```

R juga menyediakan fungsi untuk membentuk model regresi linier. Fungsi yang digunakan adalah `lm()`. Berikut sintaks yang digunakan untuk membentuk model linier menggunakan fungsi `lm()`:

```
lm(Volume~Girth+Height, data=trees)
```

```
##
## Call:
## lm(formula = Volume ~ Girth + Height, data = trees)
##
## Coefficients:
## (Intercept)        Girth       Height
##    -57.9877       4.7082       0.3393
```

6.6.2 Aliran Massa Dalam Reaktor

Pada *sub-chapter* ini penulis akan memberikan penerapan aljabar linier untuk menghitung konsentrasi suatu zat atau parameter lingkungan dalam reaktor yang saling terhubung. Pada contoh kasus kali ini diasumsikan terdapat lima buah reaktor yang saling terhubung satu sama lain sesuai Gambar 6.3. Debit air ($\frac{m^3}{detik}$) dan konsentrasi zat pencemar ($\frac{mg}{m^3}$) disajikan pula diagram alir tersebut. Diasumsikan kelima buah reaktor tersebut dalam kondisi *steady* dan volume reaktor diasumsikan sama. Kesetimbangan massa persatuan waktu dalam kondisi *steady* disajikan pada Persamaan (6.43).

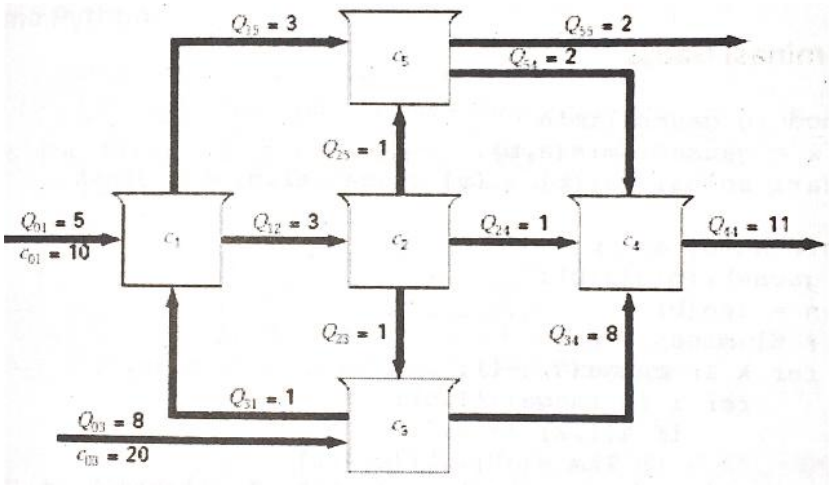

Figure 6.3: Aliran massa dalam reaktor.

$$m_in = m_out \tag{6.43}$$

$$Q_{in}C_{in} = Q_{out}C_{out} \tag{6.44}$$

Berdasarkan Gambar 6.3, dapat dibentuk lima buah sistem persamaan linier. Persamaan linier yang terbentuk disajikan sebagai berikut:

$$\begin{gathered} 6c_1 - c_3 = 50 \\ -3c_1 + 3c_2 = 0 \\ -c_2 + 9c_3 = 160 \\ -c_2 - 8c_3 + 11c_4 - 2c_5 = 0 \\ -3c_1 - c_2 + 4c_5 = 0 \end{gathered}$$

Untuk menyelesaiakan sistem persamaan linier tersebut dan memperoleh nilai c dari masing-masing reaktor, kiat perlu mengubahnya dulu kedalam bentuk matriks $Ax = b$. Berikut adalah matriks yang terbentuk:

$$\begin{bmatrix} 6 & 0 & -1 & 0 & 0 \\ -3 & 3 & 0 & 0 & 0 \\ 0 & -1 & 9 & 0 & 0 \\ 0 & -1 & -8 & 11 & -2 \\ -3 & -1 & 0 & 0 & 4 \end{bmatrix} \quad c = \begin{bmatrix} 50 \\ 0 \\ 160 \\ 0 \\ 0 \end{bmatrix}$$

Kita akan menyelesaikannya dengan menggunakan metode elminasi Gauss-Jordan, dekomposisi LU, iterasi Jacobi, dan iterasi Gauss-Seidel. Untuk dapat menyelesaikannya menggunakan metode-metode tersebut pada R, kita perlu membentuk matriksnya terlebih dahulu:

```
(A <- matrix(c(6,-3,0,0,-3,
               0,3,-1,-1,-1,
               -1,0,9,-8,0,
               0,0,0,11,0,
               0,0,0,-2,4),nrow=5))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    6    0   -1    0    0
## [2,]   -3    3    0    0    0
## [3,]    0   -1    9    0    0
## [4,]    0   -1   -8   11   -2
## [5,]   -3   -1    0    0    4
```

```
(b <- c(50,0,160,0,0))
```

```
## [1]  50   0 160   0   0
```

Metode Eliminasi Gauss-Jordan

```
gauss_jordan(cbind(A,b))
```

```
##                       b
## [1,] 1 0 0 0 0 11.50943
## [2,] 0 1 0 0 0 11.50943
## [3,] 0 0 1 0 0 19.05660
## [4,] 0 0 0 1 0 16.99828
## [5,] 0 0 0 0 1 11.50943
```

Metode Dekomposisi LU

```
lu_solve(A,b)
```

```
## $P
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    1    0    0    0
## [3,]    0    0    1    0    0
## [4,]    0    0    0    1    0
```

```
## [5,]    0    0    0    0    1
##
## $L
##      [,1]       [,2]       [,3] [,4] [,5]
## [1,]  1.0  0.0000000  0.0000000    0    0
## [2,] -0.5  1.0000000  0.0000000    0    0
## [3,]  0.0 -0.3333333  1.0000000    0    0
## [4,]  0.0 -0.3333333 -0.9245283    1    0
## [5,] -0.5 -0.3333333 -0.0754717    0    1
##
## $U
##      [,1] [,2]          [,3] [,4] [,5]
## [1,]    6    0 -1.000000e+00    0    0
## [2,]    0    3 -5.000000e-01    0    0
## [3,]    0    0  8.833333e+00    0    0
## [4,]    0    0  0.000000e+00   11   -2
## [5,]    0    0  1.110223e-16    0    4
##
## $result
## [1] 11.50943 11.50943 19.05660 16.99828 11.50943
```

Metode Iterasi Jacobi

```
jacobi(A,b, maxiter=100)
```

```
## $X
##          [,1]
## [1,] 11.50943
## [2,] 11.50943
## [3,] 19.05660
## [4,] 16.99828
## [5,] 11.50943
##
## $iter
## [1] 17
```

Metode Iterasi Gauss-Seidel

```
gauss_seidel(A,b, maxiter=200)
```

```
## $X
##          [,1]
## [1,] 11.50943
## [2,] 11.50943
```

```
## [3,] 19.05660
## [4,] 16.99828
## [5,] 11.50943
##
## $iter
## [1] 7
```

Berdasarkan seluruh metode tersebut, diperoleh konsentrasi zat pencemar pada masing-masing reaktor adalah sebagai berikut:

$$
\begin{aligned}
c_1 &= 11,50943 \frac{mg}{m^3} \\
c_2 &= 11,50943 \frac{mg}{m^3} \\
c_3 &= 19,05660 \frac{mg}{m^3} \\
c_4 &= 16,99828 \frac{mg}{m^3} \\
c_5 &= 11.50943 \frac{mg}{m^3}
\end{aligned}
$$

6.7 Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
2. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
3. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
4. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
5. Sanjaya, M. 2015. **Metode Numerik Berbasis Phython**. Penerbit Gava Media: Yogyakarta.
6. Suparno, S. 2008. **Komputasi untuk Sains dan Teknik Edisi II**. Departemen Fisika-FMIPA Universitas Indonesia.

6.8 Latihan

1. Selesaikan sistem persamaan linier berikut menggunakan eliminasi Gauss!

$$
\begin{aligned}
-4x + 4y &= -1 \\
-2x + 2y - 3z &= -3 \\
3x + 1y - 3z &= -3
\end{aligned}
$$

2. Carilah penyelesaian dari sistem persamaan linier soal no.1 menggunakan algoritma dekomposisi LU!

3. Tunjukan 5 iterasi pertama sistem persamaan linier berikut menggunakan algoritma Jacobi dan Gauss-Seidel!

$$\begin{aligned} 3x + 2y - 1z &= -3 \\ -3x - 3y - 3z &= 9 \\ 1y - 1z &= -1 \end{aligned}$$

4. Gunakan fungsi `jacobi()` dan `gauss_seidel()` untuk menyelesaikan sistem persamaan linier pada soal no.4 dan tentukan metode mana yang paling cepat memperoleh penyelesaian? (**petunjuk**: gunakan fungsi `system.time()` dan jumlah iterasi yang diperlukan untuk memperoleh hasil yang konvergen)
5. Apakah yang terjadi jika kita menginputkan matriks segiempat A kedalam fungsi `solve()` dan apa yang akan terjadi jika selanjutnya argumen pada fungsi tersebut juga menyertakan vektor b?
6. Dengan menggunakan dataset `mtcars` buatlah persamaan linier variabel `mpg` sebagai fungsi dari variabel `wt`, `hp`, dan `qsec` menggunakan algoritma dekomposisi LU?

Chapter 7

Akar Persamaan Non-Linier

Persamaan non-linier dapat diartikan sebagai persamaan yang tidak mengandung syarat seperti persamaan linier, sehingga persamaan non-linier dapat merupakan:

a. Persamaan yang memiliki pangkat selain satu (misal: x^2)
b. Persamaan yang mempunyai produk dua variabel (misal: xy)

Dalam penyelesaian persamaan non-linier diperlukan akar-akar persamaan non-linier, dimana akar sebuah persamaan non-linier $f(x) = 0$ merupakan nilai x yang menyebabkan nilai $f(x)$ sama dengan nol. Dalam hal ini dapat disimpulkan bahwa akar-akar penyelesaian persamaan non-linier merupakan titik potong antara kurva $f(x)$ dengan sumbu x. Ilustrasi penjelasan tersebut ditampilkan pada Gambar 7.1.

Contoh sederhana dari penentuan akar persamaan non-linier adalah penentuan akar persamaan kuadratik. Secara analitik penentuan akar persamaan kuadratik dapat dilakukan menggunakan Persamaan (7.1).

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4a}}{2a} \tag{7.1}$$

Untuk masalah yang lebih rumit, penyelesaian analitik sudah tidak mungkin dilakukan. Metode numerik dapat digunakan untuk menyelesaikan masalah yang lebih kompleks. Untuk mengetahui apakah suatu persamaan non-linier memiliki akar-akar penyelesaian atau tidak, diperlukan analisa menggunakan Teorema berikut:

Teorema 7.1 (root)**.** *Suatu range x=[a,b] mempunyai akar bila f(a) dan f(b) berlawanan tanda atau memenuhi f(a).f(b)<0*

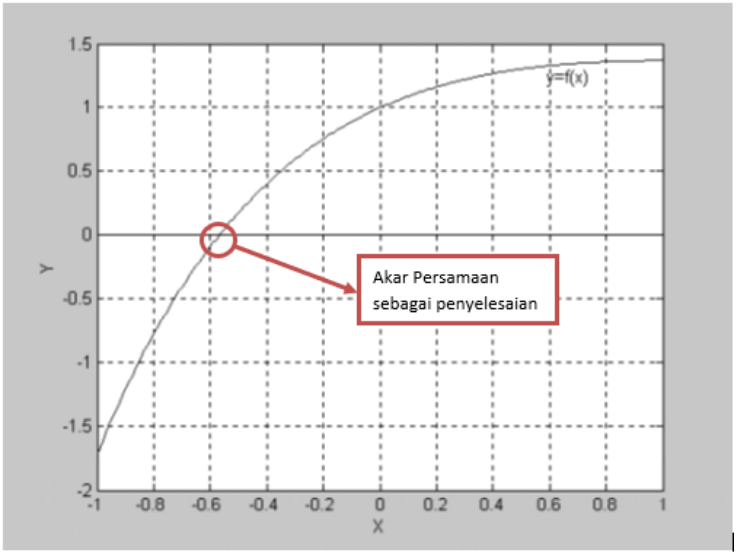

Figure 7.1: Penyelesaian persamaan non-linier.

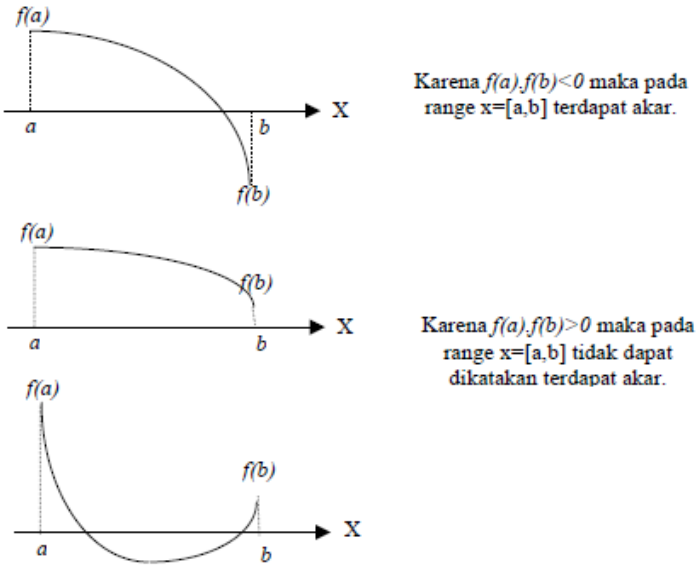

Figure 7.2: Ilustrasi teorema Bolzano.

Untuk memahami teorema tersebut perhatikan ilustrasi pada Gambar 7.2.

Pada Chapter 7 ini, akan dilakukan sejumlah pembahasan antara lain:

- penentuan akar persamaan dengan metode tertutup
- penentuan akar persamaan dengan metode terbuka
- fungsi-fungsi `R` untuk mementukan akar persamaan non-linier
- studi kasus

7.1 Metode Tertutup

Metode tertutup disebut juga metode *bracketing*. Disebut sebagai metode tertutup karena dalam pencarian akar-akar persamaan non-linier dilakukan dalam suatu selang $[a, b]$.

7.1.1 Metode Tabel

Penyelesaian persamaan non-linier menggunakan metode tabel dilakukan dengan membagi persamaan menjadi beberapa area, dimana untuk $x = [a, b]$ dibagi sebanyak N bagian dan pada masing-masing bagian dihitung nilai $f(x)$ sehingga diperoleh nilai $f(x)$ pada setian N bagian.

Bila nilai $f(x_k) = 0$ atau mendekati nol, dimana $a \leq k \leq b$, maka dikatakan bahwa x_k adalah penyelesaian persamaan $f(x)$. Bila tidak ditemukan, dicari nilai $f(x_k)$ dan $f(x_{k+1})$ yang berlawanan tanda. Bila tidak ditemukan, maka persamaan tersebut dapat dikatakan tidak mempunyai akar untuk rentang $[a, b]$.

Bila akar persamaan tidak ditemukan, maka ada dua kemungkinan untuk menentukan akar persamaan, yaitu:

a. Akar persamaan ditentukan oleh nilai mana yang lebih dekat. Bila $f(x_k) \leq f(x_{k+1})$, maka akarnya x_k. Bila $f(x_{k+1}) \leq f(x_k)$, maka akarnya x_{k+1}.
b. Perlu dicari lagi menggunakan rentang $x = [x_k, x_{k+1}]$.

Secara grafis penyelesaian persamaan non-linier menggunakan metode table disajikan pada Gambar 7.3.

Algoritma Metode Tabel

1. Definisikan fungsi $f(x)$
2. Tentukan rentang untuk x yang berupa batas bawah a dan batas atas b.
3. Tentukan jumlah pembagi N
4. Hitung step pembagi

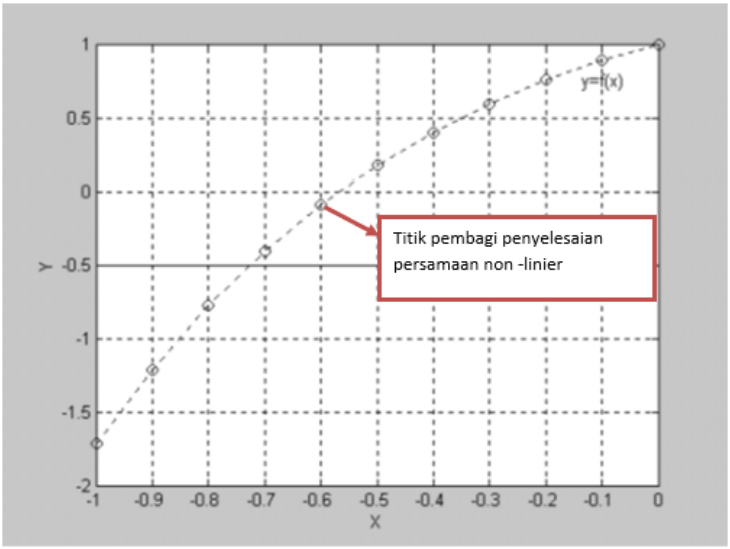

Figure 7.3: Ilustrasi metode tabel.

$$h = \frac{b+a}{N} \tag{7.2}$$

5. Untuk $i = 0$ s/d N, hitung:

$$x_i = a + i.h \tag{7.3}$$

$$y_i = f(x_i) \tag{7.4}$$

6. Untuk $i = 0$ s/d N, dimana

- Bila $f(x) = 0$, maka akarnya x_k
- Bila $f(a) f(b) < 0$, maka:
  - $f(x_k) \leq f(x_{k+1})$, maka akarnya x_k
  - Bila tida, x_{k+1} adalah penyelesaian atau dapat dikatakan penyelesaian berada diantara x_k dan x_{k+1}.

Kita dapat membuat suatu fungsi pada `R` untuk melakukan proses iterasi pada metode Tabel. Fungsi `root_table()` akan melakukan iterasi berdasarkan step algoritma 1 sampai 5. Berikut adalah sintaks yang digunakan:

Table 7.1: Penyelesaian persamaan x+exp(x)=0

x	fx
-1.0	-0.6321206
-0.9	-0.4934303
-0.8	-0.3506710
-0.7	-0.2034147
-0.6	-0.0511884
-0.5	0.1065307
-0.4	0.2703200
-0.3	0.4408182
-0.2	0.6187308
-0.1	0.8048374
0.0	1.0000000

```
root_table <- function(f, a, b, N=20){
    h <- abs((a+b)/N)
    x <- seq(from=a, to=b, by=h)
    fx <- rep(0, N+1)
    for(i in 1:(N+1)){
      fx[i] <- f(x[i])
    }
    data <- data.frame(x=x, fx=fx)
    return(data)
}
```

Contoh 7.1. Carilah akar persamaan $f(x) = x + e^x$ pada rentang $x = [-1, 0]$?

Jawab:

Sebagai permulaan, jumlah pembagi yang digunakan adalah $N = 10$. Dengan menggunakan fungsi `root_table()` diperoleh hasil yang disajikan pada Tabel 7.1.

```
tabel <- root_table(f=function(x){x+exp(x)},
                    a=-1, b=0, N=10)
```

Berdasarkan Tabel 7.1 diperoleh penyelesaian di antara $-0,6$ dan $-0,5$ dengan nilai $f(x)$ masing-masing sebesar $-0,0512$ dan $-0,1065$, sehingga dapat diambil penyelesaian $x = -0,6$. Kita dapat terus melakukan iterasi sampai memperoleh nilai $f(x) <$ nilai toleransi dengan terus merubah rentang yang diberikan. Iterasi berikutnya dengan nilai pembagi sama dan rentang nilai $x = [-0,6; -0,5]$ diperoleh nilai $x = -0,57$ dan $f(x) = 0,00447$.

Untuk melihat gambaran lokasi akar, kita dapat pulang mengeplotkan data menggunakan fungsi plot. Berikut adalah fungsi yang digunakan:

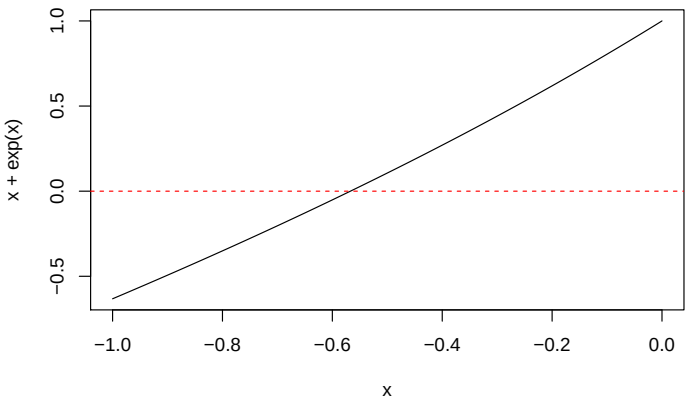

Figure 7.4: Plot fungsi x+exp(x) pada rentang -1 sampai 0.

Untuk mengetahui lokasi akar dengan lebih jelas, kita dapat memperkecil lagi rentang nilai yang dimasukkan dalam fungsi `curve()`.

Metode tabel pada dasarnya memiliki kelemahan yaitu cukup sulit untuk mendapatkan error penyelesaian yang cukup kecil, sehingga metode ini jarang sekali digunakan untuk menyelesaikan persamaan non-linier. Namun, metode ini cukup baik digunakan dalam menentukan area penyelesaian sehingga dapat dijadikan acuan metode lain yang lebih baik.

7.1.2 Metode Biseksi

Prinsip metode bagi dua adalah mengurung akar fungsi pada interval $x = [a, b]$ atau pada nilai x batas bawah a dan batas atas b. Selanjutnya interval tersebut terus menerus dibagi 2 hingga sekecil mungkin, sehingga nilai hampiran yang dicari dapat ditentukan dengan tingkat toleransi tertentu. Untuk lebih memahami metode biseksi, perhatikan visualisasi pada Gambar 7.5.

Metode biseksi merupakan metode yang paling mudah dan paling sederhana dibanding metode lainnya. Adapun sifat metode ini antara lain:

1. Konvergensi lambat
2. Caranya mudah
3. Tidak dapat digunakan untuk mencari akar imaginer
4. Hanya dapat mencari satu akar pada satu siklus.

Algoritma Metode Biseksi

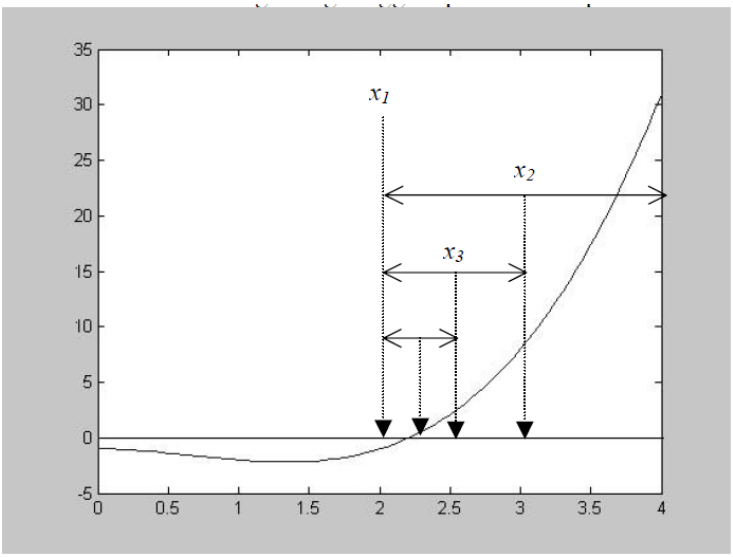

Figure 7.5: Ilustrasi metode biseksi.

1. Definisikan fungsi $f(x)$
2. Tentukan rentang untuk x yang berupa batas bawah a dan batas atas b.
3. Tentukan nilai toleransi e dan iterasi maksimum N
4. Hitung $f(a)$ dan $f(b)$
5. Hitung:

$$x = \frac{a+b}{2} \tag{7.5}$$

6. Hitung $f(x)$
7. Bila $f(x).f(a) < 0$, maka $b = x$ dan $f(b) = f(x)$. Bila tidak, $a = x$ dan $f(a) = f(x)$
8. Bila $|b-a| < e$ atau iterasi maksimum maka proses dihentikan dan didapatkan akar=x, dan bila tidak ulangi langkah 6.
9. Jika sudah diperoleh nilai dibawah nilai toleransi, nilai akar selanjutnya dihitung berdasarkan Persamaan (7.5) dengan nilai a dan b merupakan nilai baru yang diperoleh dari proses iterasi.

Berdasarkan algoritma tersebut, kita dapat menyusun suatu fungsi pada `R` yang dapat digunakan untuk melakukan iterasi tersebut. Fungsi `root_bisection()` merupakan fungsi yang telah penulis susun untuk melakukan iterasi menggunakan metode biseksi. Berikut adalah sintaks dari fungsi tersebut:

```
root_bisection <- function(f, a, b, tol=1e-7, N=100){
  iter <- 0
  fa <- f(a)
  fb <- f(b)

  while(abs(b-a)>tol){
    iter <- iter+1
    if(iter>N){
      warning("iterations maximum exceeded")
      break
    }
    x <- (a+b)/2
    fx <- f(x)
    if(fa*fx>0){
      a <- x
      fa <- fx
    } else{
      b <- x
      fb <- fx
    }
  }

  # iterasi nilai x sebagai return value
  root <- (a+b)/2
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.2. Carilah akar persamaan $f(x) = xe^{-x}+1$ pada rentang $x = [-1, 0]$ dengan nilai toleransi sebesar 10^-7?

Jawab:

Langkah pertama dalam penghitungan adalah menghitung nilai x menggunakan Persamaan (7.5).

$$x = \frac{-1+0}{2} = -0,5$$

Hitung nilai $f(x)$ dan $f(a)$.

$$f(x) = -0,5.e^{0,5} + 1 = 0,175639$$

$$f(a) = -1.e^{1} + 1 = -1,71828$$

Berdasarkan hasil perhitungan diperoleh:

$$f(x).f(a) < 0$$

Sehingga $b = x$ dan $f(b) = f(x)$. Iterasi dilakukan kembali dengan menggunakan nilai b tersebut.

Untuk mempersingkat waktu iterasi kita akan menggunakan fungsi `root_bisection()` pada `R`. Berikut adalah sintaks yang digunakan:

```
root_bisection(function(x){x*exp(-x)+1},
               a=-1, b=0)
```

```
## $`function`
## function (x)
## {
##     x * exp(-x) + 1
## }
## <bytecode: 0x000000000885f020>
##
## $root
## [1] -0.5671433
##
## $iter
## [1] 24
```

Berdasarkan hasil iterasi diperoleh akar persamaan $x = -2.980232e-08$ dan iterasi yang diperlukan untuk memperolehnya sebanyak 24 iterasi.

7.1.3 Metode Regula Falsi

Metode regula falsi merupakan metode yang menyerupai metode biseksi, dimana iterasi dilakukan dengan terus melakukan pembaharuan rentang untuk memperoleh akar persamaan. Hal yang membedakan metode ini dengan metode biseksi adalah pencarian akar didasarkan pada slope (kemiringan) dan selisih tinggi dari kedua titik rentang. Titik pendekatan pada metode regula-falsi disajikan pada Persamaan (7.6).

$$x = \frac{f(b).a - f(a).b}{f(b) - f(a)} \tag{7.6}$$

Ilustrasi dari metode regula falsi disajikan pada Gambar 7.6.

Algoritma Metode Regula Falsi

1. Definisikan fungsi $f(x)$

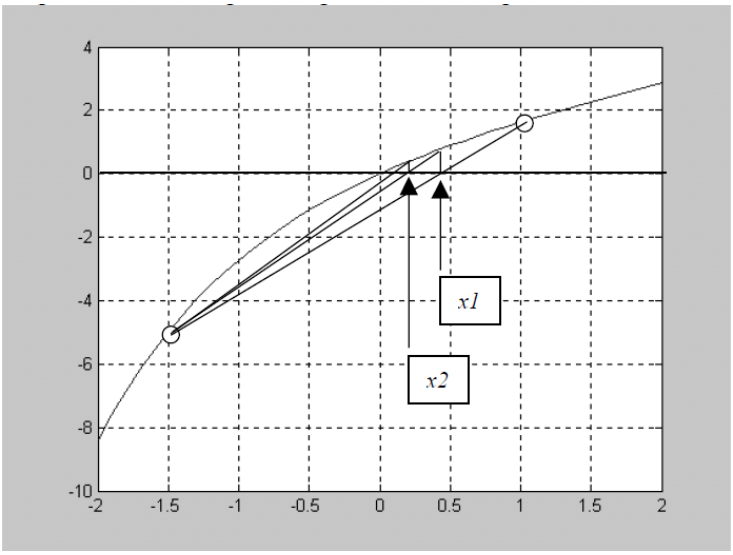

Figure 7.6: Ilustrasi metode regula falsi.

2. Tentukan rentang untuk x yang berupa batas bawah a dan batas atas b.
3. Tentukan nilai toleransi e dan iterasi maksimum N
4. Hitung $f(a)$ dan $f(b)$
5. Untuk iterasi $i = 1$ s/d N

- Hitung nilai x berdasarkan Persamaan (7.6)
- Hitung $f(x)$
- Hitung $error = |f(x)|$
- Jika $f(x).f(a) < 0$, maka $b = x$ dan $f(b) = f(x)$. Jika tidak, $a = x$ dan $f(a) = f(x)$.

6. Akar persamaan adalah x

Fungsi `root_rf()` didasarkan pada langkah-langkah di atas. Sintaks fungsi tersebut adalah sebagai berikut:

```
root_rf <- function(f, a, b, tol=1e-7, N=100){
  iter <- 1
  fa <- f(a)
  fb <- f(b)
  x <- ((fb*a)-(fa*b))/(fb-fa)
  fx <- f(x)

  while(abs(fx)>tol){
```

```
    iter <- iter+1
    if(iter>N){
      warning("iterations maximum exceeded")
      break
    }
    if(fa*fx>0){
      a <- x
      fa <- fx
    } else{
      b <- x
      fb <- fx
    }
    x <- (fb*a-fa*b)/(fb-fa)
    fx <- f(x)
  }

  # iterasi nilai x sebagai return value
  root <- x
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.3. Selesaikan persamaan non-linier pada Contoh 7.2 menggunakan metode regula falsi pada rentang $x = [-1, 0]$ dengan nilai toleransi sebesar 10^-7?

Jawab:

Langkah pertama penyelesaian dilakukan dengan mencari nilai $f(a)$ dan $f(b)$.

$$f(a) = -1.e^1 + 1 = -1,71828$$
$$f(b) = 0.e^0 + 1 = 1$$

Hitung nilai x dan $f(x)$.

$$x = \frac{(1.-1) - (-1,71828.0)}{1 + 1,71828} = -0.36788$$

$$f(x) = -0.36788.e^{0.36788} + 1 = 0.468536$$

Berdasarkan hasil perhitungan diperoleh:

$$f(x).f(a) < 0$$

Sehingga $b = x$ dan $f(b) = f(x)$. Iterasi dilakukan kembali dengan menggunakan nilai b tersebut.

Untuk mempercepat proses iterasi, kita dapat pula menggunakan fungsi `root_rf()` pada `R`. Berikut adalah sintaks yang digunakan:

```
root_rf(function(x){x*exp(-x)+1},
                a=-1, b=0)
```

```
## $`function`
## function (x)
## {
##     x * exp(-x) + 1
## }
## <bytecode: 0x0000000006207a68>
##
## $root
## [1] -0.5671433
##
## $iter
## [1] 15
```

Berdasarkan hasil perhitungan diperoleh nilai $x = -0,5671433$ dan jumlah iterasi yang diperlukan adalah 15. Jumlah ini lebih sedikit dari jumlah iterasi yang diperlukan pada metode iterasi biseksi yang juga menunjukkan metode ini lebih cepat memperoleh persamaan dibandingkan metode biseksi.

7.2 Metode Terbuka

Metode terbuka merupakan metode yang menggunakan satu atau dua tebakan awal yang tidak memerlukan rentang sejumlah nilai. Metode terbuka terdiri dari beberapa jenis yaitu metode iterasi titik tetap, metode Newton-Raphson, dan metode Secant.

7.2.1 Metode Iterasi Titik Tetap

Metode iterasi titik tetap merupakan metode penyelesaian persamaan non-linier dengan cara menyelesaikan setiap variabel x yang ada dalam suatu persamaan dengan sebagian yang lain sehingga diperoleh $x = g(x)$ untuk masing-masing variabel x. Sebagai contoh, untuk menyelesaikan persamaan $x + e^x = 0$, maka persamaan tersebut perlu diubah menjadi $x = e^x$ atau $g(x) = e^x$. Secara grafis metode ini diilustrasikan seperti Gambar 7.7.

Algoritma Metode Iterasi Titik Tetap

1. Definisikan $f(x)$ dan $g(x)$

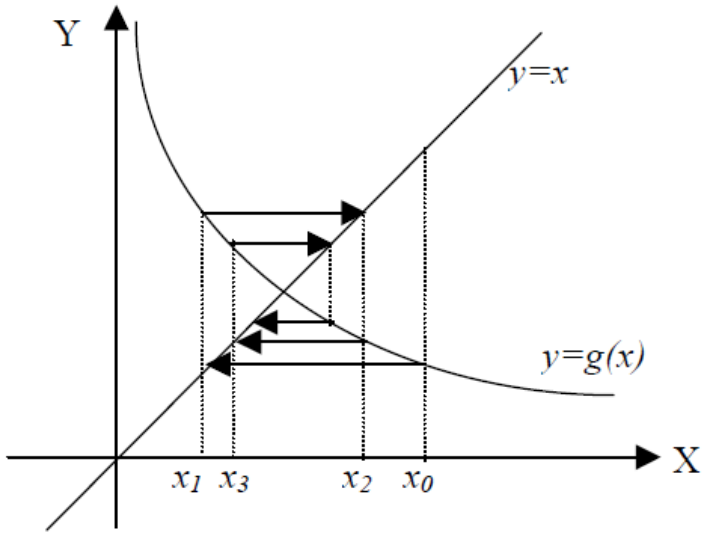

Figure 7.7: Ilustrasi metode iterasi titik tetap.

2. Tentukan nilai toleransi e dan iterasi masimum (N)
3. Tentukan tebakan awal x_0
4. Untuk iterasi $i = 1$ s/d N atau $f\left(x_i terasi\right) \geq e \rightarrow x_i = g\left(x_{i-1}\right)$, Hitung $f\left(x_i\right)$
5. Akar persamaan adalah x terakhir yang diperoleh

FUngsi `root_fpi()` dapat digunakan untuk melakukan iterasi dengan argumen fungsi berupa persamaan non-linier, nilai tebakan awal, nilai toleransi, dan jumlah iterasi maksimum. Berikut adalah sintaks fungsi tersebut:

```
root_fpi <- function(f, x0, tol=1e-7, N=100){
  iter <- 1
  xold <- x0
  xnew <- f(xold)

  while(abs(xnew-xold)>tol){
    iter <- iter+1
    if(iter>N){
      stop("No solutions found")
    }
    xold <- xnew
    xnew <- f(xold)
  }
```

```
  root <- xnew
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.4. Selesaikan persamaan non-linier pada Contoh 7.2 menggunakan metode iterasi titik tetap?

Jawab:

Untuk menyelesaikan persamaan non-linier tersebut kita perlu mentransformasi persamaan non-linier tersebut terlebih dahulu.

$$xe^{-x} + 1 = 0 \ \rightarrow x = -\frac{1}{e^{-x}}$$

Untuk tebakan awal digunakan nilai $x = -1$

$$x_1 = -\frac{1}{e^1} = -2,718282$$

Nilai x tersebut selanjutnya dijadikan nilai input pada iterasi selanjutnya:

$$x_2 = -\frac{1}{e^{2,718282}} = -0,06598802$$

iterasi terus dilakukan sampai diperoleh $|x_{i+1} - x_i| \leq e$.

Untuk mempercepat proses iterasi kita dapat menggunakan bantuan fungsi `root_fpi()`. Berikut adalah sintaks yang digunakan:

```
root_fpi(function(x){-1/exp(-x)}, x0=-1)
```

```
## $`function`
## function (x)
## {
##     -1/exp(-x)
## }
## <bytecode: 0x00000000087d6a18>
##
## $root
## [1] -0.5671433
##
## $iter
## [1] 29
```

Berdasarkan hasil iterasi diperoleh nilai $x = -0,5671433$ dengan jumlah iterasi yang diperlukan sebanyak 29 kali. Jumlah iterasi akan bergantung dengan nilai tebakan awal yang kita berikan. Semakin dekat nilai tersebut dengan akar, semakin cepat nilai akar diperoleh.

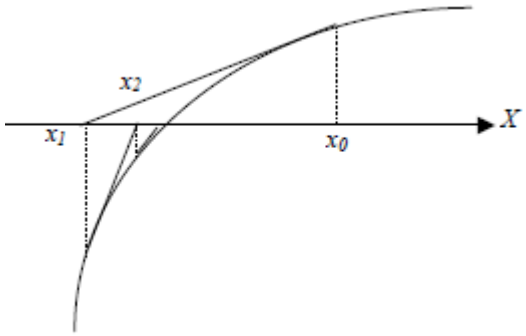

Figure 7.8: Ilustrasi metode Newton-Raphson.

7.2.2 Metode Newton-Raphson

Metode Newton-Raphson merupakan metode penyelesaian persamaan non-linier dengan menggunakan pendekatan satu titik awal dan mendekatinya dengan memperhatikan slope atau gradien. titik pendekatan dinyatakan pada Persamaan (7.7).

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{7.7}$$

Ilustrasi metode Newton-Raphson disajikan pada Gambar 7.8.

Algoritma Metode Newton-Raphson

1. Definisikan $f(x)$ dan $f'(x)$
2. Tentukan nilai toleransi e dan iterasi masimum (N)
3. Tentukan tebakan awal x_0
4. Hitung $f(x_0)$ dan $f'(x_0)$
5. Untuk iterasi $i = 1$ s/d N atau $|f(x)| \geq e$, hitung x menggunakan Persamaan (7.7)
6. Akar persamaan merupakan nilai x_i terakhir yang diperoleh.

Fungsi `root_newton()` merupakan fungsi yang dibuat menggunakan algoritma di atas. Fungsi tersebut dituliskan pada sintaks berikut:

```
root_newton <- function(f, fp, x0, tol=1e-7, N=100){
  iter <- 0
  xold<-x0
  xnew <- xold + 10*tol

  while(abs(xnew-xold)>tol){
```

```
    iter <- iter+1
    if(iter>N){
      stop("No solutions found")
    }
    xold<-xnew
    xnew <- xold - f(xold)/fp(xold)
  }

  root<-xnew
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.5. Selesaikan persamaan non-linier $x - e^{-x} = 0$ menggunakan metode Newton-Raphson?

Jawab:

Untuk dapat menggunakan metode Newton-Raphson, terlebih dahulu kita perlu memperoleh turunan pertama dari persamaan tersebut.

$$f(x) = x - e^{-x} \rightarrow f'(x) = 1 + e^{-x}$$

Tebakan awal yang digunakan adalah $x = 0$.

$$f(x_0) = 0 - e^{-0} = -1$$
$$f'(x_0) = 1 + e^{-0} = 2$$

Hitung nilai x baru:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0 - \frac{-1}{2} = 0,5$$

Untuk mempercepat proses iterasi, kita dapat menggunakan fungsi `root_newton()`. Berikut adalah sintaks yang digunakan:

```
root_newton(function(x){x-exp(-x)},
            function(x){1+exp(-x)},
              x0=0)
```

```
## $`function`
## function (x)
## {
##     x - exp(-x)
## }
```

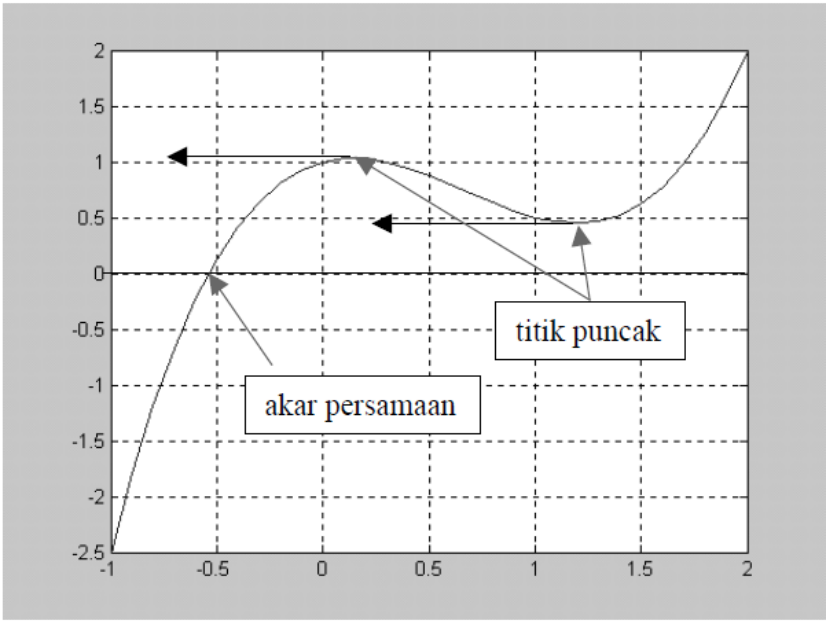

Figure 7.9: Ilustrasi titik pendekatan di titik puncak.

```
## <bytecode: 0x000000001cae7f10>
##
## $root
## [1] 0.5671433
##
## $iter
## [1] 5
```

Berdasarkan hasil iterasi diperoleh akar penyelesaian persamaan non-linier adalah $x = 0,5671433$ dengan jumlah iterasi yang diperlukan adalah 5 iterasi.

Dalam penerapannya metode Newton-Raphson dapat mengalami kendala. Kendala yang dihadapi adalah sebagai berikut:

1. titik pendekatan tidak dapat digunakan jika merupakan titik ekstrim atau titik puncak. Hal ini disebabkan pada titik ini nilai $f'(x) = 0$. Untuk memahaminya perhatikan ilustasi yang disajikan pada Gambar 7.9. Untuk menatasi kendala ini biasanya titik pendekatan akan digeser.

2. Sulit memperoleh penyelesaian ketika titik pendekatan berada diantara 2 titik stasioner. Untuk memahami kendala ini perhatikan Gambar 7.10. Untuk menghindarinya, penentuan titik pendekatan dapat menggunakan bantuan metode tabel.

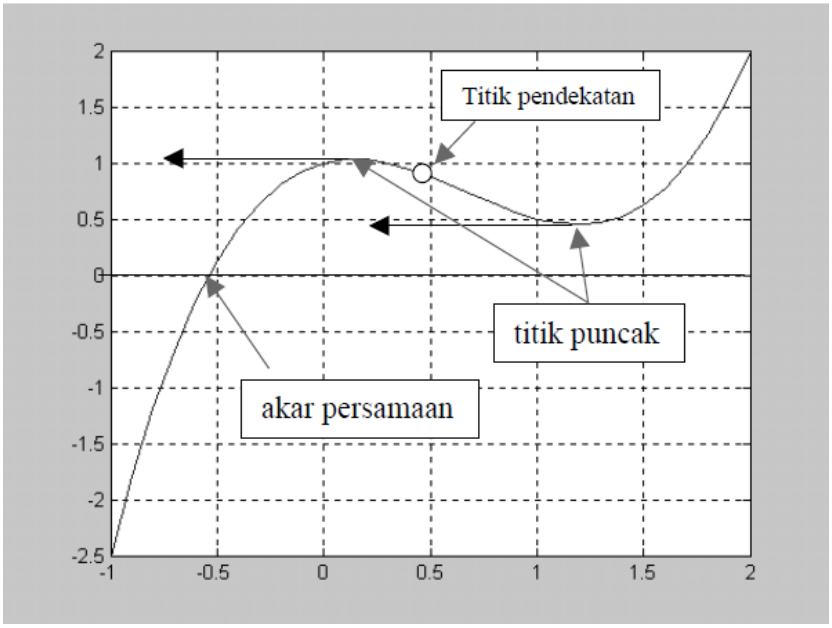

Figure 7.10: Ilustrasi titik pendekatan diantara 2 titik stasioner.

3. Turunan persamaan sering kali sulit untuk diperoleh (tidak dapat dikerjakan dengan metode analitik).

7.2.3 Metode Secant

Metode Secant merupakan perbaikan dari metode regula-falsi dan Newton Raphson, dimana kemiringan dua titik dinyatakan secara diskrit dengan mengambil bentuk garis lurus yang melalui satu titik. Persamaan yang dihasilkan disajikan pada Persamaan (7.8).

$$y - y_0 = m\,(x - x_0) \tag{7.8}$$

Nilai m merupakan transformasi persamaan tersebut.

$$m_n = \frac{f\,(x_n) - f\,(x_{n-1})}{x_n - x_{n-1}} \tag{7.9}$$

Bila $y = f\,(x)$ dan y_n dan x_n diketahui, maka titik ke $n+1$ adalah:

$$y_{n+1} - y_n = m_n\,(x_{n+1} - x_n) \tag{7.10}$$

Bila titik x_{n+1} dianggap akar persamaan maka nilai $y_{n+1} = 0$, sehingga diperoleh:

$$-y_n = m_n (x_{n+1} - x_n) \tag{7.11}$$

$$\frac{m_n x_n - y_n}{m_n} = x_{n+1} \tag{7.12}$$

atau

$$x_{n+1} = x_n - y_n \frac{1}{m_n} \tag{7.13}$$

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n+1}}{f(x_n) - f(x_{n+1})} \tag{7.14}$$

Berdasarkan Persamaan (7.14) diketahui bahwa untuk memperoleh akar persamaan diperlukan 2 buah titik pendekatan. Dalam buku ini akan digunakan titik pendekatan kedua merupakan titik pendekatan pertama ditambah sepuluh kali nilai toleransi.

$$x_1 = x_0 + 10 * tol \tag{7.15}$$

Algoritma Metode Secant

1. Definisikan $f(x)$ dan $f'(x)$
2. Tentukan nilai toleransi e dan iterasi masimum (N)
3. Tentukan tebakan awal x_0 dan x_1
4. Hitung $f(x_0)$ dan $f(x_1)$
5. Untuk iterasi $i = 1$ s/d N atau $|f(x)| \geq e$, hitung x menggunakan Persamaan (7.14)
6. Akar persamaan adalah nilai x yang terakhir.

Fungsi `root_secant()` merupakan fungsi yang penulis buat untuk melakukan iterasi menggunakan metode Secant. Berikut merupakan sintaks dari fungsi tersebut:

```
root_secant <- function(f, x, tol=1e-7, N=100){
  iter <- 0

  xold <- x
  fxold <- f(x)
  x <- xold+10*tol
```

```
  while(abs(x-xold)>tol){
    iter <- iter+1
    if(iter>N)
      stop("No solutions found")

    fx <- f(x)
    xnew <- x - fx*((x-xold)/(fx-fxold))
    xold <- x
    fxold <- fx
    x <- xnew
  }

  root<-xnew
  return(list(`function`=f, root=root, iter=iter))
}
```

Contoh 7.6. Selesaikan persamaan non-linier pada Contoh 7.5 menggunakan metode Secant?

Jawab:

Untuk menyelesaikan persamaan tersebut digunakan nilai pendekatan awal $x_0 = 0$ dan $x_1 = 0 + 10 * 10^{-7} = 10^{-6}$.

$$f(x_0) = 0 - e^{-0} = -1$$

$$f(x_1) = 10^{-6} - e^{-10^{-6}} = -0,999998$$

Hitung nilai x_2 dan $f(x_2)$.

$$x_2 = 0 + 0,999998\frac{10^{-6} - 0}{-0,999998 + 1} = 0,499999$$

Untuk mempercepat proses iterasi kita dapat menggunakan fungsi `root_secant()` pada `R`. Berikut sintaks yang digunakan:

```
root_secant(function(x){x-exp(-x)}, x=0)
```

```
## $`function`
## function (x)
## {
##     x - exp(-x)
## }
```

```
## <bytecode: 0x0000000008933ae0>
##
## $root
## [1] 0.5671433
##
## $iter
## [1] 6
```

Berdasarkan hasil iterasi diperoleh nilai akar penyelesaian adalah $x = 0,5671433$ dengan iterasi dilakukan sebanyak 6 kali.

Secara umum metode Secant menawarkan sejumlah keuntungan dibanding metode lainnya. Pertama, seperti metode Newton-Raphson dan tidak seperti metode tertutup lainnya, metode ini tidak memerlukan rentang pencarian akar penyelesaian. Kedua, tidak seperti metode Newton-Raphson, metode ini tidak memerlukan pencarian turunan pertama persamaan non-linier secara analitik, dimana tidak dapat dilakukan otomasi pada setiap kasus.

Adapun kerugian dari metode ini adalah berpotensi menghasilkan hasil yang tidak konvergen sama seperti metode terbuka lainnya. Selain itu, kecepatan konvergensinya lebih lambat dibanding metode Newton-Raphson.

7.3 Penyelesaian Persamaan Non-Linier Menggunakan Fungsi `uniroot` dan `uniroot.all`

Paket `base` pada `R` menyediakan fungsi `uniroot()` untuk mencari akar persamaan suatu fungsi pada rentang spesifik. Fungsi ini menggunakan metode Brent yaitu kombinasi antara *root bracketing*, biseksi, dan interpolasi invers kuadrat. Format fungsi tersebut secara sederhana adalah sebagai berikut:

```
uniroot(f, interval, tol=.Machine$double.eps^0.25,
        maxiter=1000)
```

Catatan:

- **f**: persamaan non-linier
- **interval**: vektor interval batas bawah dan atas
- **tol**: nilai toleransi
- **maxiter**: iterasi maksimum

Berikut adalah contoh penerapan fungsi `uniroot()`:

```
uniroot(function(x){x*exp(-x)+1},
        interval=c(-1,0), tol=1e-7)
```

```
## $root
## [1] -0.5671433
##
## $f.root
## [1] 1.532974e-08
##
## $iter
## [1] 7
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 5e-08
```

Berdasarkan hasil iterasi diperoleh akar persamaan tersebut adalah $-0,5671433$ dengan jumlah iterasi sebanyak 7 iterasi dan tingkat presisi sebesar $5e-08$.

Fungsi lain yang dapat digunakan untuk mencari akar persamaan adalah `uniroot.all()` dari paket `rootSolve`. Fungsi ini mengatasi kelemahan dari `uniroot()`, dimana `uniroot()` tidak bekerja jika fungsi hanya menyentuh dan tidak melewati sumbu nol $y = 0$. Untuk memahaminya perhatikan contoh berikut:

```
uniroot(function(x){sin(x)+1}, c(-pi,0))
```

Bandingkan dengan sintaks berikut:

```
uniroot(function(x){sin(x)+1}, c(-pi,-pi/2))
```

```
## $root
## [1] -1.570796
##
## $f.root
## [1] 0
##
## $iter
## [1] 0
##
## $init.it
## [1] NA
```

```
##
## $estim.prec
## [1] 0
```

Untuk menggunakan fungsi `uniroot.all()`, jalankan sintaks berikut:

```
library(rootSolve)
```

Jalankan kembali fungsi dan rentang di mana `uniroot()` tidak dapat bekerja:

```
uniroot.all(function(x){sin(x)+1}, c(-pi,0))
```

```
## [1] -1.570796
```

7.4 Akar Persamaan Polinomial Menggunakan Fungsi `polyroot`

Fungsi `polyroot()` pada paket `base` dapat digunakan untuk memperoleh akar dari suatu polinomial. Algortima yang digunakan dalam fungsi tersebut adalah algoritma Jenkins dan Traub.

Untuk dapat menggunakannya kita hanya perlu memasukkan vektor koefisien dari polinomial. Pengisian elemen dalam vektor dimulai dari variabel dengan pangkat tertinggi menuju variabel dengan pangkat terendah. Berikut adalah contoh bagaimana fungsi `polyroot()` digunakan untuk mencari akar polinomial $f(x) = x^2 + 1$:

```
polyroot(c(1,0,1))
```

```
## [1] 0+1i 0-1i
```

Contoh lainnya adalah mencari akar polinomial $f(x) = 4x^2 + 5x + 6$:

```
polyroot(c(4,5,6))
```

```
## [1] -0.4166667+0.7021791i -0.4166667-0.7021791i
```

Pembaca dapat mencoba membuktikan hasil yang diperoleh tersebut menggunakan metode analitik.

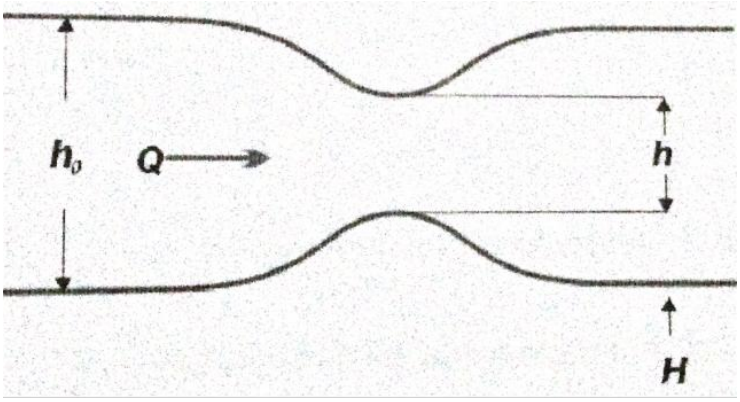

Figure 7.11: Aliran fluida pada sebuah pipa.

7.5 Studi Kasus

Penerapan penyelesaian sistem persamaan non-linier banyak dijumpai dalam berbagai kasus di bidang lingkungan. Pada bagian ini penulis tidak akan menjelaskan seluruhnya. Penulis hanya akan menjelaskan penerapannya pada sebuah persamaan yaitu Hukum Bernoulli.

7.5.1 Persamaan Van Der Walls

7.5.2 Hukum Bernoulli

Misalkan terdapat sebuah saluran dengan penampang sesuai dengan Gambar 7.11.

Berdasarkan hukum Bernoulli, maka diperoleh persamaan berikut:

$$\frac{Q^2}{2gb^2h_0^2} + h_0 = \frac{Q^2}{2gb^2h^2} + h + H \tag{7.16}$$

Persamaan tersebut dapat dilakukan transformasi menjadi persamaan berikut:

$$f(h) = h^3 + \left(H - \frac{Q^2}{2gb^2h_0^2} - h_0\right)h^2 + \frac{Q^2}{2gb^2} = 0 \tag{7.17}$$

Data-data terkait saluran tersebut adalah sebagai berikut:

- $Q = 1,2\ \frac{m^3}{\det}$ = volume aliran fluida tiap satuan waktu
- $g = 9,81\ \frac{m}{s^2}$ =percepatan gravitasi

- $b = 1,8\ m$ =lebar pipa
- $h_0 = 0,6\ m$ =ketinggian air maksimum
- $H = 0,075\ m$ =tinggi pelebaran pipa
- h = ketinggian air

Kita dapat menggunakan pendekatan numerik untuk menentukan h. Pada studi kasus ini tidak dijelaskan lokasi dimana akar penyelesaian berada, sehingga metode terbuka seperti Secant cukup sesuai untuk menyelesaikannya:

Berikut adalah persamaan yang baru setelah seluruh data dimasukkan kedalam tiap variabelnya:

$$f(h) = h^3 + \left(0,075 - \frac{1,2^2}{2 \times 9,81 \times 1,8^2 \times 0,6^2} - 0,6\right) h^2 + \frac{1,2^2}{2 \times 9,81 \times 1,8^2} = 0$$

Untuk penyelesaiannya penulis akan memberikan tebakan awal nilai $h = h_0 = 0,6$. Berikut adalah sintaks penyelesaian menggunakan metode secant:

```
f <- function(h){
  (h^3) + ((0.075-((1.2^2)/(2*9.81*(1.8^2)*(0.6^2))))*h^2)+ (1.2^2/(2*9.81*(1.8^2)))
}
root_secant(f, 0.6)
```

```
## $`function`
## function (h)
## {
##     (h^3) + ((0.075 - ((1.2^2)/(2 * 9.81 * (1.8^2) * (0.6^2)))) *
##         h^2) + (1.2^2/(2 * 9.81 * (1.8^2)))
## }
## <bytecode: 0x0000000008013738>
##
## $root
## [1] -0.2870309
##
## $iter
## [1] 26
```

Berdasarkan hasil perhitungan diperoleh nilai $h = -0,2870309$ atau ketinggian air sekitar $0,3\ m$ dengan jumlah iterasi sebanyak 26 kali.

Pembaca dapat mencoba menggunakan metode lain seperti metode tertutup. Untuk dapat melakukannya, pembaca perlu memperoleh rentang lokasi akar persamaan tersebut berada menggunakan metode tabel.

7.6 Referensi

1. Atmika, I.K.A. 2016. **Diktat Mata Kuliah: Metode Numerik**. Jurusan Teknik Mesin Universitas Udayana.
2. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press
3. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
4. Jones, O. Maillardet, R. Robinson, A. 2014. **Introduction to Scientific Programming and Simulation Using R**. CRC Press
5. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
6. Sanjaya, M. 2015. **Metode Numerik Berbasis Phython**. Penerbit Gava Media: Yogyakarta.
7. Sudiadi dan Teguh R. 2015. **Metode Numerik**. STMIK

7.7 Latihan

1. Temukan akar persamaan dari persamaan non-linier $f(x) = x^3 - 2x + 2$ menggunakan metode terbuka dengan $x_0 = 0$ dan $x_0 = \frac{1}{2}$!
2. Apakah kelebihan dari metode tertutup (contoh: metode biseksi) dibanding metode terbuka (contoh: Newton-Raphson)? (**catatan**: pembaca dapat pula mencari dari referensi lainnya)
3. Temukan akar persamaan dari persamaan $f(x) = \frac{sin(x)}{x}$ dengan rentang pencarian $x = 0,5$ dan $x = 1$!
4. Pada kondisi apakah metode Secant lebih dipilih dibanding metode Newton-Raphson?
5. Modifikasilah fungsi `root_bisection()` dan `root_rf()` sehingga kita tidak perlu memasukkan argumen `a` dan `b` dan hanya perlu memasukkan satu vektor `interval` kedalam fungsi tersebut! (**contoh**: `interval=c(a,b)`)