Metode Numerik Menggunakan R Untuk Teknik Lingkungan

Mohammad Rosidi

2019-12-04

# Kata Pengantar

# 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.

*Library* 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

## 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.



Gambar 1: Logo R.

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

## 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

1. **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.
2. **interpreted language atau script**. Bahasa R memungkinkan pengguna untuk melakukan kerja pada R tanpa perlu kompilasi kode program menjadi bahasa mesin.
3. Mendukung proses **loop**, **decision making**, dan menyediakan berbagai jenis **operstor** (aritmatika, logika, dll).
4. **Mendukung export dan import berbagai format file**, seperti:TXT, CSV, XLS, dll.
5. **Mudah ditingkatkan melalui penambahan fungsi atau *library***. Penambahan *library* dapat dilakukan secara online melalui [CRAN](https://cran.r-project.org/) atau melalui sumber seperti [github](https://github.com/).
6. **Menyedikan berbagai fungsi untuk keperluan visualisasi data**. Visualisasi data pada R dapat menggunakan *library* bawaan atau *library* lain seperti ggplo2,ggvis, dll.

## 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 (*library* 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 *library* yang memungkinkan kita berinteraksi dengan R menggunakan *GUI* (*Graphical User Interface*) sederhana menggunakan *library* 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 *library* 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 *library* hanya masalah waktu saja.

## 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.

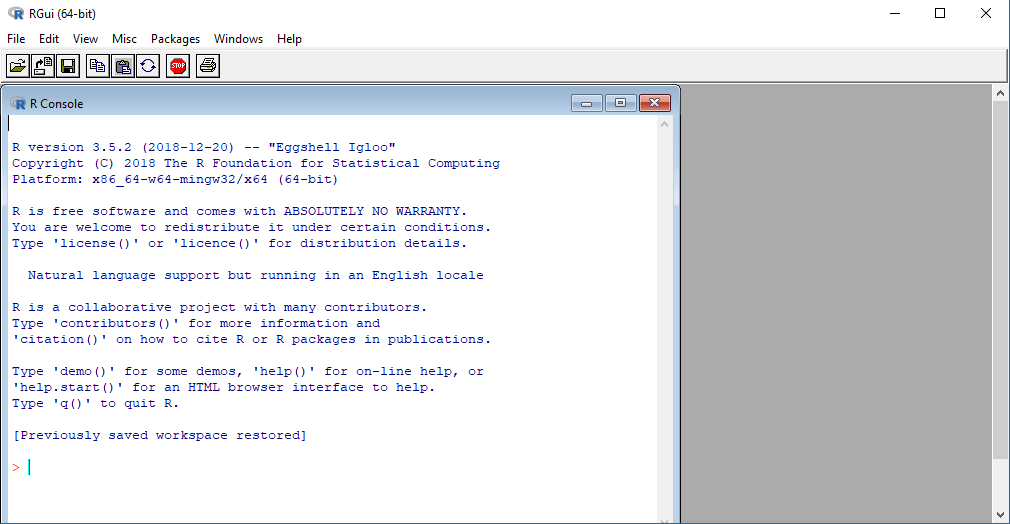
RStudio merupakan salah satu bentuk *frontend* R yang cukup populer dan nyaman digunakan. Selain nyaman digunakan, RStudio memungkinkan kita melakukan penulisan laporan menggunakan Rmarkdown atau RNotebook serta membuat berbagai bentuk project seperti shyni, dll. Pada R studio juga memungkinkan kita 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.

## Menginstall R dan RStudio

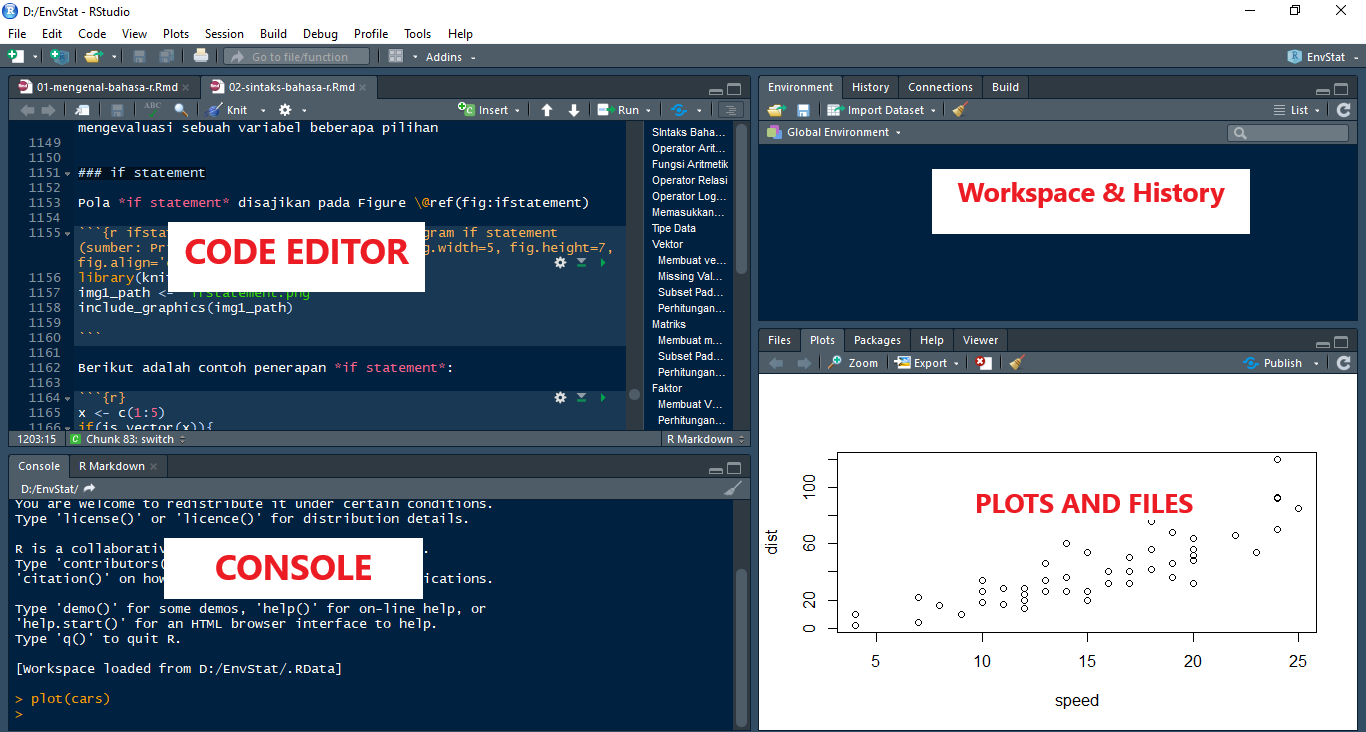
Pada tutorial ini hanya akan dijelaskan bagaimana menginstal R dan RStudio pada sistem operasi windows. Sebelum memulai menginstal sebaiknya pembaca mengunduh terlebih dahulu *installer* [R](https://cran.r-project.org/bin/windows/base/) dan [RStudio](https://www.rstudio.com/products/rstudio/download/).

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 2 dan Gambar 3.



Gambar 2: Jendela R.



Gambar 3: Jendela RStudio.

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

## 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.

### 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.

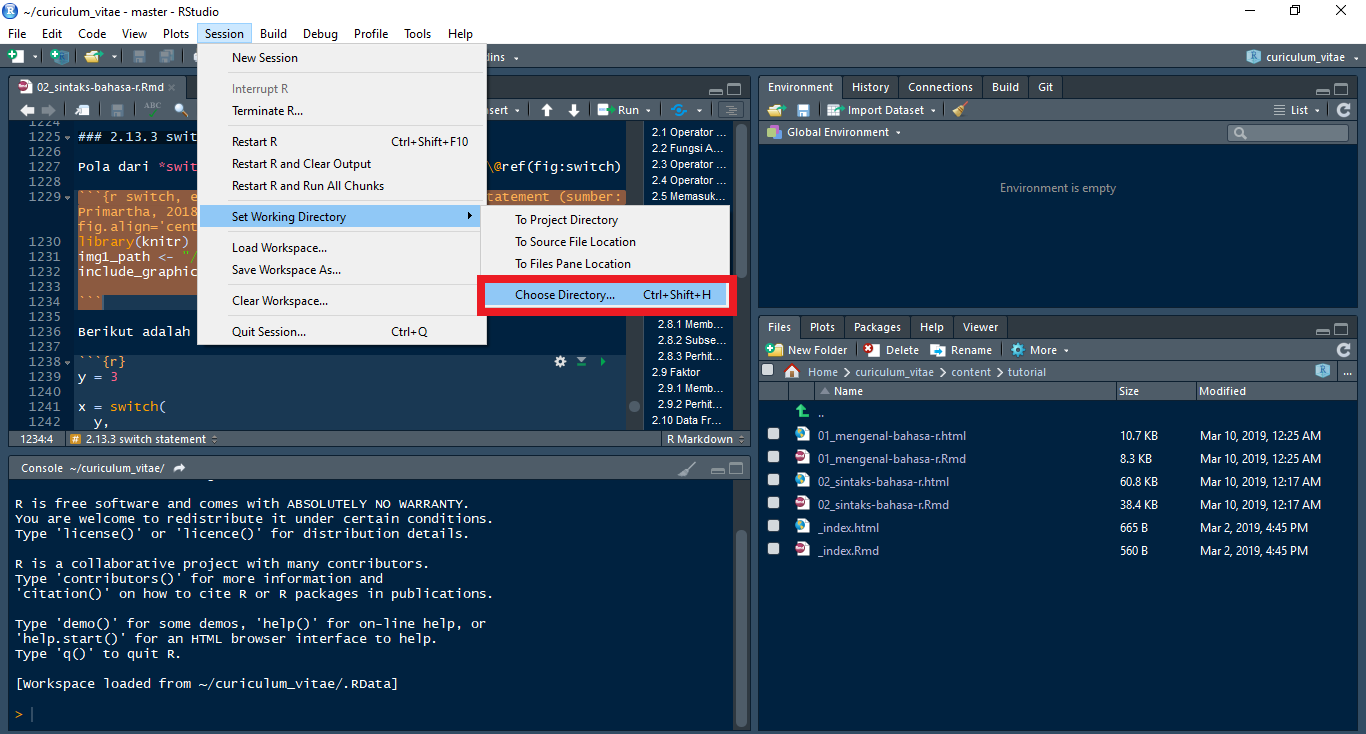
Berikut adalah cara mengubah *working directory* pada R.

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 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()



Gambar 4: Mengubah working directory.

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

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

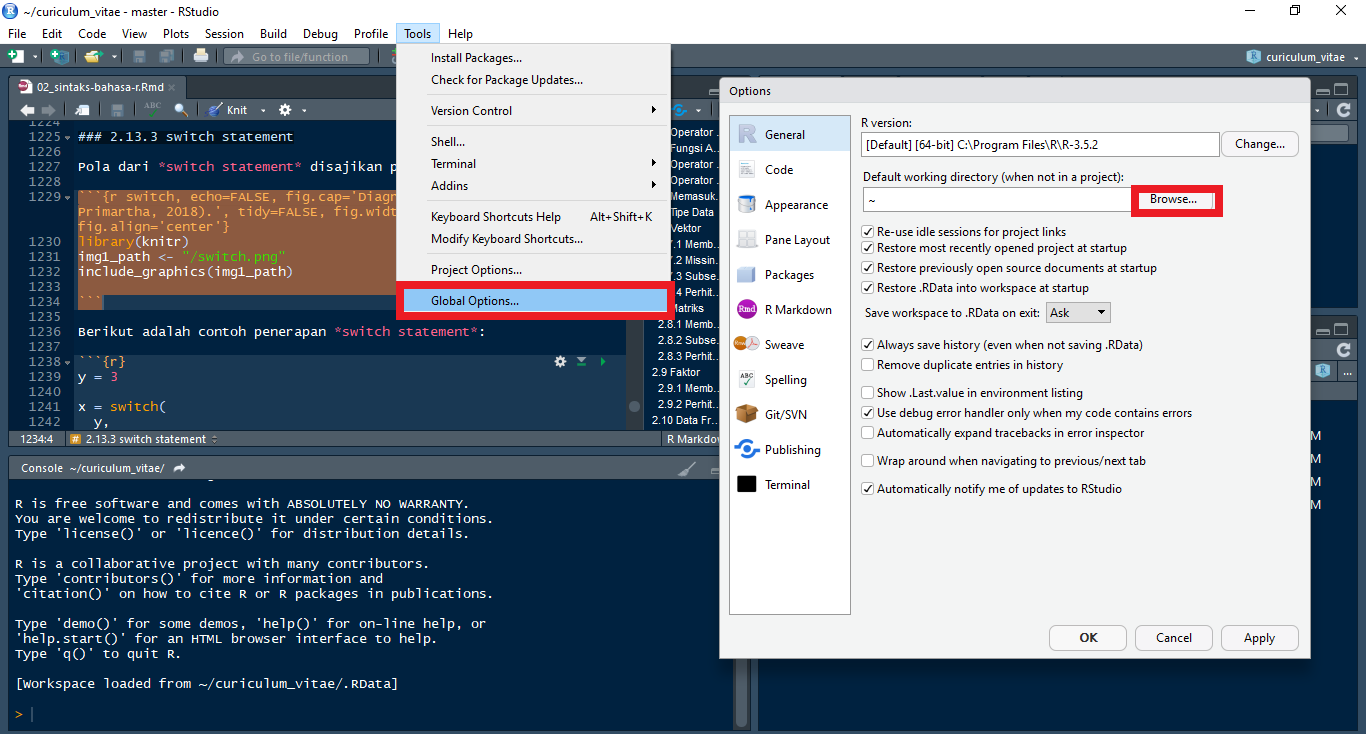
**Peringatan !!!**

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

### 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 5



Gambar 5: Merubah working directory melalui Global options.

## Memasang dan Mengaktifkan *Library* R

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

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

install.packages("tidyverse")

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

library(tidyverse)

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

readr::read\_csv("contoh.csv")

## 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.

### 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)

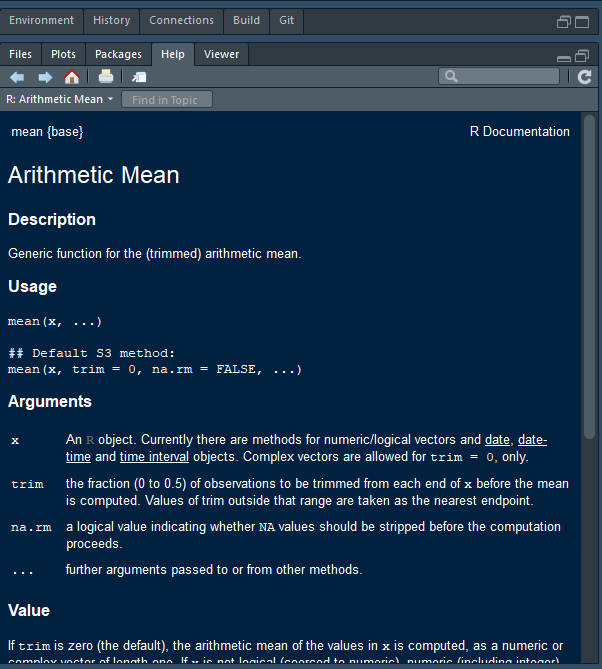
atau dapat juga menggunakan tanda tanya (?) pada awal nama\_perintah seperti berikut:

?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 6.



Gambar 6: Jendela help dokumentasi fungsi mean().

Keterangan pada jendela pada Gambar 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 *library* (*library*) *base* (*library* 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

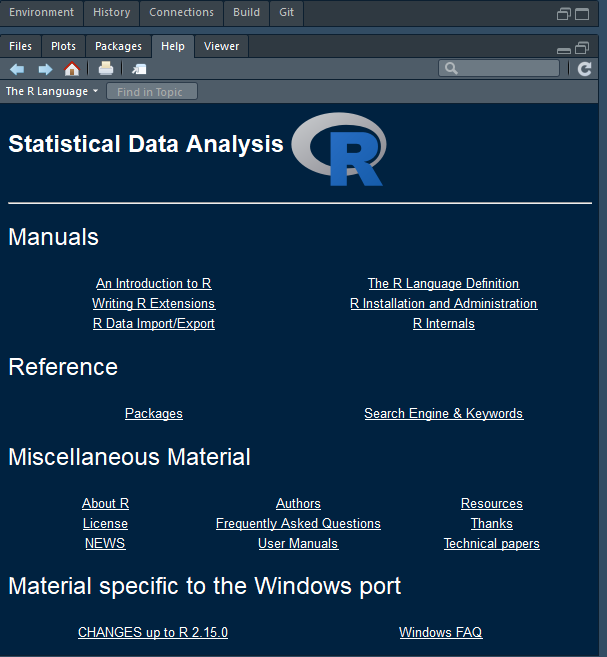
Pembaca dapat mencoba melakukanya sendiri dengan mengganti nilai yang telah ada serta mencoba contoh kode yang lain.

### 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 7.



Gambar 7: Jendela general help dokumentasi fungsi mean().

### 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")

## [1] "elNamed" "elNamed<-" "geo\_median"   
## [4] "median" "median.default" "median.zoo"   
## [7] "medpolish" "rollmedian" "rollmedian.default"  
## [10] "rollmedianr" "runmed"

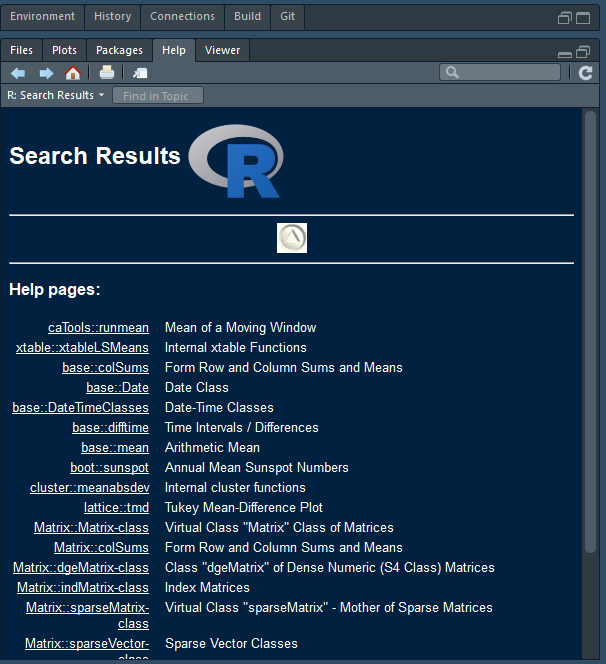
*List* yang dihasilkan berupa fungsi-fungsi yang memiliki elemen kata “med”. Berdasarkan pencaria tersebut penulis dapat mencoba menggunakan fungsi “median” untuk menghitung median.

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 8.



Gambar 8: Jendela help search dokumentasi fungsi mean().

## Referensi

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

# 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)

## 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 1:

(#tab:oparitmatika) 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

## 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

1. 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 *library* 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

1. 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 *library* pracma. Fungsi-fungsi yang tersedia antara lain:

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

1. 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

## Operator Relasi

Operator relasi digunakan untuk membandingkan satu objek dengan objek lainnya. Operator yang disediakan R disajikan pada Tabel 2.

(#tab:oprelasi) 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

## 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 3.

(#tab:oplogika) 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.

## 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.

## Tipe dan Struktur Data

Data pada R dapat dikelompokan berdasarkan beberapa tipe. Tipe data pada R disajikan pada Tabel 4.

(#tab:tipedata) 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 5.

(#tab:strukturdata) 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.

## 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.

### 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().

# 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().

length(Jumlah)

## [1] 4

### 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*.

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.

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().

### 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

### 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

### 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.

#### *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

### 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

### 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:

(#tab:randomsequence) 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* |
| Exponensial | 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] 0.474 1.474 2.474 3.474 4.474 5.474

## 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.

### 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

### 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

### 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.

## 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.

# 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.

## 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")

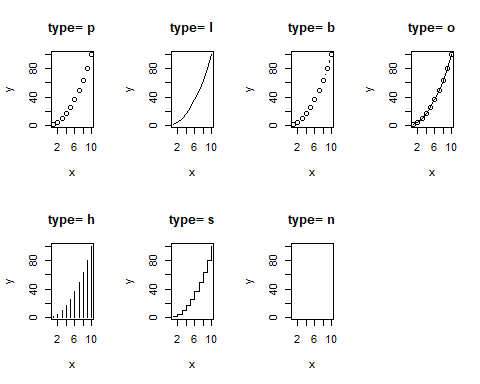
**Catatan:**

* **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 9:

# 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))  
}

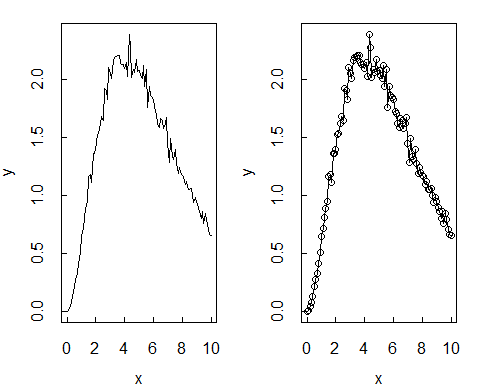


Gambar 9: Plot berbagai jenis setting type

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



Gambar 10: Plot fungsi matematika

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:

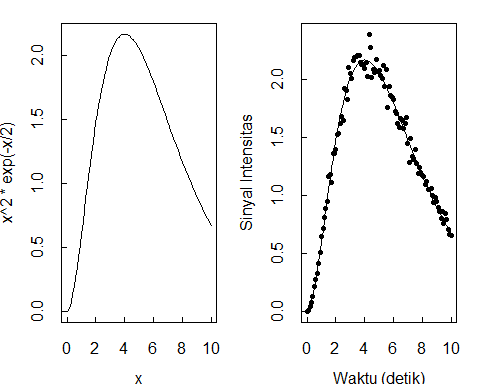
curve(expr, from = NULL, to = NULL, add = FALSE)

**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),  
 # 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)



Gambar 11: Visualisasi menggunakan fungsi curve (sebelah kiri) dan visualisasi menggunakan fungsi plot dan curve (sebelah kanan)

## Visualisasi Lainnya

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

### Bar Plot

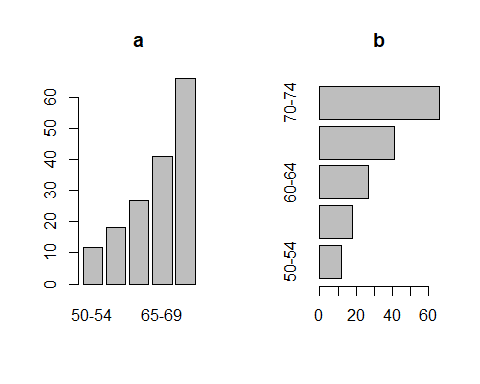
Barplot pada R dapat dibuat menggunakan fungsi barplot(). Untuk lebih memahaminya berikut disajikan contoh barplot menggunakan dataset VADeaths. Untuk memuatnya jalankan sintaks berikut:

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 12:

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

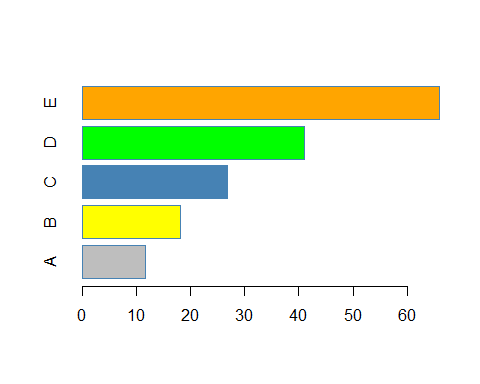


Gambar 12: a. bar plot vertikal; b. bar plot horizontal

par(mfrow=c(1,1))

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 13:

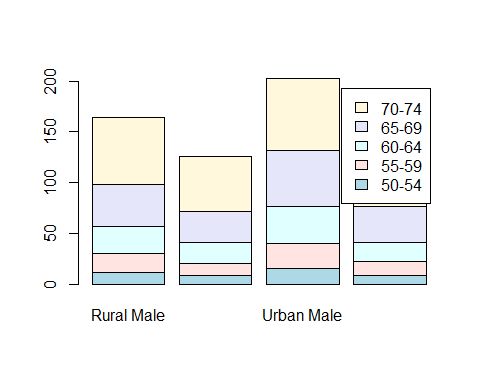
barplot(VADeaths[, "Rural Male"],  
 # ubah warna ouline menjadi steelblue  
 border="steelblue",  
 # ubah wana box  
 col= c("grey", "yellow", "steelblue", "green", "orange"),  
 # ubah nama grup dari A sampai E  
 names.arg = LETTERS[1:5],  
 # ubah orientasi menajadi horizontal  
 horiz=TRUE)



Gambar 13: Kustomisasi bar plot

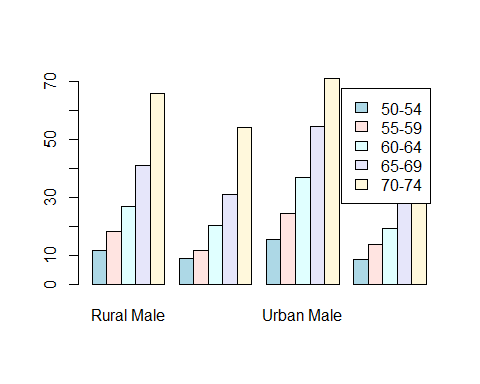
Untuk bar plot dengan *multiple group*, tersedia dua pengaturan posisi yaitu *stacked bar plot*(menunjukkan proporsi penyusun pada masing-masing grup) dan *grouped bar plot*(melihat perbedaan individual pada masing-masing grup). Pada Gambar 14 dan Gambar 15 , disajikan kedua jenis bar plot tersebut.

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



Gambar 14: Stacked bar plot

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



Gambar 15: Grouped bar plot

### Histogram dan Density Plot

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

hist(x, breaks="Sturges")

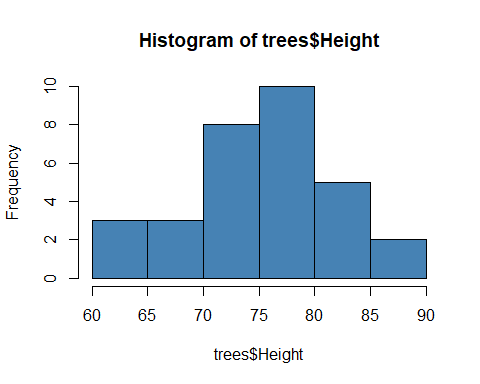
**Catatan:**

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

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

hist(trees$Height)

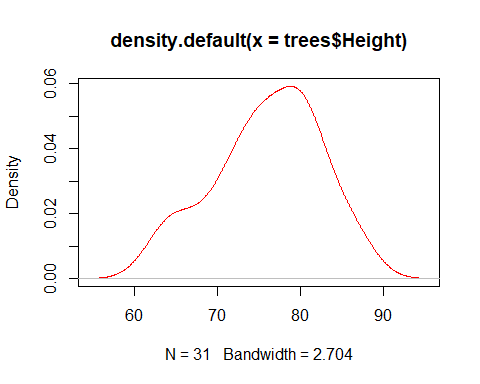
Output yang dihasilkan disajikan pada Gambar 16:



Gambar 16: 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 17:

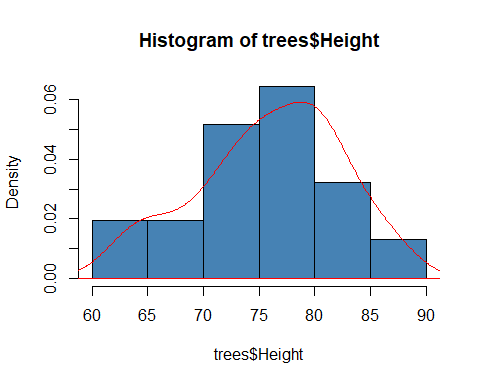
# menghitung kernel density  
dens <- density(trees$Height)  
  
# plot densitas dengan outline merah  
plot(dens,col="red")



Gambar 17: Density plot

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 18:

# menghitung kernel density  
dens <- density(trees$Height)  
  
# histogram  
hist(trees$Height, freq=FALSE, col="steelblue")  
  
# tambahkan density plot  
polygon(dens, border="red")

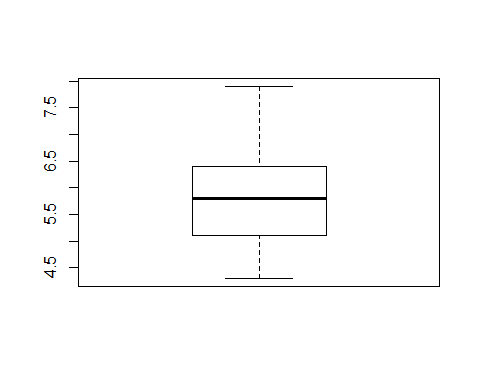


Gambar 18: Density plot dan histogram

### 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 19:

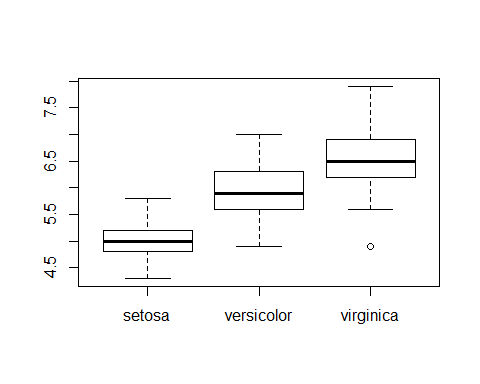
boxplot(iris$Sepal.Length)



Gambar 19: Boxplot variabel 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 20:

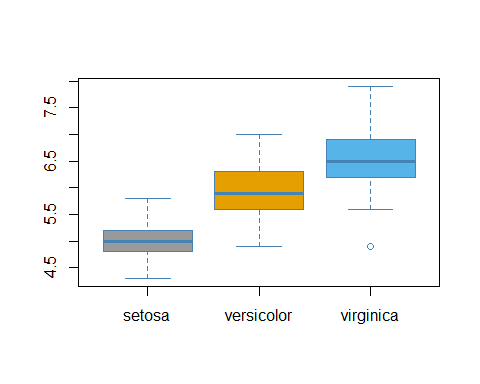
boxplot(iris$Sepal.Length~iris$Species)



Gambar 20: Boxplot berdasarkan variabel 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 21:

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



Gambar 21: Boxplot dengan warna berdasarkan spesies

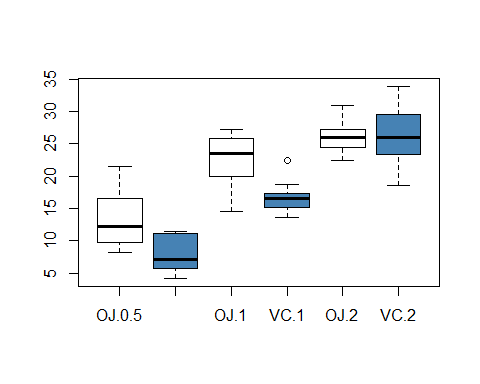
Kita juga dapat membuat boxplot pada *multiple group*. Data yang digunakan untuk contoh tersebut adalah dataset ToothGrowth. Berikut adalah sintaks untuk memuat dataset tersebut:

# 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 22:

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



Gambar 22: Boxplot multiple group

## Kustomisasi Parameter Grafik

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

1. menambahkan judul, legend, teks, axis, dan garis.
2. mengubah skala axis, simbol plot, jenis garis, dan warna.

### Menambahkan Judul

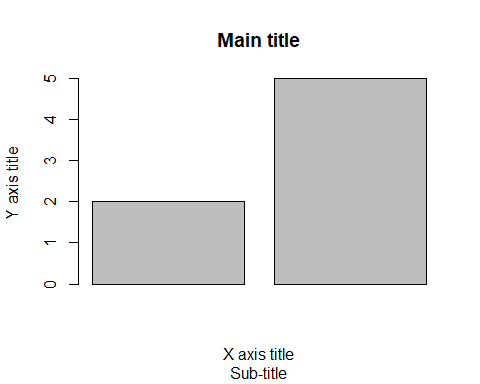
Pada grafik di R, kita dapat menambahkan judul dengan dua cara, yaitu: pada plot melalui parameter dan melalui fungsi plot(). Kedua cara tersebut tidak berbeda satu sama lain pada parameter input.

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

1. **main**: teks untuk judul.
2. **xlab**: teks untuk keterangan axis X.
3. **ylab**: teks untuk keterangan axis y.
4. **sub**: teks untuk sub-judul.

Berikut contoh sintaks penerapan masing-masing argumen tersebut beserta dengan output yang dihasilkan pada Gambar 23:

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



Gambar 23: Menambahkan Judul

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:

1. **col.main**: warna untuk judul.
2. **col.lab**: warna untuk keterangan axis.
3. **col.sub**: warna untuk sub-judul

Untuk kustomisasi font judul, kita dapat menambahkan argumen berikut:

1. **font.main**: *font style* untuk judul.
2. **font.lab**: *font style* untuk keterangan axis.
3. **font.sub**: *font style* untuk sub-judul.

**Penting!!!**

Nilai yang dapat dimasukkan antara lain:

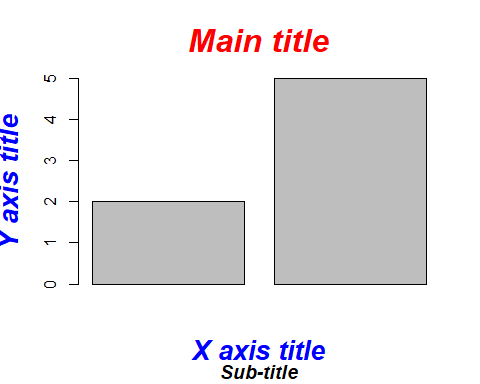
* **1**: untuk teks normal.
* **2**: untuk teks cetak tebal.
* **3**: untuk teks cetak miring.
* **4**: untuk teks cetak tebal dan miring.
* **5**: untuk font simbol.

Sedangkan untuk ukuran font, kita dapat menambahkan variabel berikut:

1. **cex.main**: ukuran teks judul.
2. **cex.lab**: ukuran teks keterangan axis.
3. **cex.sub**: ukuran teks sub-judul.

Berikut sintaks penerapan seluruh argumen tersebut beserta output yang dihasilkan pada Gambar 24:

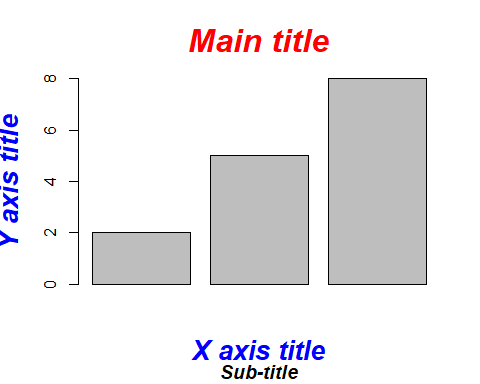
# menambahkan judul  
barplot(c(2,5),   
 # menambahkan judul  
 main="Main title",  
 xlab="X axis title",  
 ylab="Y axis title",  
 sub="Sub-title",  
 # kustomisasi warna font  
 col.main="red",   
 col.lab="blue",   
 col.sub="black",  
 # kustomisasi font style  
 font.main=4,   
 font.lab=4, 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



Gambar 24: 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 25:

# 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)



Gambar 25: Menambahkan Judul (3)

### Menambahkan Legend

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

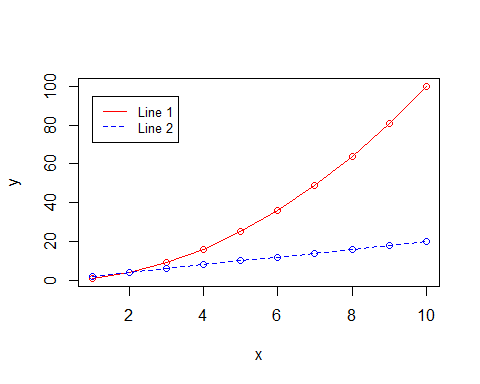
legend(x, y=NULL, legend, fill, col, bg)

**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 26:

# 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)  
  
# 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)



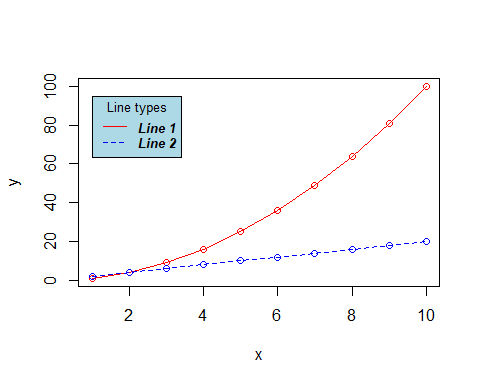
Gambar 26: Menambahkan legend

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

1. **title**: Judul legend
2. **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.
3. **bg**: warna background legend box.

Berikut adalah penerapan sintaks dan output yang dihasilkan pada Gambar 27:

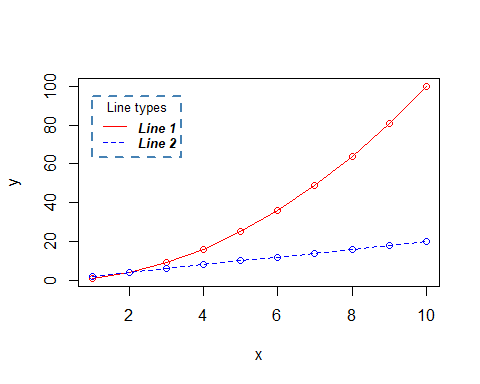
# membuat line plot  
plot(x,y, type="o", col="red", lty=1)  
  
# menambahkan line plot  
lines(x,z, type="o", col="blue", lty=2)  
  
# menambahkan legend  
legend(1, 95, legend=c("Line 1", "Line 2"),  
 col=c("red", "blue"), lty=1:2, cex=0.8,  
 title="Line types", text.font=4, bg='lightblue')



Gambar 27: Menambahkan legend (2)

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 28:

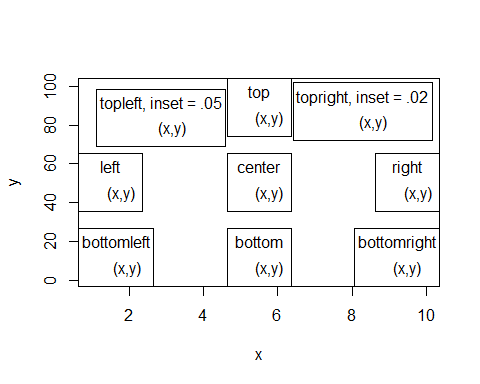
# membuat line plot  
plot(x,y, type="o", col="red", lty=1)  
  
# menambahkan line plot  
lines(x,z, type="o", col="blue", lty=2)  
  
# menambahkan legend  
legend(1, 95, legend=c("Line 1", "Line 2"),  
 col=c("red", "blue"), lty=1:2, cex=0.8,  
 title="Line types", text.font=4, bg='white',  
 box.lty=2, box.lwd=2, box.col="steelblue")



Gambar 28: Menambahkan legend (3)

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 29:

# 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",  
 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")



Gambar 29: Kustomisasi posisi legend

### 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:

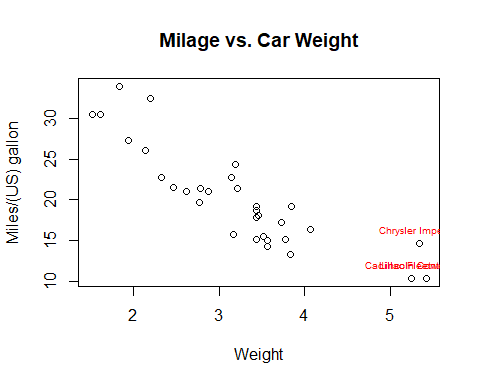
text(x, y, labels)

**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 30:

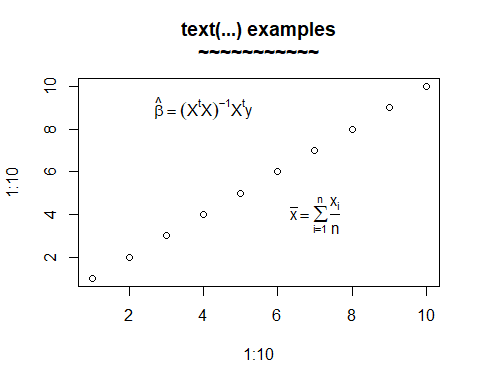
# tandai observasi yang memiliki nilai  
# mpg < 15 dan wt > 5  
d <- mtcars[mtcars$wt >= 5 & mtcars$mpg <= 15, ]  
  
  
# plot  
plot(mtcars$wt, mtcars$mpg, main="Milage vs. Car Weight",  
 xlab="Weight", ylab="Miles/(US) gallon")  
  
# menambahkan text  
text(d$wt, d$mpg, row.names(d),  
 cex=0.65, pos=3,col="red")



Gambar 30: Menambahkan teks

Sedangkan sintaks berikut adalah contoh bagaimana menambahkan persamaan kedalam bidang grafik dan output yang dihasilkan pada Gambar 31:

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)))



Gambar 31: Menambahkan teks (2)

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

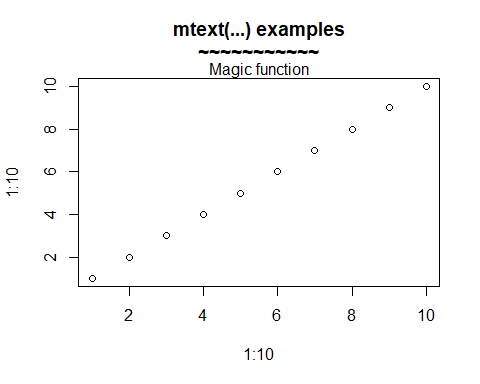
mtext(text, side=3)

**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 32:

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



Gambar 32: Menambahkan teks (3)

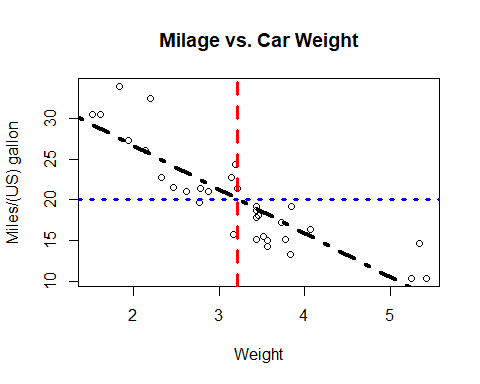
### Menambahkan Garis Pada Plot

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 33:

# 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)



Gambar 33: Menambahkan garis

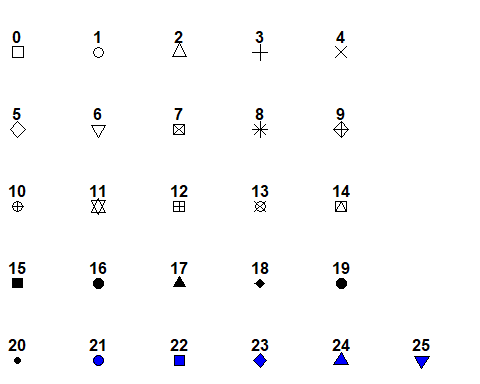
### Merubah Simbol plot dan Jenis Garis

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
* 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 34:

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()



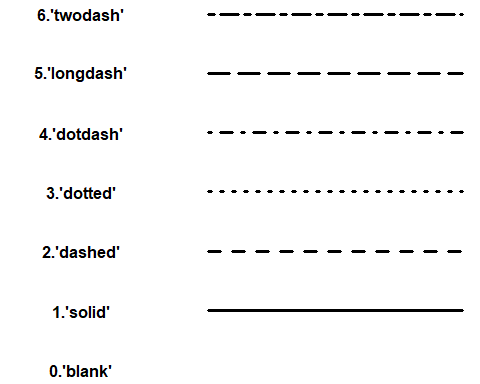
Gambar 34: Symbol plot

Pada R kita juga dapat mengatur jenis garis yang akan ditampilkan pada plot dengan menambahkan argumen lty= (*line type*) pada fungsi plot. Nilai yang 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 35:

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()



Gambar 35: Line type

### 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:

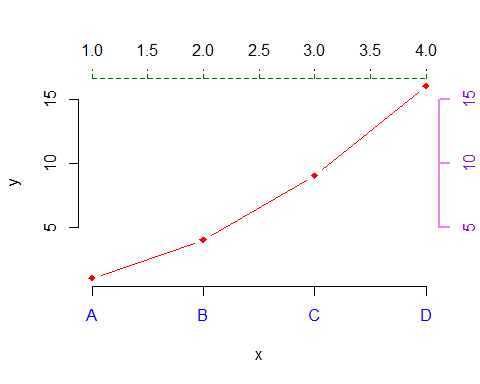
axis(side, at=NULL, labels=TRUE)

**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 36:

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



Gambar 36: Modifikasi axis

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:

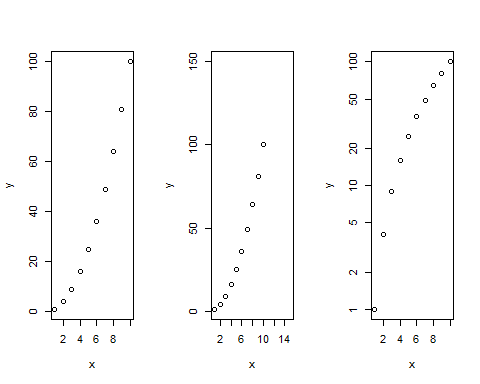
* **xlim**: limit nilai sumbu x dengan format: xlim(min, max).
* **ylim**: limit nilai sumbu x dengan format: ylim(min, max).

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 37:

# membagi jendela grafik menjadi 1 baris dan 3 kolom  
par(mfrow=c(1,3))  
  
# membuat vektor numerik  
x<-c(1:10); y<-x\*x  
  
# 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")



Gambar 37: Mengubah rentang dan skala axis

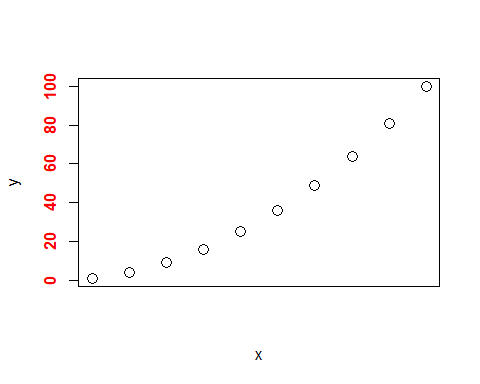
Kita dapat melakukan kustomisasi pada *tick mark*. Kustomisasi yang dapat dilakukan adalah merubah warna, *font style*, ukuran font, orientasi, serta menyembunyikan *tick mark*.

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
  + **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 38:

# 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")



Gambar 38: Kustomisasi tick mark

### 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 39:

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)

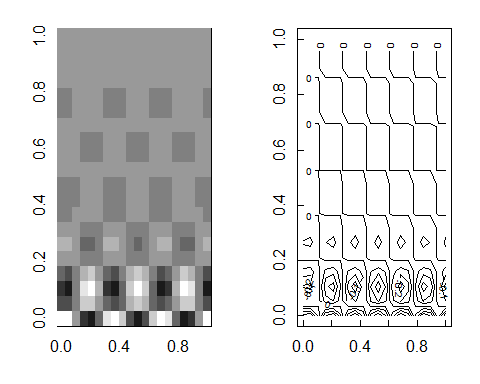


Gambar 39: Nama warna

## 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)

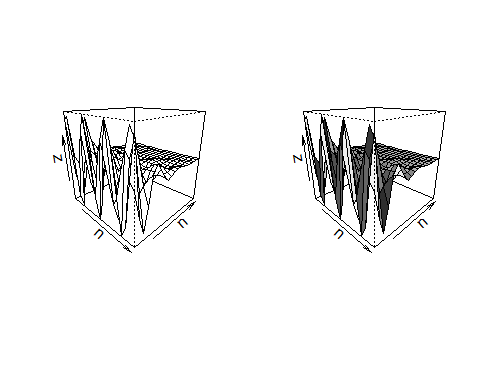


Gambar 40: image plot (kiri) dan contour plot (kanan)

par(mfrow=c(1,1))

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



Gambar 41: proyeksi 3 dimensi (kanan) dan proyeksi 3 dimensi dengan pewarnaan

par(mfrow=c(1,1))

## 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.

# Pemrograman dan Fungsi

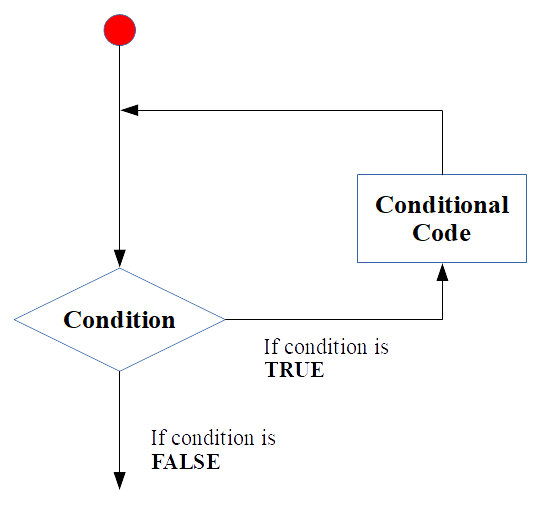
Kita telah membahas dasar-dasar kalkulasi menggunakan R pada Chapter 2. Pada Chapter 4 kita akan membahas dasar pemrograman menggunakan R. Pada chapter ini kita juga akan membahas bagaimana kita dapat membentuk suatu fungsi menggunakan R untuk pekerjaan yang berulang-ulang.

## 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 *library* 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 42



Gambar 42: Diagram umum loop (sumber: Primartha, 2018).

### 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.

### 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.

### 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*.

### 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

## 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:

# 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

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.

### 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

### 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

class(x) # cek kelas objek

## [1] "list"

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

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

## 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

## 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

### 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   
##   
##   
##

### 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

### 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

## Decision Making

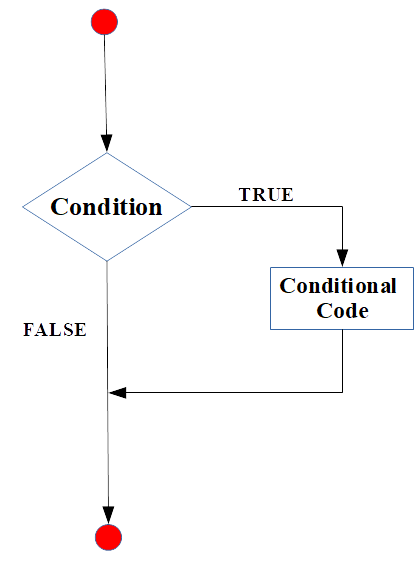
*Decicion Making* atau sering disebut sebagai *if then else statement* merupakan bentuk percabagan yang digunakan manakala kita ingin agar program dapat melakukan pengujian terhadap syarat kondisi tertentu. Pada Tabel 7 disajikan daftar percabangan yang digunakan pada R.

(#tab:percabangan) 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 |

### if statement

Pola *if statement* disajikan pada Gambar 43



Gambar 43: Diagram if statement (sumber: Primartha, 2018).

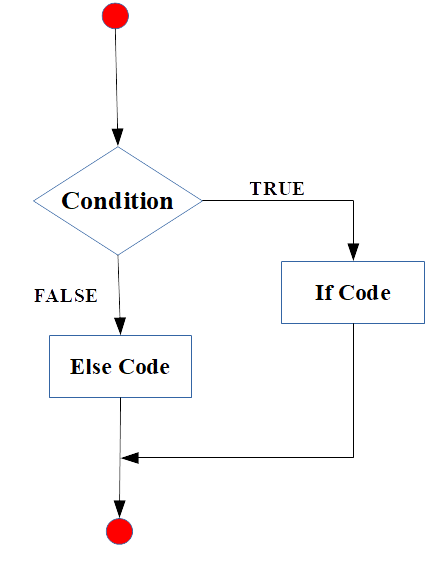
Berikut adalah contoh penerapan *if statement*:

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

## [1] "x adalah sebuah vector"

### if else statement

Pola dari *if else statement* disajikan pada Gambar 44



Gambar 44: Diagram if else statement (sumber: Primartha, 2018).

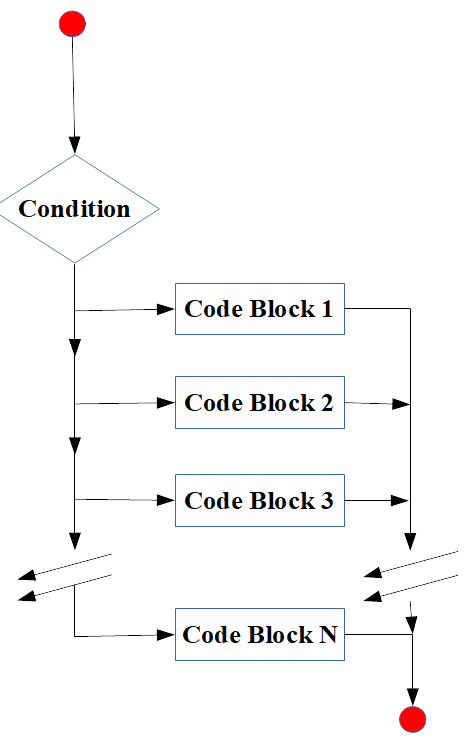
Berikut adalah contoh penerapan *if else statement*:

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"

### switch statement

Pola dari *switch statement* disajikan pada Gambar 45



Gambar 45: Diagram switch statement (sumber: Primartha, 2018).

Berikut adalah contoh penerapan *switch statement*:

y = 3  
  
x = switch(  
 y,  
 "Selamat Pagi",  
 "Selamat Siang",  
 "Selamat Sore",  
 "Selamat Malam"  
)  
  
print(x)

## [1] "Selamat Sore"

## 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

## 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).

## 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.

# 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.

## 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.

### 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 .
3. Metode analitik cocok untuk permasalahan dengan model terbatas dan sederhana, sedangkan metode numerik cocok dengan semua jenis permasalahan.

### 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

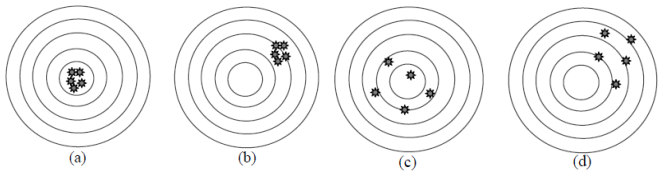
Sebelum digunakan dengan data sesungguhnya, program komputer perlu dilakukan uji coba dengan data simulasi dan dievaluasi hasilnya. jika hasil keluaran diyakini sudah sesuai, baru dioperasikan dengan data yang sesungguhnya.

* 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.

## Akurasi dan Presisi

Perhatikan Gambar 46 berikut:



Gambar 46: 4 ilustrasi terkait akurasi dan presisi

Pada Gambar 46 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 46 merupakan gambar yang menunjukkan seseorang telah berhasil mengenai bagian tengah sasaran tersebut dapat kita katakan pula tembakan pada kedua gambar tersebut akurat. Akurat dalam 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 dengan nilai observasi . Error absolut dapat dituliskan menggunakan Persamaan (1).

$$

Pengukuran lain yang sering digunakan untuk mengukur akurasi adalah error relatif. Berbeda dengan error absolut, error relatif membagi selisih antara nilai sebenarnya dan nilai observasi dengan nilai sebenarnya. Hasil yang diperoleh merupakan nilai tanpa satuan. Persamaan error relatif disajikan pada Persamaan (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 46, Gambar (a) dan (b) menunjukkan kepresisian yang tinggi. Hal ini terlihat dari jarak antara lubang peluru yang saling berdekatan dan mengelompok.

Berdasarkan Gambar 46 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.

## 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 (1) dan Persamaan (2) yang telah penulis jelaskan pada Chapter 5.2.

## 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.

# 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

## 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.

### Operasi Vektor

Misalkan saja diberikan vektor dan yang ditunjukkan pada Persamaan (3).

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.

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

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 yang belum dijumlahkan dengan vektor sampai seluruh elemen vektor dilakukan operasi penjumlahan.

Operasi lain yang dapat dilakukan pada vektor adalah menghitung *inner product* dan panjang vektor. Inner product dihitung menggunakan Persamaan (6).

Panjang vektor atau vektor yang telah dinormalisasi dihitung menggunakan Persamaan (7)

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

### Operasi matriks

Misalkan kita memiliki 2 buah matriks dan .

Jika salah satu matriks tersebut dijumlahkan atau dikurangkan dengan skalar.

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

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 (11).

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

## 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

## 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).

### *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 (12).

dimana untuk sampai dengan dan sampai dengan merupakan koefisien persamaan linier. untuk sampai dengan merupakan variabel bebas pada sistem persamaan linier.

Persamaan linier pada Persamaan (12) dapat dinyatakan ke dalam bentuk matriks pada Persamaan (13).

dimana:

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

matriks pada Persamaan (13) dapat diubah menjadi *augmented matrix*, yaitu: perluasan matriks A dengan menambahkan vektor B pada kolom terakhirnya.

Teorema 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 tidak nol atau terdapat .
* Determinan dari matriks koefisiensistem persamaan linier tidak sama dengan nol.

Untuk memperoleh penyelesaian sistem persamaan linier, Persamaan (15) perlu dilakukan operasi baris elementer. Hasil operasi baris dasar akan menghasilkan matriks *row echelon form* yang disajikan pada Persamaan (17).

Sehingga penyelesaian sistem persamaan linier dapat diperoleh menggunakan Persamaan (18).

Contoh 1: Selesaikan sistem persamaan berikut:

**Jawab**:

*Augmented matrix* sistem persamaan linier tersebut adalah sebagai berikut:

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

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 ().

Setelah diperoleh matriks *row echelon form* selanjutnya penyelesaian persamaan dapat dikerjakan menggunakan Persamaan (18).

**Algoritma Row Echelon Form**

1. Masukkan matriks , dan vektor beserta ukurannya
2. Buat *augmented matrix* namakan dengan
3. Untuk baris ke- dimana s/d , perhatikan apakah nilai sama dengan nol. **a)** **Bila iya**, lakukan *row swapping* antara baris ke- dan baris ke-, dimana tidak sama dengan nol. Bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian, **b)** **Bila tidak**, lanjutkan.
4. Untuk baris ke-, dimana s/d , lakukan operasi baris elementer:**a)** Hitung , **b)** untuk kolom , dimana s/d , hitung .
5. Hitung akar, untuk s/d 1 (bergerak dari baris pertama) menggunakan Persamaan (18).

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 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 (18).

Contoh 2: Dengan menggunakan fungsi ref\_matrix(), buatlah matriks *row echelon form* dari sistem persamaan linier berikut:

**Jawab**:

*Augmented matrix* dari sistem persamaan tersebut adalah sebagai berikut:

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:

### 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 (19)).

Sehingga penyelesaian persamaan linier tersebut adalah nilai dan atau:

Contoh 3: Selesaikan sistem persamaan berikut:

**Jawab**:

*Augmented matrix* dari persamaan linier tersebut adalah sebagai berikut:

Operasi baris elementer selanjutnya dilakukan pada matriks tersebut.

Penyelesaian persamaan linier tersebut adalah sebagai berikut:

**Algoritma Metode Eliminasi Gauss-Jordan**

1. Masukkan matriks dan vektor beserta ukurannya
2. Buat *augmented matrix* namakan dengan
3. Untuk baris ke- dimana s/d

* Perhatikan apakah nilai sama dengan nol:
  + **Bila ya**: pertukakan baris ke- dan baris ke-, dimana 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 dimana s/d , hitung

1. Untuk baris ke-, dimana s/d . Lakukan operasi baris elementer untuk kolom dimana s/d .

* Hitung
* Hitung

1. Penyelesaian untuk s/d 1 disajikan pada Persamaan (20).

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 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 4: Dengan menggunakan fungsi gauss\_jordan(), carilah penyelesaian sistem persamaan linier pada Contoh 1 dan Contoh 2:

**Jawab**:

Untuk Contoh 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 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

### 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 (21).

Penyelesaian persamaan tersebut disajikan pada Persamaan (22).

dimana .

Pada beberapa *textbook*, diagonal matriks sering dilambangkan dengan (diagonal bawah), (diagonal tengah), dan (diagonal atas). Bentuk matriksnya disajikan pada Persamaan (23).

**Algoritma Penyelesaian Matrik Tridiagonal**

1. Bentuk sistem persamaan linier menjadi matriks pada Persamaan (23).
2. Lakukan *foward sweep*. Setiap elemen diagonal dieliminasi menggunakan reduksi baris.

* Untuk
  + Hitung
  + Hitung
* Untuk s/d
  + Hitung
  + Hitung
* Hitung

1. Lakukan *backward sweep*. Setiap elemen diagonal dilakukan eliminasi.

* Untuk s/d
  + Hitung
* Hitung

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 5: Selesaikan sistem persamaan berikut menggunakan fungsi tridiagmatrix() dan fungsi gauss\_jordan()!

**Jawab**:

Langkah pertama untuk menyelesaikannya, kita harus merubah persamaan tersebut kedalam bentuk matriks

Untuk menyelesaikan persamaan tersebut menggunakan fungsi tridiagmatrix(), kita perlu membentuk vektor diagonal , , , dan .

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

### 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 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

### Penyelesaian Sistem Persamaan Linier Menggunakan Fungsi ’Solve.tridiag()`

Penyelesaian matriks tridiagonal selain menggunakan fungsi solve(), juga dapat menggunakan fungsi Solve.tridiag() dari *library* limSolve. Untuk menginstall dan mengaktifkan *library* 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 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

## Dekomposisi Matriks

Seringkali kita diminta untuk memperoleh nilai penyelesaian suatu persamaan linier , dimana nilai vektor yang selalu berubah-ubah. Penggunaan metode eliminasi Gauss mengharuskan untuk menyelesaikan sistem persamaan linier secara terpisah untuk setiap perubahan vektor . 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 , hasil pemecahan ini selanjutnya digunakan untuk memperoleh penyelesaian sistem persamaan linier melalui perkalian antara vektor dan hasil faktorisasi matriks .

### Dekomposisi LU

Misalkan kita memiliki persamaan linier seperti yang ditunjukkan oleh Persamaan (14). Pada metode dekomposisi LU, matriks difaktorkan menjadi matriks dan matriks , dimana ukuran kedua matriks tersebut harus sama dengan ukuran matriks atau dapat kita tuliskan bahwa hasil perkalian kedua matriks tersebut akan menghasilkan matriks .

Sehingga Persamaan (14) akan menjadi Persamaan (25).

Langkah penyelesaian sistem persamaan linier, diawali dengan menghadirkan vektor yang ditunjukkan pada Persamaan (26).

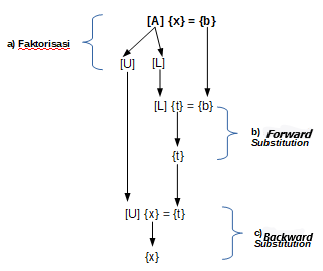
Langkah pada Persamaan (26) tidak dimaksudkan untuk menghitung vektor , melainkan untuk menghitung vektor . Vektor diperoleh dengan menggunakan Persamaan (27).

Kita dapat menyelesaikan sistem persamaan yang ditunjukkan pada Persamaan (26) dan Persamaan (27) menggunakan berbagai algoritma penyelesaian yang telah dibahas sebelumnya. Namun, karena matriks merupakan matriks segitiga bawah dengan nilai nol berada pada bagian atas diagonal utama, penyelesaian 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 akan berupa matriks persegi dengan ukuran , di mana merupakan jumlah baris matriks . Persamaan (27) dalam bentuk matriks akan terlihat seperti Persamaan (28).

Berdasarkan Persamaan (28), diketahui nilai . Nilai ini selanjutnya dapat digunakan untuk melakukan proses substitusi guna memperoleh seluruh nilai vektor . Proses ini disebut sebagai *foward substitution*. Proses substitusi dapat dituliskan menggunakan Persamaan (29).

Seteleh nilai vektor dihitung, kita dapat menghitung nilai pada Persamaan (30).

Jika diperhatikan, kita dapat mengetahui mengetahui nilai . 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 47.



Gambar 47: Tahapan dekomposisi LU.

Dekomposisi LU didasarkan pada operasi baris elementer. Pertama, kita perlu menemukan matriks segitiga atas yang sesuai dengan matriks . Solusi untuk melakukan dekomposisi bisa jadi tak terhingga, namun solusi yang paling sederhana adalah mengubah matriks menjadi matriks *row echelon form*. Kedua, harus menjadi matriks segitiga bawah yang mereduksi ke- dengan mengikuti operasi baris yang sama yag menghasilkan . Kita dapat menggunakan algoritma Doolittle untuk menghasilkan , 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 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 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, . Matriks ini merupakan matriks identitas dengan ukuran sama dengan matriks . Jika pertukaran baris dilakukan selama proses pembentukan matriks , maka pertukaran baris yang sama juga akan diimplemenntasikan pada matriks . oleh karena itu, dalam praktiknya matriks dan perkalian dengan matriks berfungsi untuk mengembalikan urutan baris.

Contoh 6: Selesaikan sistem persamaan linier berikut menggunakan faktorisasi LU

**Jawab**:

Nayatakan sistem persamaan tersebut ke dalam bentuk matriks .

Lakukan operasi baris elementer pada matriks untuk memperoleh matriks . Urutan operasi baris elementer yang dilakukan adalah sebagai berikut:

* ,
* ,
* ,
* ,
* ,

Simpan pengali tiap tahapan pada masing-masing elemen matriks . Hasil operasi tersebut akan menghasilkan matriks triangular .

Untuk matriks sebagai berikut:

Karena pada proses operasi baris elementer tidak terdapat operasi pertukaran baris, maka matriks tidak mengalami perubahan:

Lakukan operasi *forward substitution* menggunakan Persamaan (28).

Berdasarkan hasil perhitungan diperoleh nilai vektor .

Operasi terakhir yang perlu dilakukan untuk memperoleh nilai adalah dengan melakukan *backward substitution* menggunakan nilai vektor yang telah dihitung.

Berdasarkan hasil perhitungan diperoleh nilai sebagai berikut:

**Algoritma Dekomposisi LU**

1. Masukkan matriks , dan vektor beserta ukurannya
2. Lakukan langkah poin ke-4 s/d poin 5 untuk meperoleh matriks .
3. Untuk baris ke- di mana s/d , perhatikan apakah nilai sama dengan nol.

* **Bila iya**, lakukan *row swapping* antara baris ke- dan baris ke-, dimana tidak sama dengan nol. Bila tidak ada berarti perhitungan tidak bisa dilanjutkan dan proses dihentikan dengan tanpa penyelesaian.
* **Bila tidak**, lanjutkan.

1. Untuk baris ke-, dimana s/d , lakukan operasi baris elementer:

* Hitung
* untuk kolom , dimana s/d , hitung

1. Lakukan langkah poin ke-7 s/d poin 9 untuk memperoleh matriks
2. Untuk diagonal matriks isikan dengan nilai 1 dan elemen di atas diagonal dengan nilai nol.
3. Untuk elemen di bawah diagonal isikan dengan faktor pengali operasi baris elementer matriks .
4. Lakukan proses *forward substitution* menggunakan Persamaan (29) untuk memperoleh nilai vektor .
5. Lakukan *backward substituion* menggunakan Persamaan (18).

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 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 , kita dapat mengalikan matriks , , dan .

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 7: Lakukan dekomposisi LU pada matriks berikut dan lakukan pengecekan apakah perkalian hasil dekomposisi matriks akan menghasilkan matriks semula!

**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 .

decomp$P %\*% decomp$L %\*% decomp$U

## [,1] [,2] [,3]  
## [1,] 0 1 -2  
## [2,] 1 5 9  
## [3,] 7 -1 -5

Fungsi lu() pada *library* Matrix dapat digunakan untuk melakukan dekomposisi LU. Untuk meggunakan fungsi tersebut, kita harus menginstall dan mengaktifkan *library* 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::Matrix(round(rnorm(9),2), nrow=3)  
  
# dekomposisi  
lum <- Matrix::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 <- Matrix::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,] . | .

### Dekomposisi Cholesky

Dekomposisi Cholesky memberikan faktorisasi matriks alternatif sehingga , di mana merupakan transpose konjugat dari matriks . Dalam kasus ini, penulis hanya bekerja dengan matriks rill dengan nilai rill dan bagian imajiner nol. Jadi untuk tujuan *sub-chapter* ini, matriks hanyalah transpose dari matriks .

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 dan pada matriks , . Definit positif berarti bahwa setiap entri pivot (nilai elemen diagonal utama) selelu bernilai positif. Selain itu, untuk matriks definit positif, hubungan untuk semua vektor, .

Karena transpose dari matriks , maka untuk semua nilai dan . Tanpa kendala (*constraint*) ini, dekomposisi Cholesky akan mirip dekomposisi LU. Tetapi dengan kendala ini, nilai elemen matriks dan harus dipilih dengan cermat sehingga hubungan berlaku. Bentuk dekomposisi Cholesky disajikan pada Persamaan (31).

Untuk setiap elemen matriks memiliki hubungan yang dituliskan pada Persamaan (32).

Berdasarkan Persamaan (31), sejumlah nilai elemen dan adalah nol. Nilai tiap elemen diagonal utama yang tidak bernilai nol dihitung menggunakan Persamaan (33).

Elemen diagonal dihitung menggunakan Persamaan (34)

**Algoritma Dekomposisi Cholesky**

1. Masukkan matriks , dan vektor beserta ukurannya .
2. Untuk elemen matriks , hitung menggunakan Persamaan (34).
3. Untuk nilai diagonal utama matriks , hitung menggunakan Persamaan (33).
4. Untuk memperoleh matriks , lakukan transpose pada matriks .
5. Untuk memperoleh nilai ,

* Hitung vektor menggunakan Persamaan (27).
* Hitung vektor menggunakan Persamaan (26), dimana matriks .

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 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!

**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 *library* Matrix. Pada fungsi tersebut, kita hanya perlu menginputkan objek matrik kedalamnya. Berikut adalah contoh penerapan fungsi tersebut menggunakan matriks pada Contoh 8.

chol(a)

## [,1] [,2] [,3]  
## [1,] 3 -1 2  
## [2,] 0 4 -2  
## [3,] 0 0 2

**Penting!!!**

Fungsi chol() hanya menampilkan matriks . Untuk menampilkan matriks , kita perlu melakukan transpose

### 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 *library* base.

#### 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

#### *Singular Value Decomposition*

*Singular value decomposition* (SVD) merupakan algoritma faktorisasi matriks yang mendekomposisi matriks segiempat menjadi matriks , dimana merupakan mmatriks diagonal non negatif, dan merupakan matriks *unitary*, dan merupakan matriks tanspose konjugat dari matriks . Algoritma ini banyak digunakan dalam analisis *principal component*.

Pada R, SVD dapat dilakukan menggunakan fungsi svd() dari *library* 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

#### 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 , dimana merupakan matriks diagonal yang terbentuk dari nilai eigen, dan merupakan vektor eigen. Proses dekomposisi ini akan berguna bagi pembaca yang ingin mempelajari *principal component analysis*.

Fungsi eigen() pada *library* 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

## 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 (35) menampilkan hubungan antara jumlah kuadrat dan nilai toleransi pada proses iterasi.

dimana merupakan iterasi ke- dari algoritma dan merupakan nilai toleransi maksimum yang diterima.

### Iterasi Jacobi

Untuk menyelesaikan matriks menggunakan metode iterasi, kita dapat mulai dengan premis terdapat matriks dan vektor dan b, sehingga . Dengan menggunakan metode Jacobi, pertama-tama kita dapat amati bahwa terdapat matriks dan yang memiliki hubungan . Berdasarkan kedua hubungan tersebut, dapat diturunkan operasi matriks melalui persamaan berikut:

Persamaan (39) merupakan persamaan yang dapat kita gunakan untuk memperoleh nilai . Jika kita menulis kembali persamaan tersebut, maka kita akan memperoleh persamaan yang digunakan sebagai acuan iterasi Jacobi.

dimana merupakan matriks diagonal dengan nilai elemen diagonal berupa diagonal utama matriks . Invers dari matriks secara sederhana sebagai matriks diagonal sama dengan satu dibagi dengan elemen diagonal utama matriks . Matriks identik dengan matriks . Namun, diagonal utamanya bernilai nol. Suatu iterasi dikatakan konvergen jika jumlah kuadrat dari vektor dan vektor 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 9: Selesaikan sistem persamaan berikut menggunakan iterasi Jacobi!

**Jawab**:

Berdasarkan matriks (matriks koefisien), kita dapat memastikan bahwa matriks tersebut memiliki nilai dominan pada elemen diagonal utama. Sebagai contoh:

Untuk mempermudah proses iterasi, kita akan menggunakan bantuan R untuk melakukan komputasi. Langkah pertama yang perlu dilakukan adalah menyiapkan matriks , vektor , dan vektor (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 .

(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<-A-diag(diag(A)))

## [,1] [,2] [,3]  
## [1,] 0 2 3  
## [2,] 2 0 4  
## [3,] 1 3 0

Iterasi selanjutnya dilakukan menggunakan Persamaan (40).

**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 .

Proses iterasi berlangsung sampai dengan iterasi ke-62 dengan nilai akhir sebagai berikut:

**Algoritma Iterasi Jacobi**

1. Masukkan matriks , dan vektor beserta ukurannya .
2. Hitung invers matriks , dimana nilai invernya merupakan matriks diagonal dari satu per diagonal utama matriks .
3. Hitung matriks , dimana merupakan selisih matriks dikurangi dengan matriks diagonal dengan entri dari diagonal utama matriks .
4. Tetapkan vektor estimasi.
5. Tetapkan nilai toleransi maksimum yang dapat diterima.
6. Lakukan iterasi menggunakan Persamaan (40).
7. Hitung akar jumlah kuadrat dari vektor dan vektor .
8. Jadikan nilai sebagai nilai taksiran untuk iterasi berikutnya.
9. Hentikan proses iterasi jika telah memenuhi syarat yang ditampilkan pada Persamaan (35).

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 10: Selesaikan sistem persamaan berikut menggunakan fungsi jacobi()

**Jawab**:

Matriks (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 sesungguhnya dapat diperoleh menggunakan fungsi solve().

solve(A,b)

## [1] 2.425476 3.573016 1.925954

Berdasarkan hasil perhitungan, vektor 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 lebih besar dari nilai elemen lainnya pada satu kolom. Selain itu, nilai diagonal matriks tidak boleh sama dengan nol agar inver matriks 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.

### Iterasi Gauss-Seidel

Metode iterasi Gauss-Seidel melakukan dekomposisi pada matriks menjadi matriks segitiga atas dan matriks segitiga bawah . Dekomposisi ini tidak sama dengan dekomposisi LU pada Chapter 6.4.1. Matriks pada metode Gauss-Seidel merupakan elemen (entri) matriks pada bagian atas diagonal utama, sedangkan matriks merupakan elemen diagonal utama dan bagian bawah diagonal utama matriks . Elemen selain yang penulis sebutkan pada kedua matriks tersebut akan bernilai nol. Persamaan iterasi Gauss-Seidel ditampilkan pada Persamaan (41).

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 11: Selesaikan sistem persamaan pada Contoh 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 dan .

# 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 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 .

# tebakan awal nilai x  
(x <- rep(0, length(b)))

## [1] 0 0 0

Lakukan iterasi menggunakan Persamaan (41).

**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 sebesar:

**Algoritma Iterasi Gauss-Seidel**

1. Masukkan matriks , dan vektor beserta ukurannya .
2. Lakukan dekomposisi LU, dimana matriks merupakan matriks segitiga bawah dengan nilai entri diagonal utama matriks dan bagian bawah diagonalnya dan matriks merupakan matriks segitiga atas dengan entri berasal dari elemen atas diagonal utama matriks . Isi elemen lain yang tidak disebut pada kedua matriks tersebut dengan nol.
3. Tetapkan vektor estimasi.
4. Tetapkan nilai toleransi maksimum yang dapat diterima.
5. Lakukan iterasi menggunakan Persamaan (41).
6. Hitung akar jumlah kuadrat dari vektor dan vektor .
7. Jadikan nilai sebagai nilai taksiran untuk iterasi berikutnya.
8. Hentikan proses iterasi jika telah memenuhi syarat yang ditampilkan pada Persamaan (35).

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 12: Selesaikan sistem persamaan pada Contoh 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

## 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.

### 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 dan sumbu . 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:

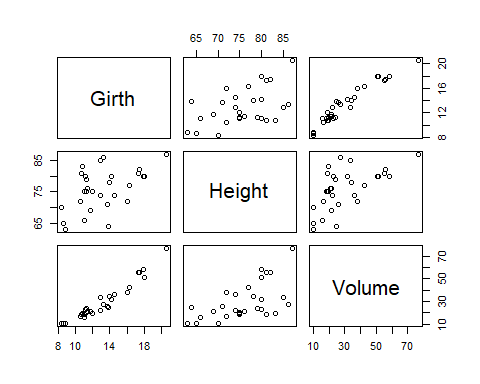
head(trees)

## 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:



(#fig:trees, LUfig)Scatterplot matriks dataset trees

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 (42).

dimana merupakan intersep persamaan regresi linier dan nilai 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 (43).

dimana merupakan vektor dari variabel respon (Volume) dan matrik merupakan matriks variabel prediktor (variabel Girth dan Heiht).

Untuk menginputkan intercept kedalam persamaan linier kita perlu menmabhakan satu kolom di awal matriks yang berisi nilai 1. Berikut adalah sintaks yang digunakan untuk membentuk matriks :

# 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 . 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 , kita dapat mencarinya dengan cara menyelesaikan Persamaan (43). 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 (44).

Pembaca juga dapat menggunakan fungsi lain untuk memperoleh nilai koefisien tersebut, seperti: lu\_solve()dansolve(). 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

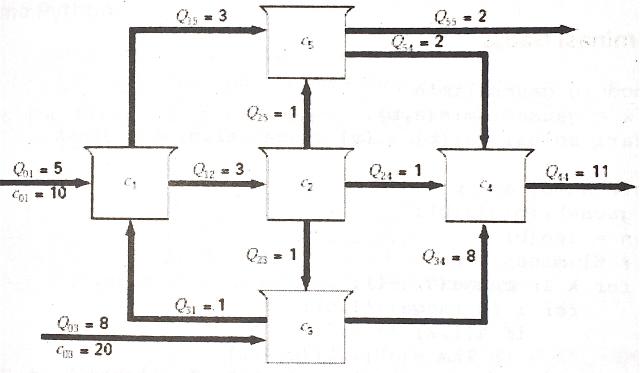
R juga menyediakan fungsi untuk membentuk model regresi linier. Fungsi yang digunakan adalah lm(). Berikut sintaks yang digunakan untuk membentuk model linier menggunakan fungsi lm():

lm(Volume~Girth+Height, data=trees)

##   
## Call:  
## lm(formula = Volume ~ Girth + Height, data = trees)  
##   
## Coefficients:  
## (Intercept) Girth Height   
## -57.9877 4.7082 0.3393

### 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 48. Debit air () dan konsentrasi zat pencemar () 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 (45).



Gambar 48: Aliran massa dalam reaktor.

Berdasarkan Gambar 48, dapat dibentuk lima buah sistem persamaan linier. Persamaan linier yang terbentuk disajikan sebagai berikut:

Untuk menyelesaiakan sistem persamaan linier tersebut dan memperoleh nilai dari masing-masing reaktor, kiat perlu mengubahnya dulu kedalam bentuk matriks . Berikut adalah matriks yang terbentuk:

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:

## 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.

## Latihan

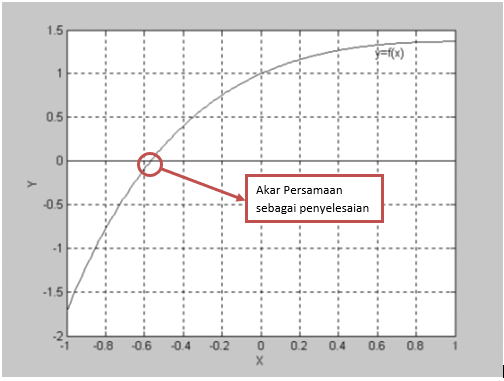
1. Selesaikan sistem persamaan linier berikut menggunakan eliminasi Gauss!
2. Carilah penyelesaian dari sistem persamaan linier soal no.1 menggunakan algoritma dekomposisi LU!
3. Tunjukan 5 iterasi pertama sistem persamaan linier berikut menggunakan algoritma Jacobi dan Gauss-Seidel!
4. Gunakan fungsi 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 kedalam fungsi solve() dan apa yang akan terjadi jika selanjutnya argumen pada fungsi tersebut juga menyertakan vektor ?
6. Dengan menggunakan dataset mtcars buatlah persamaan linier variabel mpg sebagai fungsi dari variabel wt, hp, dan qsec menggunakan algoritma dekomposisi LU?

# Akar Persamaan Non-Linier

Persamaan non-linier dapat diartikan sebagai persamaan yang tidak mengandung syarat seperti persamaan linier, sehingga persamaan non-linier dapat merupakan:

1. Persamaan yang memiliki pangkat selain satu (misal: )
2. Persamaan yang mempunyai produk dua variabel (misal: )

Dalam penyelesaian persamaan non-linier diperlukan akar-akar persamaan non-linier, dimana akar sebuah persamaan non-linier merupakan nilai yang menyebabkan nilai sama dengan nol. Dalam hal ini dapat disimpulkan bahwa akar-akar penyelesaian persamaan non-linier merupakan titik potong antara kurva dengan sumbu . Ilustrasi penjelasan tersebut ditampilkan pada Gambar 49.



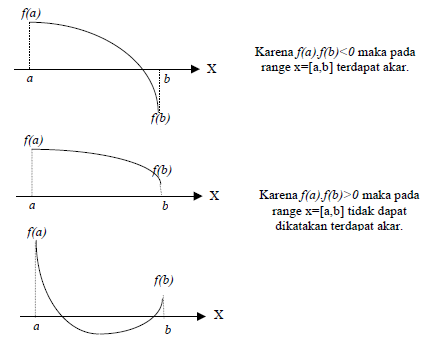
Gambar 49: Penyelesaian persamaan non-linier.

Contoh sederhana dari penentuan akar persamaan non-linier adalah penentuan akar persamaan kuadratik. Secara analitik penentuan akar persamaan kuadratik dapat dilakukan menggunakan Persamaan (47).

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 2: (root) Suatu range x=[a,b] mempunyai akar bila f(a) dan f(b) berlawanan tanda atau memenuhi f(a).f(b)<0

Untuk memahami teorema tersebut perhatikan ilustrasi pada Gambar 50.



Gambar 50: Ilustrasi teorema Bolzano.

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

## Metode Tertutup

Metode tertutup disebut juga metode *bracketing*. Disebut sebagai metode tertutup karena dalam pencarian akar-akar persamaan non-linier dilakukan dalam suatu selang .

### Metode Tabel

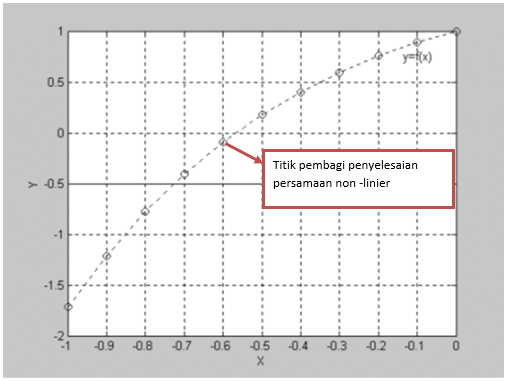
Penyelesaian persamaan non-linier menggunakan metode tabel dilakukan dengan membagi persamaan menjadi beberapa area, dimana untuk dibagi sebanyak bagian dan pada masing-masing bagian dihitung nilai sehingga diperoleh nilai pada setian bagian.

Bila nilai atau mendekati nol, dimana , maka dikatakan bahwa adalah penyelesaian persamaan . Bila tidak ditemukan, dicari nilai dan yang berlawanan tanda. Bila tidak ditemukan, maka persamaan tersebut dapat dikatakan tidak mempunyai akar untuk rentang .

Bila akar persamaan tidak ditemukan, maka ada dua kemungkinan untuk menentukan akar persamaan, yaitu:

1. Akar persamaan ditentukan oleh nilai mana yang lebih dekat. Bila , maka akarnya . Bila , maka akarnya .
2. Perlu dicari lagi menggunakan rentang .

Secara grafis penyelesaian persamaan non-linier menggunakan metode table disajikan pada Gambar 51.



Gambar 51: Ilustrasi metode tabel.

**Algoritma Metode Tabel**

1. Definisikan fungsi
2. Tentukan rentang untuk yang berupa batas bawah dan batas atas .
3. Tentukan jumlah pembagi
4. Hitung step pembagi
5. Untuk s/d , hitung:
6. Untuk s/d , dimana

* Bila , maka akarnya
* Bila , maka:
  + , maka akarnya
  + Bila tida, adalah penyelesaian atau dapat dikatakan penyelesaian berada diantara dan .

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:

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 13: Carilah akar persamaan pada rentang ?

**Jawab**:

Sebagai permulaan, jumlah pembagi yang digunakan adalah . Dengan menggunakan fungsi root\_table() diperoleh hasil yang disajikan pada Tabel 8.

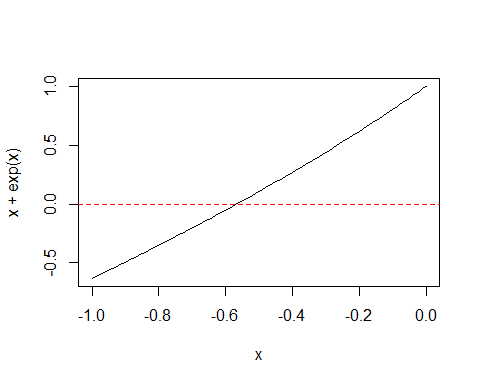
tabel <- root\_table(f=function(x){x+exp(x)},  
 a=-1, b=0, N=10)

Tabel 8: 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 |

Berdasarkan Tabel 8 diperoleh penyelesaian di antara dan dengan nilai masing-masing sebesar dan , sehingga dapat diambil penyelesaian . Kita dapat terus melakukan iterasi sampai memperoleh nilai < nilai toleransi dengan terus merubah rentang yang diberikan. Iterasi berikutnya dengan nilai pembagi sama dan rentang nilai diperoleh nilai dan .

Untuk melihat gambaran lokasi akar, kita dapat pulang mengeplotkan data menggunakan fungsi plot. Berikut adalah fungsi yang digunakan:



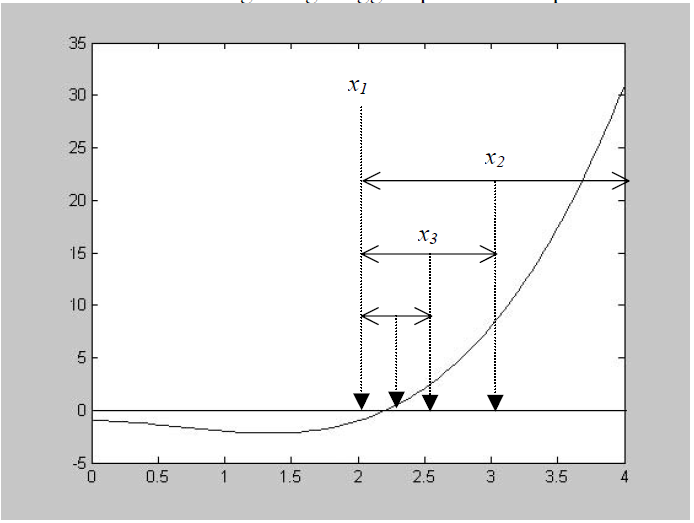
Gambar 52: Plot fungsi x+exp(x) pada rentang -1 sampai 0.

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 memdapatkan 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.

### Metode Biseksi

Prinsip metode bagi dua adalah mengurung akar fungsi pada interval atau pada nilai batas bawah dan batas atas . 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 53.



Gambar 53: Ilustrasi metode biseksi.

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**

1. Definisikan fungsi
2. Tentukan rentang untuk yang berupa batas bawah dan batas atas .
3. Tentukan nilai toleransi dan iterasi maksimum
4. Hitung dan
5. Hitung:
6. Hitung
7. Bila , maka dan . Bila tidak, dan
8. Bila atau iterasi maksimum maka proses dihentikan dan didapatkan akar=, dan bila tidak ulangi langkah 6.
9. Jika sudah diperoleh nilai dibawah nilai toleransi, nilai akar selanjutnya dihitung berdasarkan Persamaan (51) dengan nilai dan 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 14: Carilah akar persamaan pada rentang dengan nilai toleransi sebesar ?

**Jawab**:

Langkah pertama dalam penghitungan adalah menghitung nilai menggunakan Persamaan (51).

Hitung nilai dan .

Berdasarkan hasil perhitungan diperoleh:

Sehingga dan . Iterasi dilakukan kembali dengan menggunakan nilai 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)

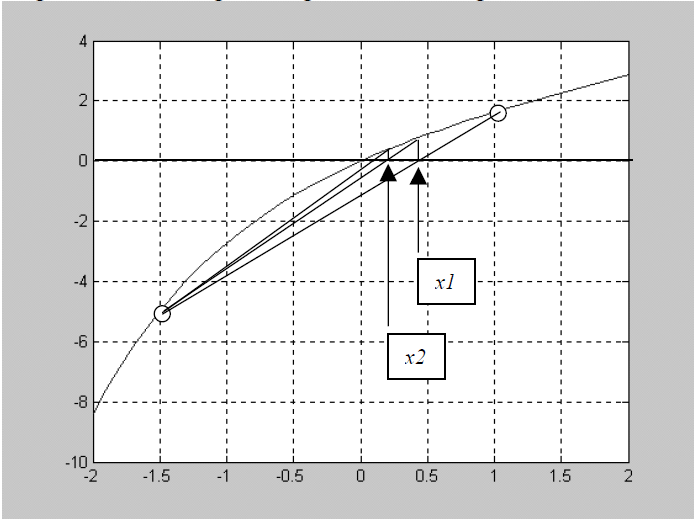
## $`function`  
## function (x)   
## {  
## x \* exp(-x) + 1  
## }  
## <bytecode: 0x00000000059c7178>  
##   
## $root  
## [1] -0.5671433  
##   
## $iter  
## [1] 24

Berdasarkan hasil iterasi diperoleh akar persamaan dan iterasi yang diperlukan untuk memperolehnya sebanyak iterasi.

### 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 (52).

Ilustrasi dari metode regula falsi disajikan pada Gambar 54.



Gambar 54: Ilustrasi metode regula falsi.

**Algoritma Metode Regula Falsi**

1. Definisikan fungsi
2. Tentukan rentang untuk yang berupa batas bawah dan batas atas .
3. Tentukan nilai toleransi dan iterasi maksimum
4. Hitung dan
5. Untuk iterasi s/d

* Hitung nilai berdasarkan Persamaan (52)
* Hitung
* Hitung
* Jika , maka dan . Jika tidak, dan .

1. Akar persamaan adalah

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 15: Selesaikan persamaan non-linier pada Contoh 14 menggunakan metode regula falsi pada rentang dengan nilai toleransi sebesar ?

**Jawab**:

Langkah pertama penyelesaian dilakukan dengan mencari nilai dan .

Hitung nilai dan .

Berdasarkan hasil perhitungan diperoleh:

Sehingga dan . Iterasi dilakukan kembali dengan menggunakan nilai 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: 0x000000001b87d108>  
##   
## $root  
## [1] -0.5671433  
##   
## $iter  
## [1] 15

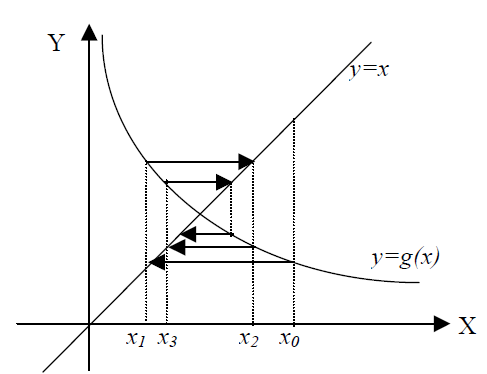
Berdasarkan hasil perhitungan diperoleh nilai dan jumlah iterasi yang diperlukan adalah . Jumlah ini lebih sedikit dari jumlah iterasi yang diperlukan pada metode iterasi biseksi yang juga menunjukkan metode ini lebih cepat memperoleh persamaan dibandingkan metode biseksi.

## 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.

### Metode Iterasi Titik Tetap

Metode iterasi titik tetap merupakan metode penyelesaian persamaan non-linier dengan cara menyelesaikan setiap variabel yang ada dalam suatu persamaan dengan sebagian yang lain sehingga diperoleh untuk masing-masing variabel . Sebagai contoh, untuk menyelesaikan persamaan , maka persamaan tersebut perlu diubah menjadi atau . Secara grafis metode ini diilustrasikan seperti Gambar 55.



Gambar 55: Ilustrasi metode iterasi titik tetap.

**Algoritma Metode Iterasi Titik Tetap**

1. Definisikan dan
2. Tentukan nilai toleransi dan iterasi masimum (N)
3. Tentukan tebakan awal
4. Untuk iterasi s/d atau , Hitung
5. Akar persamaan adalah 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 16: Selesaikan persamaan non-linier pada Contoh 14 menggunakan metode iterasi titik tetap?

**Jawab**:

Untuk menyelesaikan persamaan non-linier tersebut kita perlu mentransformasi persamaan non-linier tersebut terlebih dahulu.

Untuk tebakan awal digunakan nilai

Nilai tersebut selanjutnya dijadikan nilai input pada iterasi selanjutnya:

iterasi terus dilakukan sampai diperoleh .

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)

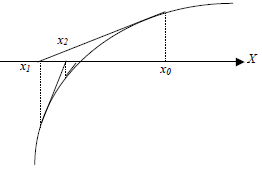
## $`function`  
## function (x)   
## {  
## -1/exp(-x)  
## }  
## <bytecode: 0x00000000066403d0>  
##   
## $root  
## [1] -0.5671433  
##   
## $iter  
## [1] 29

Berdasarkan hasil iterasi diperoleh nilai dengan jumlah iterasi yang diperlukan sebanyak kali. Jumlah iterasi akan bergantung dengan nilai tebakan awal yang kita berikan. Semakin dekat nilai tersebut dengan akar, semakin cepat nilai akar diperoleh.

### 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 (53).

Ilustrasi metode Newton-Raphson disajikan pada Gambar 56.



Gambar 56: Ilustrasi metode Newton-Raphson.

**Algoritma Metode Newton-Raphson**

1. Definisikan dan
2. Tentukan nilai toleransi dan iterasi masimum (N)
3. Tentukan tebakan awal
4. Hitung dan
5. Untuk iterasi s/d atau , hitung menggunakan Persamaan (53)
6. Akar persamaan merupakan nilai 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 17: Selesaikan persamaan non-linier menggunakan metode Newton-Raphson?

**Jawab**:

Untuk dapat menggunakan metode Newton-Raphson, terlebih dahulu kita perlu memperoleh turunan pertama dari persamaan tersebut.

Tebakan awal yang digunakan adalah .

Hitung nilai baru:

Untuk mempercepat proses iterasi, kita dapat menggunakan fungsi root\_newton(). Berikut adalah sintaks yang digunakan:

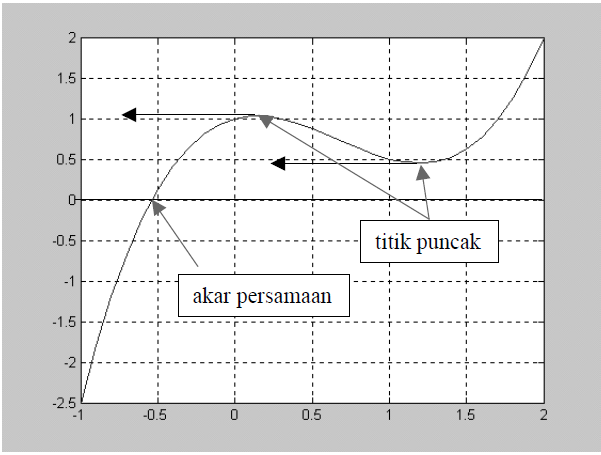
root\_newton(function(x){x-exp(-x)},  
 function(x){1+exp(-x)},  
 x0=0)

## $`function`  
## function (x)   
## {  
## x - exp(-x)  
## }  
## <bytecode: 0x000000001e8aa9b8>  
##   
## $root  
## [1] 0.5671433  
##   
## $iter  
## [1] 5

Berdasarkan hasil iterasi diperoleh akar penyelesaian persamaan non-linier adalah dengan jumlah iterasi yang diperlukan adalah iterasi.

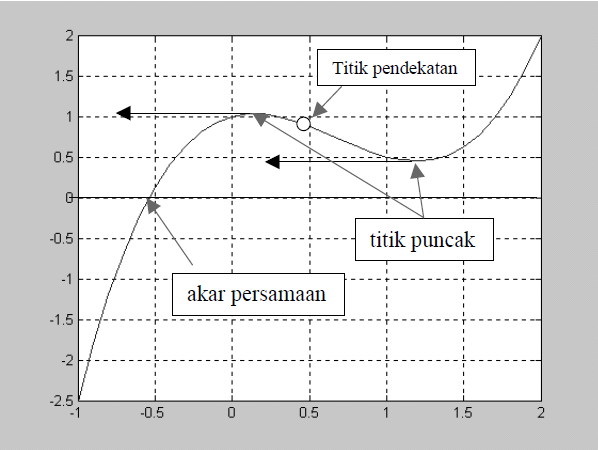
Dalam penerapannya metode Newton-Raphson dapat mengalami kendala. Kendala yang dihadapi adalah sebagai berikut:

1. titik pendekatan tidak dapat digunakan jika merupakan titik ekstrim atau titik puncak. Hal ini disebabkan pada titik ini nilai . Untuk memahaminya perhatikan ilustasi yang disajikan pada Gambar 57. Untuk menatasi kendala ini biasanya titik pendekatan akan digeser.



Gambar 57: Ilustrasi titik pendekatan di titik puncak.

1. Sulit memperoleh penyelesaian ketika titik pendekatan berada diantara 2 titik stasioner. Untuk memahami kendala ini perhatikan Gambar 58. Untuk menghindarinya, penentuan titik pendekatan dapat menggunakan bantuan metode tabel.



Gambar 58: Ilustrasi titik pendekatan diantara 2 titik stasioner.

1. Turunan persamaan sering kali sulit untuk diperoleh (tidak dapat dikerjakan dengan metode analitik).

### 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 (54).

Nilai merupakan transformasi persamaan tersebut.

Bila dan dan diketahui, maka titik ke adalah:

Bila titik dianggap akar persamaan maka nilai , sehingga diperoleh:

atau

Berdasarkan Persamaan (60) 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.

**Algoritma Metode Secant**

1. Definisikan dan
2. Tentukan nilai toleransi dan iterasi masimum (N)
3. Tentukan tebakan awal dan
4. Hitung dan
5. Untuk iterasi s/d atau , hitung menggunakan Persamaan (60)
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 18: Selesaikan persamaan non-linier pada Contoh 17 menggunakan metode Secant?

**Jawab**:

Untuk menyelesaikan persamaan tersebut digunakan nilai pendekatan awal dan .

Hitung nilai dan .

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: 0x0000000007ee1678>  
##   
## $root  
## [1] 0.5671433  
##   
## $iter  
## [1] 6

Berdasarkan hasil iterasi diperoleh nilai akar penyelesaian adalah dengan iterasi dilakukan sebanyak 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.

## Penyelesaian Persamaan Non-Linier Menggunakan Fungsi uniroot dan uniroot.all

*Library* 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 dengan jumlah iterasi sebanyak iterasi dan tingkat presisi sebesar .

Fungsi lain yang dapat digunakan untuk mencari akar persamaan adalah uniroot.all() dari *library* rootSolve. Fungsi ini mengatasi kelemahan dari uniroot(), dimana uniroot() tidak bekerja jika fungsi hanya menyentuh dan tidak melewati sumbu nol . 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

## Akar Persamaan Polinomial Menggunakan Fungsi polyroot

Fungsi polyroot() pada *library* 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 :

polyroot(c(1,0,1))

## [1] 0+1i 0-1i

Contoh lainnya adalah mencari akar polinomial :

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.

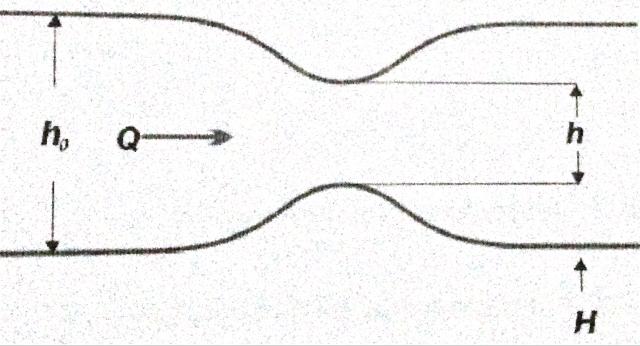
## Studi Kasus

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.

### Persamaan Van Der Walls

### Hukum Bernoulli

Misalkan terdapat sebuah saluran dengan penampang sesuai dengan Gambar 59.



Gambar 59: Aliran fluida pada sebuah pipa.

Berdasarkan hukum Bernoulli, maka diperoleh persamaan berikut:

Persamaan tersebut dapat dilakukan transformasi menjadi persamaan berikut:

Data-data terkait saluran tersebut adalah sebagai berikut:

* = volume aliran fluida tiap satuan waktu
* =percepatan gravitasi
* =lebar pipa
* =ketinggian air maksimum
* =tinggi pelebaran pipa
* = ketinggian air

Kita dapat menggunakan pendekatan numerik untuk menentukan . 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:

Untuk penyelesaiannya penulis akan memberikan tebakan awal nilai . 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: 0x000000001c322f00>  
##   
## $root  
## [1] -0.2870309  
##   
## $iter  
## [1] 26

Berdasarkan hasil perhitungan diperoleh nilai atau ketinggian air sekitar dengan jumlah iterasi sebanyak 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.

## 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

## Latihan

1. Temukan akar persamaan dari persamaan non-linier menggunakan metode terbuka dengan dan !
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 dengan rentang pencarian dan !
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))

# Interpolasi dan Ekstrapolasi

Pada dunia nyata, data sering kali tidak tersaji secara lengkap. Seringkali terdapat nilai data yang hilang (*missing value*). Terdapat banyak penyebab dari kondisi tersebut, baik akibat kesalahan manusianya maupun keterbatasan kemampuan alat ukur.

Kondisi lain yang muncul dari data yang kita miliki adalah adanya *outlier* atau nilai yang berbeda jauh dengan mayoritas data yang kita miliki. Nilai tersebut akan menentukan hasil analisis atau uji statistik yang kita lakukan, terlebih lagi jika uji statistik yang kita lakukan menggunakan metode parametrik.

Terdapat banyak cara untuk menangani kondisi-kondisi tersebut. Sejumlah peneliti memilih untuk menghapus data tersebut. Hal ini dapat dilakukan jika jumlah data yang kita miliki cukup besar. Bagaimana jika data yang kita miliki sedikit dan pengukuran ulang cukup mahal atau cukup sulit dilakukan?. Salah satu cara yang dapat dilakukan adalah dengan melakukan interpolasi terhadap data.

Interpolasi dan ekstrapolasi adalah proses “menebak” nilai data dengan memperhatikan data lain yang kita miliki. Interpolasi merupakan teknik untuk mencari nilai suatu variabel yang hilang pada rentang data yang diketahui, sedangkan ektrapolasi merupakan teknik menemukan nilai suatu variabel diluar rentang data yang telah diketahui. Data lain yang kita miliki seringkali memiliki sejumlah pola. Pola yang terbentuk dapat berupa polinomial atau mengelompok. Tiap pola akan memiliki metode pendekatan yang berbeda-beda. Terdapat kemungkinan tak terbatas dari pola data tersebut. Penilaian profesional atau ahli diperlukan untuk menentukan metode mana yang sesuai berdasarkan riwayat penelitian atau pekerjaan yang pernah dilakukan sebelumnya.

Pada Chapter 8 penulis akan menjelaskan teknik-teknik interpolasi yang dapat kita lakukan. Adapun yang akan dibahas pada *Chapter* ini adalah sebagai berikut:

* Teknik interpolasi polinomial
* Teknik interpolasi piecewise
* Studi kasus penerapan teknik interpolasi

## Interpolasi Polinomial

Interpolasi polinomial merupakan teknik interpolasi dengan mengasumsikan pola data yang kita miliki mengikuti pola polinomial baik berderajat satu (linier) maupun berderajat tinggi. Interpolasi dengan metode ini dilakukan dengan terlebih dahulu membentuk persamaan polinomial. Persamaan polinomial yang terbentuk selanjutnya digunakan untuk melakukan interpolasi dari nilai yang diketahui atau ekstrapolasi (prediksi) dari nilai diluar rentang data yang diketahui.

Pada Chapter 8.1 pembahasan akan dibagi menjadi 3 bagian. Bagian pertama kita akan mengulang kembali teknik evaluasi polinomial, sedangkan dua bagian selanjutnya akan membahas teknik interpolasi linier dan polinomial orde tinggi dengan menjadikan pembahasan bagian pertama sebagai dasar pada dua bagian berikutnya.

### Mengevaluasi Polinomial

Pada *Chapter* ini pembaca akan mempelajari teknik untuk melakukan substitusi nilai pada persamaan polinomial untuk memperoleh nilai . Terdapat berbagai pendekatan dalam melakukan proses tersebut, mulai dari metode naive maupun metode Horner. Kedua metode akan menghasilkan hasil yang sama namun dengan proses komputasi yang berbeda. Metode naive cenderung lambat dalam proses komputasi karena jumlah proses yang dilakukan dalam sekali proses lebih banyak dari pada metode Horner.

Untuk memahami metode-metode evaluasi polinomial yang telah disebutkan tersebut, secara umum persamaan polinomial disajikan pada Persamaan (64).

dimana merupakan koefisien polinomial, merupakan variabel, dan merupakan indeks dan pangkat polinomial.

Pada metode naive kita melakukan evaluasi polinomial sama dengan cara kita melakukan evaluasi polinomial saat kita SMA. Nilai akan disubstitusikan pada masing-masing elemen persamaan polinomial. Masing-masing elemen polinomial selanjutnya dijumkahkan untuk menghitung .

Pada R kita dapat menuliskan sebuah fungsi untuk melakukan evaluasi polinomial menggunakan metode naive tersebut. Pada fungsi tersebut, koefisien polinomial akan disimpan kedalam sebuah vektor dengan urutan pengisian mulai dari koefisien dengan pangkat terendah ke tertinggi.

naive\_poly <- function(x, coeff){  
 n <- length(x)  
 y <- rep(0, n)  
   
 for(i in 1:length(coeff)){  
 y <- y + coeff[i]\*(x^(i-1))  
 }  
   
 return(y)  
}

Contoh 19: Hitung nilai pada persamaan , jika diketahui nilai adalah -1, 0, dan 1!

**Jawab**:

Untuk dapat menghitung nilai menggunakan fungsi naive\_poly() pada persamaan tersebut dengan nilai yang diketahui, kita perlu merubah koefisien persamaan tersebut dan nilai yang diketahui menjadi vektor:

x <- c(-1,0,1)  
coeff <- c(30,-19,-15,3,1)

Masukkan vektor-vektor yang telah terbentuk tersebut kedalam fungsi naive\_poly().

naive\_poly(x, coeff)

## [1] 32 30 0

Berdasarkan hasil perhitungan diperoleh nilai masing-masing sebesar 32, 30, dan 0. Pembaca dapat mengeceknya sendiri hasil perhitungan tersebut menggunakan cara manual.

Kita dapat meningkatkan efisiensi proses perhitungan pada fungsi naive\_poly() tersebut. Sebagai contoh, setiap kali kita melakukan loop untuk menghitung nilai pada polinomial, kita dapat memperoleh eksponensial dari . Namun untuk setiap koefisien , nilai eksponensial yang terkait merupakan seri produk.

Untuk , terdapat sebanyak koefisien dikalikan bersamaan. Namun, untuk setiap terdapat lebih sedikit 1 perkalian dibanding koefisien dengan pangkat yang lebih besar dan seterusnya.

Berdasarkan ilustrasi tersebut, kita dapat membentuk fungsi better\_poly() sebagai perbaikan dari fungsi naive\_poly(). Berikut adalah sintaks yang digunakan:

better\_poly <- function(x, coeff){  
 n <- length(x)  
 y <- rep(0, n)  
 cached\_x <- 1  
   
 for(i in 1:length(coeff)){  
 y <- y + coeff[i]\*cached\_x  
 cached\_x <- cached\_x \* x  
 }  
 return(y)  
}

Contoh 20: Hitung nilai pada persamaan yang disajikan pada Contoh 19 menggunakan nilai yang telah diketahui pada soal tersebut menggunakan fungsi better\_poly()?

**Jawab**:

better\_poly(x, coeff)

## [1] 32 30 0

Sejauh ini kita telah membentuk 2 fungsi yaitu naive\_poly() dan better\_poly(). Kedua fungsi tersebut memiliki perbedaan proses menghitung yang mempengaruhi efisiensinya masing-masing. Sebagai contoh jika diberikan polinomial berderajat 10, fungsi naive\_poly() akan mengejakan perkalian dalam proses *loop* sebanyak 55 kali, sedangkan fungsi better\_poly() akan melakukannya sebanyak 20 kali ( perkalian).

Metode lain yang lebih efisien dalam melakukan evaluasi polinomial adalah metode Horner. Metode ini oleh William Horner pada abad ke-18. Dalam metode Horner, bentuk polinomial pada Persamaan (64) akan ditransformasi menjadi Persamaan (66).

Berdasarkan Persamaan (66), jika kita melakukan perhitungan pada persamaan polinomial berderajat 10, kita dapat mereduksi perhitungan menjadi 10 perkalian dan 10 penjumlahan. Jumlah tersebut sangat kecil dibandingkan kedua metode sebelumnya dan dapat dikatakan lebih efisien dibandingkan metode lainnya.

FUngsi horner\_poly() merupakan fungsi yang dibentuk bedasarkan Persamaan (66). Sintaks fungsi tersebut adalah sebagai berikut:

horner\_poly <- function(x, coeff){  
 n <- length(x)  
 y <- rep(0, n)  
   
 for(i in length(coeff):1){  
 y <- coeff[i] + x \* y  
 }  
 return(y)  
}

Contoh 21: Kerjakan kembali Contoh 20 fungsi horner\_poly()?

**Jawab**:

horner\_poly(x, coeff)

## [1] 32 30 0

### Interpolasi Linier

Misalkan kita memiliki 3 buah data dengan dua buah variabel kita misalkan variabel dan variabel . Pada salah satu data terdapat data yang hilang pada. Agar ketiga data tersebut tetap dapat digunakan dalam iterasi diperlukan interpolasi untuk “menebak” nilai dari data yang hilang.

Berdasarkan pengukuran yang sebelumnya pernah dilakukan diketahui bahwa pola data yang terbentuk variabel dan divisualisasikan menggunakan scatterplot adalah pola linier. Berdasarkan hal tersebut interpolasi dilakukan dengan menggunakan metode linier.

Interpolasi linier dilakukan dengan terlebih dahulu membentuk fungsi linier. Dengan kata lain kita perlu mencari nilai slope dan *intercept* . Nilai dihitung sebagai rasio selisih jarak dua titik pada sumbu dan sumbu yang dapat dituliskan melalui Persamaan (67).

Nilai *intercept* (titik potong pada sumbu ) dihitung menggunakan Persamaan (68).

**Algoritma Interpolasi Linier**

1. Tentukan dua buah titik sebagai dasar pembentukan persaman linier.
2. Hitung menggunakan Persamaan (67)
3. Hitung menggunakan Persamaan (68)
4. Definiskan fungsi linier berdasarkan nilai dan
5. Hitung dengan cara substitusi nilai pada persamaan linier untuk melakukan interpolasi atau ekstrapolasi nilai yang ingin dicari.

Algoritma poin 1 sampai 4 tersebut, kita dapat membentuk fungsi pembentuk persamaan linier dari 2 titik yang diketahui. Fungsi tersebut disajikan pada sintaks berikut:

linear\_inter <- function(x, y){  
 m <- (y[2]-y[1]) / (x[2]-x[1])  
 b <- y[2] - m\*x[2]  
   
 return(c(b, m))  
}

Contoh 22: Diketahui koordinat 2 buah titik yaitu (0,-1) dan (2,3). Jika diketahui titik ketiga memiliki koordinat sumbu sebesar 1. Lakukan interpolasi untuk menentukan koordinat sumbu titik ketiga tersebut!

**Jawab**:

Berdasarkan data-data yang terdapat pada soal terserbut, kita dapat menghitung nilai dan . Nilai dapat dihitung sebagai berikut:

Dengan menggunakan nilai tersebut kita dapat menghitung nilai .

Berdasarkan hasil perhitungan diperoleh persamaan linier yang terbentuk adalah sebagai berikut:

Berdasarkan persamaan linier tersebut nilai dapat dihitung.

Kita dapat pula membentuk persamaan linier menggunakan fungsi linear\_inter(). Berikut adalah sintaks yang digunakan:

x <- c(0,2)  
y <- c(-1,3)  
  
(coeff <- linear\_inter(x,y))

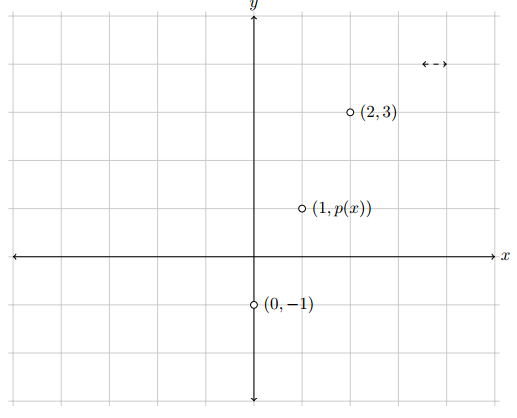
## [1] -1 2

Setelah diperoleh koefisien persamaan linier berdasarkan dua titik tersebut, kita akan menggunakan fungsi horner\_poly() untuk memperoleh nilai . Berikut adalah sintaks yang digunakan:

horner\_poly(1, coeff)

## [1] 1

Hasil interpolasi yang diperoleh dapat dikatakan sesuai dengan lokasi kedua titik data yang ditunjukkan pada Gambar 60. Berdasarkan hal tersebut, kita dapat yakin bahwa hasil interpolasi yang telah kita lakukan telah sesuai.



(#fig:linviz)Interpolasi linier dua titik (Sumber:Howard, 2017).

Metode interpolasi linier dapat dibilang merupakan metode interpolasi yang sangat sederhana. Disamping kemudahannya, metode ini memiliki potensi error numerik jika jarak antara kedua titik cukup berdekatan terlebih lagi jika selisih penyebut () sangat kecil sehingga akan menghasilkan nilai yang sangat besar.

Disamping adanya potensi error numerik tersebut, metode ini menjadi dasar bagi metode interpolasi lain yang lebih kompleks. Metode selanjutnya merupakan pengembangan dari metode interpolasi ini.

### Interpolasi Polinomial Orde Tinggi

Dengan menggunakan dua titik, kita dapat membentuk garis lurus (linier) yang tepat pada dua titik tersebut. Masalah timbul jika selisih nilai kedua titik tersebut sangat kecil atau kedua titik tersebut memiliki nilai yang sama. Hal ini akan menyebabkan slope yang dihasilkan menjadi tidak terhingga atau garis yang terbentuk adalah garis vertikal tegak lurus.

Bagaimana jika terdapat tiga buah titik? apakah kita masih bisa menggunakan interpolasi linie?. Ya, asalkan ketiga titik tersebut membentuk pola linier atau terletak pada satu garis yang sama. Pada kenyatannya kondisi tersebut jarang terjadi, sehingga pendekatan menggunakan polinomial orde lebih tinggi diperlukan. Persamaan kuadratik (polinomial orde dua) dapat digunakan untuk membentuk persamaan polinomial pada ketiga titik tersebut, sehingga iterasi dapat dilakukan. Untuk 4 buah titik data, polinomial orde tiga dapat digunakan untuk melakukan interpolasi. Secara umum berdasarkan penjelasan tersebut, untuk titik data interpolasi dapat dilakukan menggunakan persamaan polinomial orde .

Diberikan set data berpasangan yang telah diurutkan , fungsi interpolasi harus memenuhi persyaratan berikut:

Untuk setiap . Sebagai tambahan, fungsi interpolasi berupa fungsi polinomial dengan bentuk umum sebagai berikut:

Persamaan (70) dapat dituliskan kedalam bentuk matriks yang ditampilkan pada Persamaan (71).

Persamaan matrik tersebut dapat dituliskan sebagai . Untuk menyelesaikan persamaan tersebut (memperoleh nilai ), pembaca dapat membaca kembali Chapter 6. Matriks disebut sebagai matriks Vandemonde dan matriks tersebut mengandung sejumlah nilai dengan pangkat sampai dengan .

**Algoritma Interpolasi Polinomial Orde Tinggi**

1. Tentukan set titik berpasangan yang telah diurutkan.
2. Bentuk matriks Vandermonde sesuai dengan Persamaan (71).
3. Definiskan persamaan matriks
4. Selesaikan persamaan matriks pada poin 3 untuk memperoleh nilai
5. Definisikan persamaan polinomial berdasarkan koefisien yang diperoleh
6. Lakukan substitusi persamaan polinomial pada poin 5 untuk memperoleh nilai

Berdasarkan algoritma poin 1 sampai 5, kita dapat membentuk suatu fungsi untuk membentuk persamaan polinomial berdasarkan Persamaan (71). Berikut adalah sintaks fungsi tersebut:

poly\_inter <- function(x, y){  
 if(length(x) != length(y))  
 stop("Lenght of x and y vectors must be the same")  
   
 n <- length(x)-1  
 vandermonde <- rep(1, length(x))  
 for(i in 1:n){  
 xi <- x^i  
 vandermonde <- cbind(vandermonde, xi)  
 }  
 beta <- solve(vandermonde, y)  
   
 names(beta) <- NULL  
 return(beta)  
}

Contoh 23: Diketahui koordinat 3 buah titik yaitu (-1,-2), (1,2) dan (0,1). Jika diketahui titik keempat memiliki koordinat sumbu sebesar -2. Lakukan interpolasi untuk menentukan koordinat sumbu titik keempat tersebut!

**Jawab**:

Untuk menyelesaiakn contoh soal tersebut, kita perlu terlebih dahulu membentuk matriks sesuai dengan Persamaan (71). Berdasarkan soal tersebut, terdapat tiga buah titik data yang diketahui, sehingga polinomial yang hendak dibentuk selanjutnya adalah polinomial berderajat 2.

Setelah matriks tersebut terbentuk, pembaca dapat menyelesaikannya menggunakan berbagai metode yang telah penulis jelaskan pada Chapter 6 untuk memperoleh nilai .

Untuk menyelesaikan contoh soal tersebut pada R, kiat perlu membentuk matriks dan terlebih dahulu.

x <- c(-1, 1, 0)  
y <- c(-2, 2, -1)

Koefisien persamaan polinomial dihitung menggunakan fungsi poly\_inter().

(coeff <- poly\_inter(x, y))

## [1] -1 2 1

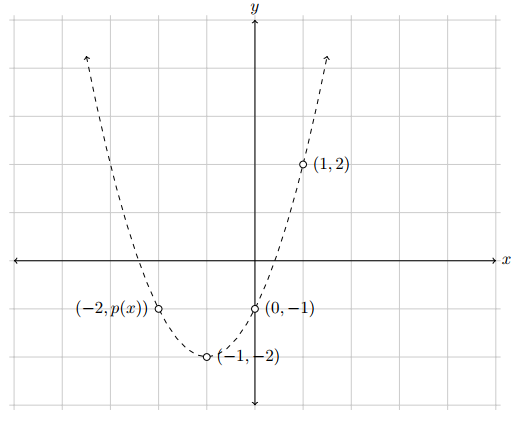
Berdasarkan hasil perhitungan, diperoleh nilai . Nilai tersebut selanjutnya digunakan untuk membentuk persamaan polinomial. Berikut merupakan persamaan polinomial yang terbentuk:

Fungsi horner\_poly() selanjutnya digunakan untuk mengevaluasi polinomial tersebut. Berikut adalah hasil substitusi pada persamaan tersebut:

horner\_poly(-2, coeff)

## [1] -1

Hasil yang diperoleh terlihat cukup sesuai jika kita perhatikan visualisasi ketiga titik tersebut pada Gambar 61.



(#fig:hopoliviz)Interpolasi kuadratik tiga titik (Sumber:Howard, 2017).

Contoh 24: Dengan menggunakan data pada Contoh 22, lakukan proses perhitungan untuk membentuk persamaan polinomial menggunakan fungsi poly\_inter()

**Jawab**:

Berdasarkan data pada Contoh 22, polinomial yang terbentuk merupakan polinomial derajat 1 (linier). Berikut adalah nilai koefisien yang dihasilkan dari perhitungan menggunakan fungsi poly\_inter().

x <- c(2, 0)  
y <- c(3, -1)  
poly\_inter(x, y)

## [1] -1 2

Meskipun proses perhitungan menggunakan fungsi poly\_inter() lebih rumit dibandingkan dengan fungsi linear\_poly(), hasil perhitungan keduanya menggunakan data pada Contoh 22 menghasilkan hasil yang sama.

## Interpolasi Piecewise

Interpolasi dengan polinomial sering memberikan hasil yang tidak dapat diterima. Interpolasi polinomial yang dihasilkan dari sejumlah besar data titik biasanya berderajat tinggi. Polinomial berderajat tinggi pada umumnya bersifat osilatif (grafiknya naik turun secara cepat). Akibatnya, perubahan data pada interval kecil dapat menyebabkan fluktuasi besar pada keseluruhan interval. Karena alasan ini, biasanya interpolasi hanya menggunakan polinomial berderajat rendah.

Interpolasi piecewise menawarkan alternatif lain. Pada interpolasi piecewise, pada titik yang berbeda sepanjang kurva, nilai fungsi lebih mungkin lebih baik didekati menggunakan dua atau lebih interpolasi. Pada metode ini kita akan membuat fungsi interpolasi ditiap antara dua titik observasi.

Pada sub-Chapter ini akan dijelaskan 2 buah metode interpolasi piecewise, yaitu: interpolasi linier piecewise dan interpolasi kubik spline. Interpolasi pertama dilakukan menggunakan persamaan linier, sehingga kurva yang terbentuk bukan merupakan kurva kontinu. Interpolasi selanjutnya dilakukan menggunakan persamaan polinomial berderajat tinggi sehingga kurva yang dihasilkan lebih halus (tidak ada sudut siku pada setiap titik).

### Interpolasi Linier Piecewise

Interpolasi linier piecewise merupakan interpolasi yang menggunakan pendekatan interpolasi linier. Fungsi linier akan dibentuk pada setiap dua titik observasi. Untuk lebih memahaminya perhatikan kembali Gambar 61. Pada gambar tersebut sebelumnya kita telah membentuk persamaan kudratik untuk menghubungkan titik-titik tersebut. Dibanding menggunakan persamaan polinomial seperti kuadratik tersebut, interpolasi piecewise akan menghubungkan tiap dua titik observasi tersebut dengan garis lurus.

**Algoritma Interpolasi Linier Piecewise**

1. Tentukan set titik berpasangan yang telah diurutkan berdasarkan nilai sumbu .
2. Hitung pada setiap dua titik berdekatan menggunakan Persamaan (67)
3. Hitung pada setiap dua titik berdekatan menggunakan Persamaan (68)
4. Definiskan fungsi linier berdasarkan nilai dan
5. Hitung dengan cara substitusi nilai pada persamaan linier untuk melakukan interpolasi nilai yang ingin dicari.
6. Untuk melakukan ekstrapolasi dengan titik observasi diluar rentang titik diketahui, gunakan persamaan linier yang berada pada bagian ujung terdekat dengan nilai yang hendak dicari nilai -nya.

Berdasarkan algoritma tersebut, kita dapat menyusun fungsi pada R untuk membentuk persamaan linier piecewise. Berikut adalah sintaks yang digunakan:

pwise\_linterp <- function(x, y){  
 n <- length(x)-1  
   
 y <- y[order(x)]  
 x <- x[order(x)]  
   
 m\_vec <- b\_vec <- c()  
   
 for(i in 1:n){  
 m <- (y[i+1]-y[i]) / (x[i+1]-x[i])  
 b <- y[i+1] - m\*x[i+1]  
 m\_vec <- c(m\_vec, m)  
 b\_vec <- c(b\_vec, b)  
 }  
   
 return(list(b = b\_vec, m = m\_vec))  
}

Contoh 25: Tentukan persamaan-persamaan linier yang dihasilkan dari titik observasi yang ditampilkan pada Contoh 23?

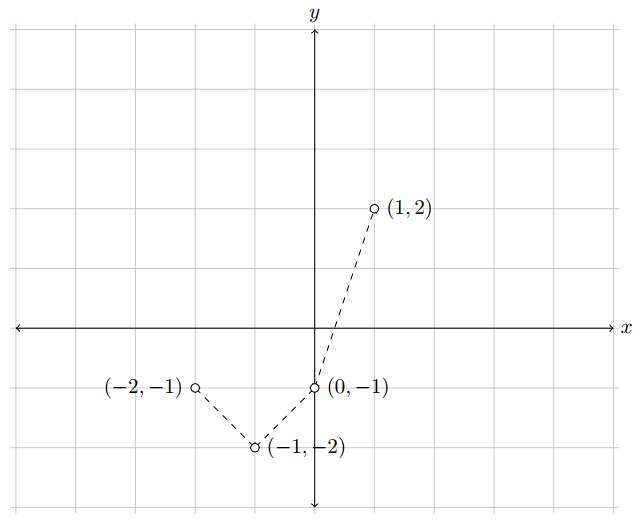
**Jawab**:

Untuk menentukan persamaan-persamaan linier yang menghubungkan setiap titik, kita akan menggunakan fungsi pwise\_linterp() yang telah kita buat. Berikut adalah sintaks yang digunakan:

x <- c(-1, 1, 0, -2)  
y <- c(-2, 2, -1, -1)  
  
pwise\_linterp(x, y)

## $b  
## [1] -3 -1 -1  
##   
## $m  
## [1] -1 1 3

Jika persamaan-persamaan yang terbentuk tersebut divisualisasikan akan terlihat seperti pada Gambar 62.



(#fig:pwiselinviz)Interpolasi linier piecewise empat titik (Sumber:Howard, 2017).

R juga menyediakan fungsi untuk melakukan interpolasi linier piecewise. Fungsi approxfun() dapat digunakan untuk melakukan interpolasi tersebut. Fungsi approxfun() hanya memerlukan dua input yaitu vektor dan vektor . Untuk lebih memahaminya, kita akan menggunakan kembali data yang disajikan pada Contoh 23 untuk membuat fungsi linier piecewise. Berikut adalah sintaks yang digunakan:

f <- approxfun(x, y)  
  
# tentukan nilai y jika x= 0  
f(0)

## [1] -1

# tentukan nilai y jika x = 0.5  
f(0.5)

## [1] 0.5

### Interpolasi Spline Kubik

Jika menggunakan interpolasi polinomial berderajat satu (sebuah garis) lebih dari beberapa interval merupakan peningkatan dari satu baris interpolasi, dan jika menggunakan polinomial berderajat tinggi juga merupakan peningkatan dari satu garis interpolasi tunggal, maka dapat disimpulkan bahwa penggunaan polinomial berderajat tinggi pada selang beberapa interval juga akan menjadi peningkatan dalam proses interpolasi. Dalam beberapa kasus, hal tersebut benar, tetapi kita masih menghadapi sudut tajam di mana masing-masing kurva interpolasi tergabung (interpolasi linier piecewise). Sudut tajam ini mencegah diferensiasi dan pada prakteknya tidak dapat digunakan untuk memodelkan beberapa fungsi di dunia nyata, seperti *roller coaster span*.

Interpolasi spline kubik memecahkan masalah ini. Interpolasi ini akan memberikan kurva tergabung yang halus. Hal tersebut juga membuat spline terintegrasi. Karena setiap bagian individu diwakili oleh kurva kubik (polinomial derajat 3), maka masing-masing bagian individu juga dapat dianalisis sebagai kurva kubik. Dengan asumsi ada titik data untuk interpolasi, kita akan mendefinisikan sebagai fungsi polinomial kubik yang mewakili kurva pada domain . Kemudian untuk titik observasi, ada interpolasi polinomial kubik.

Bentuk umum seri polinomial dituliskan pada Persamaan (72).

Ini mengarah ke 4 nilai yang tidak diketahui tidak diketahui, yaitu: , , , dan pada setiap Persamaan (72). Oleh karena itu, ada nilai yang tidak diketahui. Karena kita ingin spline membentuk garis kontinu dan dapat didiferensiasi, ada satu set persamaan yang menentukan spline kubik:

Persamaan (73) dan (74) sudah cukup jelas. Persyaratan ini memastikan bahwa jika kita mengevaluasi spline di salah satu node internal, hasil yang akan kita peroleh merupakan jawaban yang telah ditentukan, yaitu spline yang dievaluasi pada untuk beberapa adalah , dan setiap komponen spline bergabung dengan rapi. Persamaan (75) memastikan bahwa kita memiliki turunan pertama yang berkelanjutan di setiap simpul internal. Ini mencegah terbentuknya sudut tajam pada tiap node. Persamaan (76) memastikan turunan kedua juga kontinu, dimana kondisi ini menguntungkan karena itu berarti turunan pertama itu sendiri dapat didiferensiasi juga.

Kondisi ini menyebabkan ada kondisi yang harus kita penuhi. Jika kita yang tidak diketahui dipecahkan sebagai sebuah matriks, dan akhirnya matriks tersebut akan terpecahkan, matriks akan menjadi kurang ditentukan. Kita dapat menyelesaikan kondisi tersebut dengan memasukkan dua ketentuan tambahan. Dengan splines kubik, secara normal adalah menentukan akhir di kedua ujung untuk mencapai dua kondisi tambahan. Untuk contoh ini, dua kondisi yang akan kita tambahkan adalah dan . Kedua kondisi ini memastikan bahwa pada titik akhir, turunan pertamanya linier dan oleh karena itu fungsi spline berlanjut ke arah yang sudah berjalan. Interpolasi spline kubik ini disebut juga sebagai “natural spline”.

Awalnya, kita dapat melihat bahwa setiap polinomial kubik digeser ke kanan oleh unit atau bergeser ke kiri jika negatif. Pada nilai fungsi adalah yang berarti untuk setiap nilai . Menyelesaikan sisa koefisien yang ada akan lebih kompleks, tetapi sekarang tidak diketahui dengan kondisi. Derivasi penuh tersedia dari berbagai sumber, tetapi secara garis besar dari Persamaan (75) kita mendapatkan , kemudian , dan kita dapat mensubstitusi Persamaan (72) ke dalam kedua komponen, pemecahan untuk dalam hal ini , , dan . Proses yang sama dapat direplikasi dengan Persamaan (76) dan . Hasilnya adalah matriks tridiagonal, seperti yang ada pada Chapter 6.3.3, yang dapat dipecahkan untuk menemukan koefisien. Terdapat sebuah matriks, A, sedemikian rupa sehingga,

dimana merupakan matriks tridiagonal. Pada matriks tridiagonal ini merupakan vektor , dan vektor , , dan ditentukan dengan cara serupa. Matriks ini sedemikian rupa,

kecuali pada . Lebih jauh, diagonal utama ditentukan menggunakan Persamaan (79).

kecuali pada . Akhirnya vektor ditentukan dengan Persamaan (80).

kecuali pada . Sehingga,

Penyelesaian Persamaan (81) akan menghasilkan vektor koefisien , sehingga koefisien dapat dihitung menggunakan Persamaan (82).

Dengan menggunakan vektor yang sudah diketahui, koefisien dapat dihitung menggunakan Persamaan (83).

**Algoritma Interpolasi Spline Kubik**

1. Tentukan set titik berpasangan .
2. Tentukan koefisien menggunakan Persamaan (73), dimana .
3. Hitung elemen diagonal bawah () dan elemen diagonal atas () matriks tridiagonal menggunakan Persamaan (78), dimana .
4. Hitung elemen diagonal utama () menggunakan Persamaan (79), dimana .
5. Hitung elemen vektor menggunakan Persamaan (80), dimana .
6. Susunlah elemen , , dan menjadi matriks tridiagonal .
7. Deifinisikan persamaan linier seperti pada Persamaan (77)
8. Selesaikan sistem persamaan linier pada Persamaan (77) sehingga diperoleh vektor yang merupakan kumpulan koefisien .
9. Hitung koefisien menggunakan Persamaan (82)
10. Hitung koefisien menggunakan Persamaan (83).
11. Bentuk seri persamaan polinomial menggunakan elemen koefisien , , , dan yang telah dihitung.
12. Untuk melakukan ekstrapolasi dengan titik observasi diluar rentang titik diketahui, gunakan persamaan polinomial yang berada pada bagian ujung terdekat dengan nilai yang hendak dicari nilai -nya.

Berdasarkan algoritma tersebut, kita dapat menyusun suatu fungsi pada R untuk mencari seri persamaan polinomial derajat tiga. Fungsi tersebut adalah sebagai berikut:

cubic\_spline <- function(x, y){  
 n <- length(x)  
 d\_vec <- b\_vec <- a\_vec <- rep(0, n-1)  
 vec <- rep(0, n)  
 delta\_x <- delta\_y <- rep(0, n-1)  
   
 ## Menghitung nilai selisih dan vektor A  
 for(i in 1:(n-1)){  
 a\_vec[i] <- y[i]  
 delta\_x[i] <- x[i+1] - x[i]  
 delta\_y[i] <- y[i+1] - y[i]  
 }  
   
 ## Menyusun matriks tridiagona  
 Au <- c(0, delta\_x[2:(n-1)])  
 Am <- c(1, 2\*(delta\_x[1:(n-2)]+delta\_x[2:(n-1)]), 1)  
 Al <- c(delta\_x[1:(n-2)], 0)  
   
 vec[0] <- vec[n] <- 0  
 for(i in 2:(n-1))  
 vec[i] <- 3 \* (delta\_y[i]/delta\_x[i] -   
 delta\_y[i-1]/delta\_x[i-1])  
   
 ## penyelesaian tridiagonal matriks  
 nm <- length(Am)  
 Al <- c(NA , Al)  
   
 ### forward sweep  
 Au[1] <- Au[1] / Am[1]  
 vec[1] <- vec[1] / Am[1]  
 for(i in 2:(n - 1)){  
 Au[i] <- Au[i] / (Am[i] - Al[i] \* Au[i - 1])  
 vec[i] <- (vec[i] - Al[i] \* vec[i - 1]) /  
 (Am[i] - Al[i] \* Au[i - 1])  
 }  
 vec[n] <- (vec[n] - Al[n] \* vec[n - 1])/  
 (Am[n] - Al[n] \* Au[n - 1])  
   
 ### backward sweep  
 c\_vec <- rep.int (0, n)  
 c\_vec[n] <- vec[n]  
 for(i in (n - 1) :1)  
 c\_vec[i] <- vec[i] - Au[i] \* c\_vec[i + 1]  
   
 ## Hitung vektor B dan D dari vektor C  
 for(i in 1:(n-1)){  
 b\_vec[i] <- (delta\_y[i]/delta\_x[i])-  
 (delta\_x[i]/3)\*(2\*c\_vec[i]+c\_vec[i+1])  
 d\_vec[i] <- (c\_vec[i+1]-c\_vec[i]) / (3\*delta\_x[i])  
 }  
   
 return(list(a = a\_vec, b = b\_vec, c = c\_vec[-n], d = d\_vec))  
}

Contoh 26: Tentukan persamaan-persamaan spline kubik yang dihasilkan dari titik observasi yang ditampilkan pada Contoh 23?

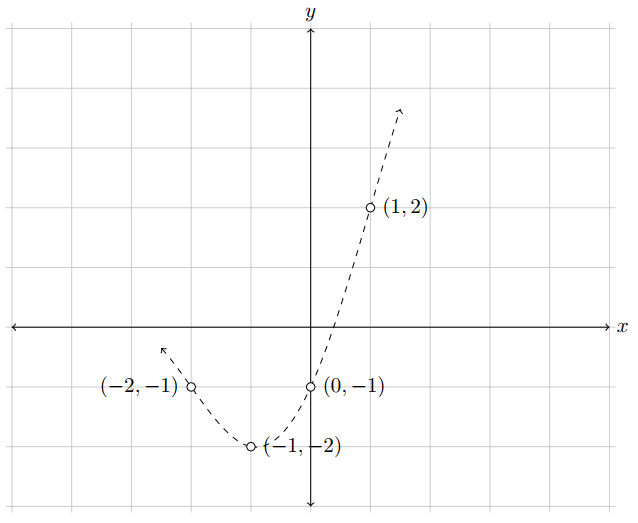
**Jawab**:

Untuk menentukan persamaan-persamaan linier yang menghubungkan setiap titik, kita akan menggunakan fungsi cubic\_spline() yang telah kita buat. Berikut adalah sintaks yang digunakan:

x <- c(-2, -1, 0, 1)  
y <- c(-1, -2, -1, 2)  
  
cubic\_spline(x, y)

## $a  
## [1] -1 -2 -1  
##   
## $b  
## [1] -1.4 -0.2 2.2  
##   
## $c  
## [1] 0.0 1.2 1.2  
##   
## $d  
## [1] 0.4 0.0 -0.4

Sebagai contoh untuk domain , persamaan spline kubiknya adalah . Jika persamaan-persamaan yang terbentuk tersebut divisualisasikan akan terlihat seperti pada Gambar 63.



(#fig:cubicsplineviz)Interpolasi spline kubik empat titik (Sumber:Howard, 2017).

Pada R juga terdapat fungsi splinefun() untuk melakukan interpolasi spline. Format fungsi tersebut adalah sebagai berikut:

splinefun(x, y = NULL,  
 method = c("fmm", "periodic", "natural", "monoH.FC", "hyman"),  
 ties = mean)

**Catatan:**

* x,y : vektor titik yang akan dilakukan interpolasi
* method : spesifikasi jenis spline yang digunakan. Nilai yang mungkin antara lain: "fmm", "natural", "periodic", "monoH.FC" dan "hyman".
* ties : metode untuk menangani nilai imbang pada titik yang akan diinterpolasi.

Untuk melakukan interpolasi spline kubik, metode yang digunakan adalah "natural". Berikut adalah contoh interpolasi menggunakan kembali data pada Contoh 23:

spline <- splinefun(x, y, method="natural")  
  
# uji dengan nilai x yang sama  
spline(x)

## [1] -1 -2 -1 2

FUngsi splinefun() menghasilkan nilai yang sama persis dengan proses interpolasi yang telah kita lakukan.

## Studi Kasus

Pada studi kasus kali ini, kita akan membahas teknik mengisi nilai yang hilang pada data runtun waktu. Terdapat banyak teknik untuk yang dapat digunakan untuk mengisi data yang hilang. Salah satu teknik yang dapat digunakan adalah dengan melakukan interpolasi.

### Interpolasi Data Runtun Waktu

Pengisian data hilang pada data runtun waktu (*time series*) dapat dilakukan dengan berbagai cara sesuai dengan situasi yang dihadapi. Pengisian dapat menggunakan nilai rata-rata jika data memiliki pola *white noise*, observasi terakhir atau observasi dimasa mendatang, dan interpolasi linier.

Data yang digunakan pada contoh kasus kali ini adalah data airquality. Dataset tersebut merupakan data kualitas udara bulan Mei sampai September 1973 yang ada di New York. Berikut adalah ringkasan data airquality tersebut:

summary(airquality)

## Ozone Solar.R Wind Temp   
## Min. : 1.00 Min. : 7.0 Min. : 1.700 Min. :56.00   
## 1st Qu.: 18.00 1st Qu.:115.8 1st Qu.: 7.400 1st Qu.:72.00   
## Median : 31.50 Median :205.0 Median : 9.700 Median :79.00   
## Mean : 42.13 Mean :185.9 Mean : 9.958 Mean :77.88   
## 3rd Qu.: 63.25 3rd Qu.:258.8 3rd Qu.:11.500 3rd Qu.:85.00   
## Max. :168.00 Max. :334.0 Max. :20.700 Max. :97.00   
## NA's :37 NA's :7   
## Month Day   
## Min. :5.000 Min. : 1.0   
## 1st Qu.:6.000 1st Qu.: 8.0   
## Median :7.000 Median :16.0   
## Mean :6.993 Mean :15.8   
## 3rd Qu.:8.000 3rd Qu.:23.0   
## Max. :9.000 Max. :31.0   
##

str(airquality)

## 'data.frame': 153 obs. of 6 variables:  
## $ Ozone : int 41 36 12 18 NA 28 23 19 8 NA ...  
## $ Solar.R: int 190 118 149 313 NA NA 299 99 19 194 ...  
## $ Wind : num 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...  
## $ Temp : int 67 72 74 62 56 66 65 59 61 69 ...  
## $ Month : int 5 5 5 5 5 5 5 5 5 5 ...  
## $ Day : int 1 2 3 4 5 6 7 8 9 10 ...

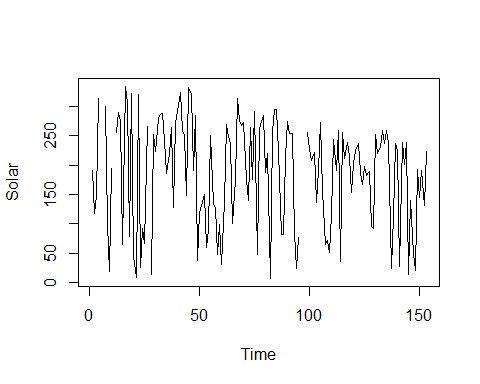
head(airquality)

## Ozone Solar.R Wind Temp Month Day  
## 1 41 190 7.4 67 5 1  
## 2 36 118 8.0 72 5 2  
## 3 12 149 12.6 74 5 3  
## 4 18 313 11.5 62 5 4  
## 5 NA NA 14.3 56 5 5  
## 6 28 NA 14.9 66 5 6

Pada contoh kasus kali ini kita akan mencoba melakukan pengisian data hilang pada data Solar.R pada dataset airquality. Langkah pertama yang perlu dilakukan adalah membuat objek data runtun waktu pada data tersebut.

Solar <- ts(airquality[,"Solar.R"])

Visualisasi data tersebut disajikan pada Gambar 64.

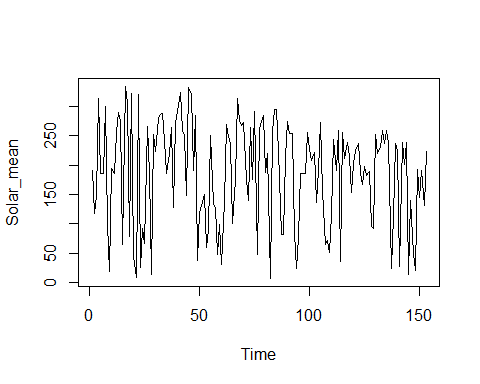


Gambar 64: Visualisasi variabel radiasi matahari pada dataset airquality.

Berdasarkan visualisasi tersebut terdapat garis yang terputus yang menunjukkan data yang hilang. Agar garis tersebut dapat tersambung, kita perlu melakukan pengisian nilai yang hilang pada data tersebut. Berikut adalah sintaks yang digunakan untuk melakukan pengisian nilai hilang tersebut:

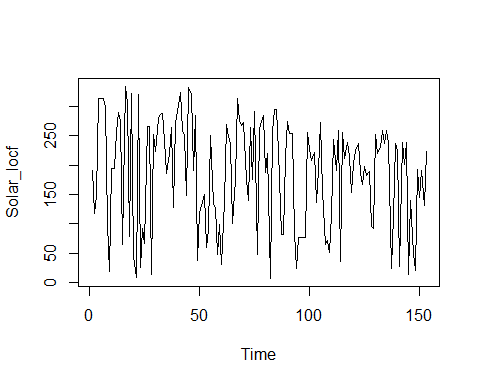
library(xts)  
# metode nilai rata-rata  
Solar\_mean <- na.fill(Solar, fill=mean(Solar, na.rm=TRUE))  
  
# metode last observation carried forward  
Solar\_locf <- na.locf(Solar)  
  
# metode next observation caried backward  
Solar\_nocb <- na.locf(Solar, fromLast = TRUE)  
  
# metode interpolasi linier  
Solar\_linterp <- na.approx(Solar)

Berikut adalah visualisasi menggunakan metode nilai rata-rata:

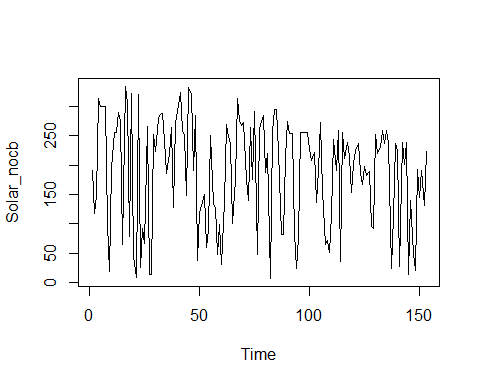


Gambar 65: Visualisasi data radiasi matahari menggunakan metode nilai rata-rata.

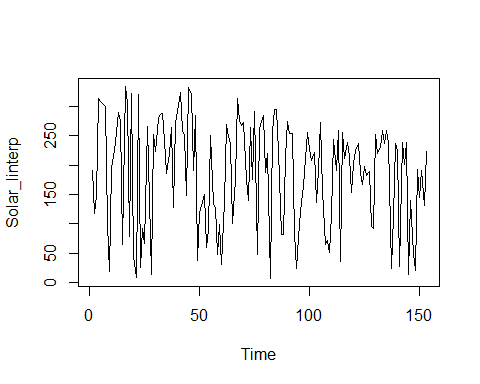
Secara berturut-turut berikut adalah visualisasi dari metode locf, nocb, dan interpolasi linier:



Gambar 66: Visualisasi data radiasi matahari menggunakan metode locf.



Gambar 67: Visualisasi data radiasi matahari menggunakan metode nocb.



Gambar 68: Visualisasi data radiasi matahari menggunakan metode interpolasi linier.

Pemilihan metode interpolasi mana yang sesuai akan berbeda pada setiap situasi dan jenis data yang akan dilakukan interpolasi. Interpolasi data runtun waktu pada bidang lingkungan umumnya menggunakan metode interpolasi nilai rata-rata dan linier dan tidak menutup kemungkinan interpolasi dengan metode lain yang telah dijelaskan pada buku ini dapat pula digunakan.

Pada situasi dimana data membentuk pola *white noises* (pola acak disekitar nilai rata-rata dan memiliki varians yang konstan) seperti yang ditunjukkan variabel Solar.R, interpolaasi dengan nilai rata-rata cukup sesuai untuk digunakan untuk mengisi nilai hilang (*missing value*) pada data runtun waktu tersebut.

## Referensi

1. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
2. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
3. Suparno, S. 2008. **Komputasi untuk Sains dan Teknik Edisi II**. Departemen Fisika-FMIPA Universitas Indonesia.

## Latihan

1. Diberikan data titik (3,5), (0,-2), dan (4,1). Tentukan persamaan polinomial untuk melakukan interpolasi pada ketiga titik tersebut!
2. Jika diberikan 13 titik observasi, apakah saudara cenderung akan menggunakan interpolasi polinomial atau spline? jelaskan alasan saudara?
3. Bentuklah kembali fungsi poly\_inter() dengan menambahkan metode evaluasi polinomial ke dalam fungsi tersebut!
4. Pada Latihan No.1, bentuklah persamaan spline kubik menggunakan titik observasi tersebut!

# Diferensiasi dan Integrasi Numerik

Pada Chapter 9, penulis akan menjabarkan mengenai metode numerik untuk melakukan diferensiasi dan integrasi pada suatu fungsi. Adapun yang akan dibahas pada *Chapter* ini antara lain:

* Metode Beda Hingga
* Metode Integrasi Newton-Cotes
* Metode Integrasi Kudratur Gauss
* Metode Integrasi Adaptif
* Metode Integrasi Romberg
* Metode Integrasi Monte Carlo
* Studi Kasus

## Metode Beda Hingga

Diferensiasi merupakan proses mencari slope suatu garis pada titik yang diberikan. Secara umum proses diferensiasi dinyatakan melalui Persamaan (84).

Kita dapat menyatakan secara formal proses diferensiasi sebagai limit Persamaan (84) dimana mendekati nol. Jadi kita ingin membuat nilai sekecil mungkin untuk memperoleh pendekatan terbaik terhadap nilai turunan suatu fungsi. Kita membatasi nilai pada sejumlah nilai yang masuk akal untuk mencegah pembagian dengan nilai yang tidak biasa. Kita juga harus memastikan dan terpisah cukup jauh untuk mencegah *floating point round off error* mempengaruhi proses substraksi.

Terdapat 3 buah metode untuk memperoleh turunan pertama suatu fungsi dengan menggunakan metode numerik, yaitu: metode selisih maju, metode selisih mundur, dan metode selisih tengah. Error pada ketiga metode numerik tersebut ditaksir menggunakan deret Taylor. Persamaan (85) dan Persamaan (86) menunjukkan persamaan untuk memperoleh turunan pertama dan taksiran error menggunakan metode selisih maju dan metode selisih mundur.

Metode nilai tengah menggunakan ukuran langkah dua kali dibandingkan dengan 2 metode lainnya. Error yang dihasilkan juga berbeda dengan kedua metode sebelumnya, dimana error dihasilkan dari pemotongan turunan ketiga pada deret Taylor. Secara umum metode selisih tengah memiliki akurasi yang lebih baik dibandingkan kedua metode sebelumnya karena metode ini mempertimbangkan dua sisi untuk memeriksa nilai . Persamaan (87) merupakan persamaan untuk memperoleh nilai turunan pertama suatu fungsi dan estimasi error menggunakan deret Taylor.

Bagaimana menentukan ? beberapa literatur menggunakan pendekatan *machine error* berdasarkan program yang digunakan untuk melakukan proses perhitungan. Metode selisih maju dan selisih mundur menggunakan pendekatan yang ditunjukkan pada Persamaan (88).

Untuk metode selisih tengah pendekatan nilai menggunakan Persamaan (89).

Kita dapat menggunakan Persamaan (85) sampai Persamaan (87) untuk membentuk sebuah program yang digunakan untuk menghitung turunan pertama suatu fungsi. Sintaks yang digunakan adalah sebagai berikut:

findiff <- function(f, x, h, method=NULL){  
 if(is.null(method)){  
 warning("please select a method")  
 }else{  
 if(method == "forward"){  
 return((f(x+h)-f(x))/h)  
 }else if(method=="backward"){  
 return((f(x)-f(x-h))/h)  
 }else if(method=="central"){  
 return((f(x+h)-f(x-h))/(2\*h))  
 }else{  
 warning("you can use method: forward, bacward, or central")  
 }  
 }  
}

Contoh 27: Hitunglah turunan pertama persamaan berikut menggunakan metode selisih titik tengah pada x =1 dan nilai h=0,05!

**Jawab**:

Untuk menghitung turunan pertama menggunakan metode selisih tengah, kita dapat menggunakan Persamaan (87). Berikut adalah proses perhitungannya:

Dengan menggunakan fungsi findiff(), hasil yang diperoleh adalah sebagai berikut:

findiff(function(x)  
 exp(-x)\*sin(2\*x)+1, x=1, h=0.05,  
 method="central")

## [1] -0.6390352

Kita dapat memperkecil nilai untuk memperoleh akurasi yang lebih baik berdasarkan pendekatan Persamaan (89).

findiff(function(x){ exp(-x)\*sin(2\*x)+1}, x=1,  
 h=1\*.Machine$double.eps^(1/3),   
 method="central")

## [1] -0.6406956

Penyelesaian persamaan matematik dalam bidang Teknik Lingkungan pada umumnya tidak hanya melibatkan turunan pertama, pada penyelesaian persamaan difusi umumnya menggunakan turunan kedua. Persamaan (90) merupakan pendekatan numerik untuk memperoleh nilai turunan kedua suatu persamaan dengan pendekatan deret Taylor.

Fungsi findiff2() merupakan fungsi yang digunakan untuk menghitung turunan kedua suatu persamaan yang didasarkan pada Persamaan (90).

findiff2 <- function(f, x, h){  
 return((f(x+h)-2\*f(x)+f(x-h))/(h^2))  
}

Kita dapat menghitung kembali turunan kedua fungsi pada Contoh 27 menggunakan fungsi findiff2(). Berikut adalah sintaks yang digunakan:

findiff2(function(x){  
 exp(-x)\*sin(2\*x)+1  
}, x=1, h=0.05)

## [1] -0.3924205

## Diferensiasi Menggunakan Fungsi Lainnya di R

Terdapat sejumlah fungsi R yang dapat digunakan untuk menghitung turunan suatu persamaan matematik. Fungsi-fungsi tersebut tersedia dalam sejumlah *library*, baik *base package* maupun yang berasal dari *library* lainnya.

### Diferensiasi Metode Titik Pusat Mengggunakan Fungsidiff()

Fungsi diff() pada *library* *base* dapat digunakan untuk menghitung turunan suatu persamaan menggunakan metode titik pusat. Fungsi ini pada umumnya digunakan untuk menghitung *lag* suatu data runtun waktu. Agar fungsi tersebut dapat digunakan untuk menghitung turunan pertama suatu persamaan, kita dapat menggunakan argumen lag = 2. Berikut adalah contoh penerapan fungsi diff() untuk memperoleh turunan pertama persamaan matematik pada Contoh 27:

f <- function(x){exp(-x)\*sin(2\*x)+1}  
x <- 1  
h <- x\*.Machine$double.eps^(1/3)  
xvec <- seq(x-h, x+h, h)  
  
# turunan pertama  
diff(f(xvec), lag=2)/(2\*h)

## [1] -0.6406956

### Diferensiasi Menggunakan *Library* numDeriv

*Library* standar yang sering digunakan untuk melakukan taksiran numerik turunan suatu fungsi adalah *library* numDeriv. Pada *library* tersebut terdapat fungsi grad() yang digunakan untuk menaksir turunan pertama suatu persamaan. Format fungsi tersebut adalah sebagai berikut:

grad(func, x, method="Richardson", method.args=list(), ...)

**Catatan:**

* **func**: Fungsi persamaan matematik yang akan dicari turunannya.
* **x**: Lokasi atau titik yang akan dicari gradiennya
* **method**: Metode estimasi yang digunakan. Metode yang dapat digunakan antara lain:
* “simple”: metode selisih maju dengan
* “Richardson”: metode interpolasi Richardson
* “complex”: *complex-step derivative approach* dan dapat digunakan untuk persamaan *complex-differentiable*.
* **method.args**: argumen tambahan yang digunakan bersama dengan argumen **method**. Jika metode yang digunakan adalah “simple”, nilai dapat dispesifikasikan pada **method.args** jika diinginkan nilai lainnya.

Berikut adalah contoh penerapan fungsi grad() untuk memperoleh turunan pertama persamaan matematik pada Contoh 27:

numDeriv::grad(function(x){exp(-x)\*sin(2\*x)+1},  
 x=1, method = "simple",   
 method.args = list(eps=1\*sqrt(.Machine$double.eps)))

## [1] -0.6406956

### DIferensiasi Menggunakan *Library* pracma

Terdapat sejumlah fungsi pada *library* pracma yang dapat digunakan untuk melakukan diferensiasi suatu persamaan matematik. FUngsi-fungsi yang dapat digunakan untuk melakukan diferensiasi sederhana antara lain: fderiv(), numderiv(), numdiff(), dan grad().

Fungsi fderiv() dapat digunakan untuk melakukan diferensiasi orde pertama sampai dengan orde tinggi. Perlu dicatat bahwa diferensiasi cenderung kurang akurat jika orde diferensiasi semakin tinggi. Format yang digunakan untuk melakukan diferensiasi menggunakan fungsi fderiv() adalah sebagai berikut:

fderiv(f, x, n = 1, h = 0,  
 method = c("central", "forward", "backward"),   
 ...)

**Catatan:**

* **f**: Fungsi persamaan matematik yang akan dicari turunannya.
* **x**: Lokasi atau titik yang akan dicari gradiennya
* **n**: Orde diferensiasi yang digunakan. Orde diferensiasi yang dapat digunakan adalah 1 sampai 8.
* **method**: Metode estimasi yang digunakan. Metode yang dapat digunakan antara lain:
* “central”: metode titik pusat
* “forward”: metode selisih maju
* “bacward”: metode selisih mundur.
* **…**: argumen tambahan fungsi **f**.

Berikut adalah contoh penerapan fungsi fderiv() untuk memperoleh turunan pertama dan kedua persamaan matematik pada Contoh 27:

library(pracma)  
# turunan 1  
fderiv(function(x){exp(-x)\*sin(2\*x)+1},  
 x = 1, n = 1, h = 1\*.Machine$double.eps^(1/3),   
 method = "central")

## [1] -0.6406956

# turunan 2  
fderiv(function(x){exp(-x)\*sin(2\*x)+1},  
 x = 1, n = 2, h = 1\*.Machine$double.eps^(1/3),   
 method = "central")

## [1] -0.3911581

Fungsi numderiv() menggunakan ekstrapolasi Richardson untuk melakukan taksiran turunan suatu persamaan matematik. Berbeda dengan fungsi lainnya, fungsi numderiv() tidak hanya menampilkan hasil diferensiasi, fungsi ini juga menampilkan error absolut, error relatif, dan jumlah iterasi yang berlangsung. Berikut adalah format fungsi yang digunakan:

numderiv(f, x0, maxiter = 16, h = 1/2, ...,   
 tol = .Machine$double.eps)

**Catatan:**

* **f**: Fungsi persamaan matematik yang akan dicari turunannya.
* **x0**: Lokasi atau titik yang akan dicari gradiennya
* **maxiter**: Iterasi maksimum yang digunakan.
* **h**: *step size* yang digunakan
* **tol**: toleransi error yang digunakan.

Berikut adalah contoh penerapan fungsi numderiv() untuk memperoleh turunan pertama persamaan matematik pada Contoh 27:

numderiv(function(x){exp(-x)\*sin(2\*x)+1},  
 x0 = 1, h = 1\*.Machine$double.eps^(1/3))

## $df  
## [1] -0.6406956  
##   
## $rel.err  
## [1] 7.63096e-11  
##   
## $niter  
## [1] 2

Fungsi numderiv() memiliki keterbatasan dalam penggunaannya. Argumen x0 yang digunakan haruslah angka numerik tunggal. Fungsi numdiff() mengatasi keterbatasan tersebut. Fungsi ini dapat menerima input berupa vektor, sehingga dapat digunakan untuk mencari nilai turunan pada sejumlah titik. Selain itu, output fungsi ini lebih sederhana, dimana hanya menampilkan hasil diferensiasinya saja. Berikut adalah format fungsi numdiff():

numdiff(f, x, maxiter = 16, h = 1/2, ...,   
 tol = .Machine$double.eps)

**Catatan:**

* **f**: Fungsi persamaan matematik yang akan dicari turunannya.
* **x**: Vektor titik yang akan dicari gradiennya
* **maxiter**: Iterasi maksimum yang digunakan.
* **h**: *step size* yang digunakan
* **tol**: toleransi error yang digunakan.

Berikut adalah contoh penerapan fungsi numdiff() untuk memperoleh turunan pertama persamaan matematik pada Contoh 27:

numdiff(function(x){exp(-x)\*sin(2\*x)+1},  
 x = 1:4, h = 1\*.Machine$double.eps^(1/3))

## [1] -0.64069556 -0.07450001 0.10951941 -0.02345058

Fungsi grad() pada *library* pracma berbeda dengan yang digunakan pada *library* numDeriv. Perbedaan utama fungsi pada kedua *library* tersebut adalah metode estimasi yang digunakan untuk menghitung turunan pertama suatu persamaan matematik. Pada *library* pracma, metode yang digunakan adalah metode titik pusat, sedangkan pada *library* numDeriv metode yang digunakan adalah metode selisih maju, ekstrapolasi Richardson, dan *complex*. Format fungsi grad() pada *library* pracma adalah sebagai berikut:

grad(f, x0, heps = .Machine$double.eps^(1/3),   
 ...)

**Catatan:**

* **f**: Fungsi persamaan matematik yang akan dicari turunannya.
* **x0**: Titik yang akan dicari gradiennya
* **heps**: *step size* yang digunakan
* **…**: Argumen lain yang digunakan pada fungsi **f**.

Berikut adalah contoh penerapan fungsi grad() untuk memperoleh turunan pertama persamaan matematik pada Contoh 27:

grad(function(x){exp(-x)\*sin(2\*x)+1}, x0 = 1)

## [1] -0.6406956

## Metode Integrasi Newton-Cotes

Metode integrasi Newton-Cotes secara umum merupakan metode integrasi yang dilakukan dengan membagi area di bawah kurva suatu fungsi menjadi beberapa panel dengan terlebih dahulu menetapkan batas atas dan batas bawah interval. Integral atau luas area di bawah kurva ditentukan berdasarkan jumlah luas panel yang digunakan untuk mendekati luas area di bawah kurva.

Terdapat beberapa metode yang akan penulis jelaskan pada sub-Chapter ini. Metode-metode tersebut antara lain:

* Metode integral Riemann
* Metode trapezoida
* Metode Simpson 1/3
* Metode Simpson 3/8

### Metode Integral Riemann

Metode integral Riemann dilakukan dengan membagi interval di bawah kurva suatu fungsi matematik sebanyak subinterval sama besar. Pada setiap subinterval dibentuk persegi panjang setinggi kurva pada setiap titik tengah persegi panjang tersebut. Area setiap subinterval diperoleh dengan mengalikan panjang dan lebar masing-masing persegi panjang. Jumlah masing-masing area tersebut digunakan untuk menaksir interval integral suatu fungsi dengan interval tertentu. Fungsi proses integrasi menggunakan metode titik tengah dapat dituliskan pada Persamaan (91).

dimana dan masing-masing merupakan batas atas dan bawah interval kurva yang hendak dihitung integralnya.

Error dari metode ini dapat diestimasi menggunakan Persamaan (92).

dimana merupakan nilai antara dan .

Contoh 28: Hitunglah intergral fungsi di bawah ini menggunakan metode integral Reimann dengan interval 0 sampai 1 dan jumlah panel 2 dan 4!

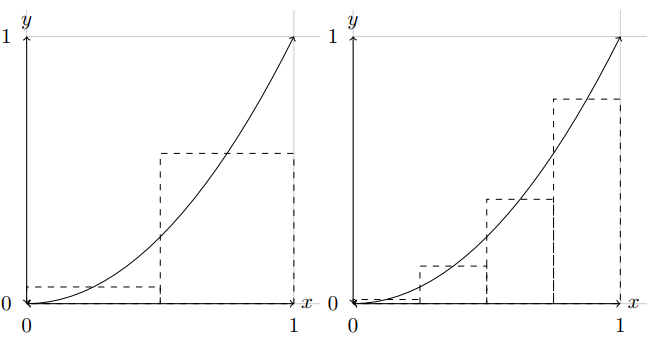
**Jawab**:

Fungsi pada Contoh 28 dapat diselesaikan menggunakan metode analitik. Penyelesaian analitik fungsi tersebut adalah sebagai berikut:

Penyelesaian numerik menggunakan metode titik tengah dengan jumlah panel 2 dapat dilakukan dengan menentukan lokasi titik tengah kedua panel. Berdasarkan interval fungsi dapat kita tentukan titik tengah kedua panel berada pada dan . Perhitungan dilakukan seperti berikut:

Untuk meningkatkan akurasi dari nilai yang dihasilkan, jumlah panel dapat ditingkatkan. Untuk jumlah panel 4, titik tengah berada pada .

Visualisasi proses integrasi dengan metode Riemann dapat dilihat pada Gambar 69.



Gambar 69: Visualisasi integral Riemann dengan 2 panel dan 4 panel (sumber:Howard, 2017).

Berdasarkan Persamaan (91), kita dapat mengembangkan fungsi R yang dapat digunakan untuk melakukan perhitungan integral Riemann. Sintaks fungsi tersebut adalah sebagai berikut:

riemann <- function(f, a, b, m = 100){  
 n\_width <- (b-a)/m  
 x <- seq(a, b-n\_width, length.out = m) + n\_width/2  
 y <- f(x)  
   
 return(sum(y)\*abs(b-a)/m)  
}

Kita akan menghitung kembali fungsi pada Contoh 28 dengan menggunakan jumlah panel 2, 4 dan 100. Berikut adalah sintaks yang digunakan:

# m=2  
riemann(function(x) x^2, a=0, b=1, m=2)

## [1] 0.3125

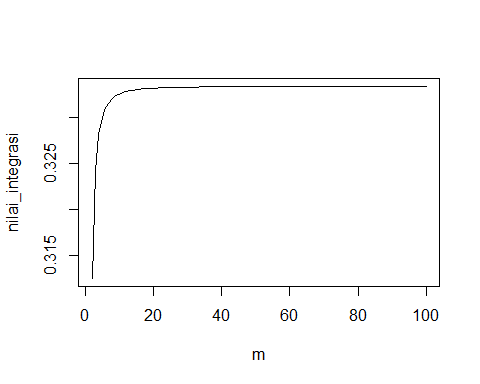
# m=4  
riemann(function(x) x^2, a=0, b=1, m=4)

## [1] 0.328125

# m=100  
riemann(function(x) x^2, a=0, b=1)

## [1] 0.333325

Berdasarkan teori yang telah dipaparkan sebelumnya, kita ketahui bahwa untuk memperoleh nilai pendekatan integral yang sebenarnya kita dapat meningkatkan jumlah panel yang digunakan. Untuk mengetahui jumlah panel minimum yang diperlukan untuk memperoleh hasil integrasi yang stabil, kita akan melakukan simulasi menggunakan data yang disajikan pada Contoh 28 dengan memvariasikan jumlah panel yang akan digunakan. Pada simulasi yang akan dilakukan kita akan coba memvariasikan jumlah panel dari 2 hingga 100. Berikut adalah sintaks yang digunakan:

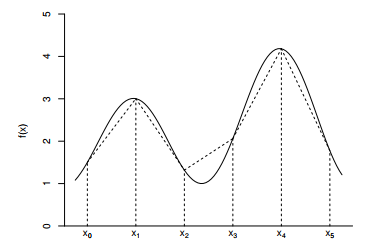


Gambar 70: Visualisasi simulasi pemilihan jumlah panel minimum metode integrasi Riemann.

Berdasarkan hasil simulasi dapat disimpulkan jumlah panel minimum yang diperlukan untuk memperoleh hasil integrasi yang stabil kira-kira sebesar .

### Metode Trapezoida

Pendekatan trapezoida dilakukan dengan melakukan pendekatan area dibawah kurva fungsi dengan subinterval menggunakan trapesium. Untuk memahami pendekatan yang digunakan pembaca dapat memperhatikan Gambar 71.



Gambar 71: Visualisasi integragrasi numerik menggunakan metode tapezoida (sumber: Jones et.al., 2014).

Fungsi proses integrasi menggunakan metode trapezoida dapat dituliskan pada Persamaan (93).

dimana

merupakan nilai subinterval dan merupakan jumlah panel trapesium yang digunakan.

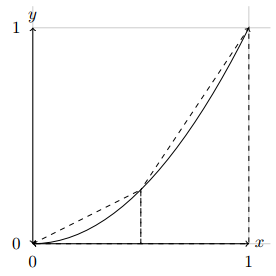
Error dari metode ini dapat diestimasi menggunakan Persamaan (94).

dimana merupakan nilai antara dan .

Contoh 29: Hitung kembali nilai intergrasi persamaan pada Contoh 28 menggunakan metode trapezoida dengan jumlah panel m=2!

**Jawab**:

Penyelesaian numerik menggunakan trapezoida dengan jumlah panel 2 dapat dilakukan dengan menentukan lokasi titik evaluasi. Berdasarkan Gambar 72, terdapat 3 batas subinterval yaitu pada ,, dan . Perhitungan intergral menggunakan ketiga titik evaluasi tersebut adalah sebagai berikut:



Gambar 72: Visualisasi integrasi metode trapezoida dengan 2 panel (sumber:Howard, 2017).

Berdasarkan Persamaan (93), kita dapat mengembangkan fungsi R yang dapat digunakan untuk melakukan perhitungan integral metode trapezoida. Sintaks fungsi tersebut adalah sebagai berikut:

trap <- function(f, a, b, m=100){  
 x <- seq(a, b, length.out = m+1)  
 y <- f(x)  
   
 p\_area <- sum((y[2:(m+1)] + y[1:m]))   
 p\_area <- p\_area \* abs(b-a)/(2\*m)  
 return(p\_area)  
}

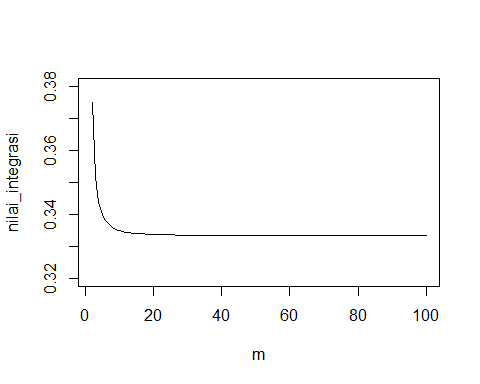
Kita dapat menghitung kembali intergral persamaan pada Contoh 28 menggunakan fungsi trap() yang telah dibuat. Berikut adalah sintaks yang digunakan:

trap(function(x)x^2, a=0, b=1, m=2)

## [1] 0.375

Untuk mengetahui jumlah panel minimum yang diperlukan untuk memperoleh hasil integrasi yang stabil pada persamaan tersebut, kita akan kembali melakukan simulasi menggunakan variasi jumlah panel yang digunakan. Dalam simulasi variasi jumlah panel yang digunakan adalah 2 sampai 100. Berikut adalah sintaks yang digunakan:

## [1] 0.33335



Gambar 73: Visualisasi simulasi pemilihan jumlah panel minimum metode integrasi trapezoida.

Berdasarkan hasil simulasi diperoleh nilai panel minimum sebesar . Hasil yang diperoleh tersebut menujukkan bahwa metode trapezoida lebih efisien dalam proses komputasi dibandingkan metode Riemann.

### Metode Simpson

Metode Simpson membagi subinterval menjadi subinterval, dimana merupakan bilangan genap. Untuk setiap pasang subinterval, luas area di bawah fungsi ditaksir menggunakan polinomial berderajat 2.

Misalkan merupakan titik sembarang pada suatu fungsi yang akan dicari integralnya yang terpisah sejauh . Untuk kita ingin menaksir menggunakan parabola yang melalui titik , , dan . Terdapat tepat 1 parabola yang dapat dibentuk dari ketiga titik koordinat tersebut yang ditunjukkan melalui Persamaan (95).

Sebagai taksiran luas di bawah kurva digunakan . Hasil integrasi kurva Persamaan (95) disajikan pada Persamaan (96).

Sekarang asumsikan merupakan bilangan genap, maka kita perlu menambahkan taksiran untuk subinterval untuk memperoleh taksiran pada integral yang disajikan pada Persamaan (97).

Persamaan (97) disebut sebagai kaidah Simpson 1/3 karena terdapat koefisien 1/3 pada bagian depan persamaan tersebut. Persamaan tersebut juga mudah diingat mengingat pola koefisien persamaan tersebut adalah . Namun penggunaan kaidah 1/3 Simpson mengharuskan jumlah subinterval genap. Kondisi tersebut jelas berbeda dengan metode trapezoida yang tidak mensyaratkan jumlah selang.

Error dari metode Simpson 1/3 dapat dihitung menggunakan Persamaan (97).

dimana merupakan nilai antara dan .

**Algoritma Metode Simpson**

1. Tentukan fungsi dan selang integrasinya .
2. Tentukan jumlah subinterval .
3. Hitung nilai selang subinterval , .
4. Tentukan awal integrasi dan akhir dan hitung nilai dan .
5. Untuk ,

* jika ganjil, hitung:
* jika genap, hitung:

1. Jumlahkan nilai-nilai taksiran tersebut menggunakan Persamaan (97).

Berdasarkan algoritma tersebut, kita dapat membentuk fungsi simpson() yang dapat digunakan untuk melakukan integrasi menggunakan metode Simpson 1/3. Berikut adalah sintaks yang digunakan:

simpson <- function(f, a, b, m=100){  
 h <- (b-a)/m # jarak selang  
 x <- a # awal selang  
 I <- f(a)+f(b)  
 sigma <- 0  
   
 if(m%%2 != 0){  
 stop("Jumlah panel harus genap")  
 }else{  
 for(i in 1:(m-1)){  
 x <- x+h  
 if(i%%2==0){  
 sigma <- sigma + 2\*f(x)  
 }else{  
 sigma <- sigma + 4\*f(x)  
 }  
 }  
 }  
   
 return((h/3)\*(I+sigma))  
}

Contoh 30: Hitung integral persamaan di bawah ini dengan menggunakan jumlah panel m=8!

**Jawab**:

lebar selang dapat dihitung seperti berikut:

Integral persamaan tersebut selanjutnya dapat dihitung menggunakan Persamaan (97):

Fungsi simpson() juga menghasilkan nilai yang serupa dengan perhitungan manual yang telah dilakukan. Berikut adalah sintaks yang digunakan:

simpson(function(x)1/(1+x), a=0, b=1, m=8)

## [1] 0.6931545

### Metode Simpson 3/8

Jika pada metode Simpson 1/3 digunakan pendekatan polinomial berderajat 2 untuk mencari luas dibawah kurva, pada metode Simpson 3/8 digunakan pendekatan polinomial berderajat 3 untuk memperoleh hasil yang lebih baik. Bentuk umum integrasi yang digunakan disajikan pada Persamaan (99).

Error dari metode ini dapat diestimasi menggunakan Persamaan (100).

**Algoritma Metode Simpson 3/8**

1. Tentukan fungsi dan selang integrasinya .
2. Tentukan jumlah subinterval .
3. Hitung nilai selang subinterval , .
4. Tentukan awal integrasi dan akhir dan hitung nilai dan .
5. Untuk ,

* jika bukan kelipatan 3, hitung:
* jika kelipatan 3, hitung:

1. Jumlahkan nilai-nilai taksiran tersebut menggunakan Persamaan (99).

Berdasarkan algoritma tersebut, fungsi R dapat disusun untuk melakukan komputasi metode simpson 3/8. Berikut adalah sintaks fungsi tersebut:

simpson38 <- function(f, a, b, m=90){  
 h <- (b-a)/m # jarak selang  
 x <- a # awal selang  
 I <- f(a)+f(b)  
 sigma <- 0  
   
 if(m%%3 != 0){  
 stop("jumlah panel harus kelipatan 3")  
 }else{  
 for(i in 1:(m-1)){  
 x <- x+h  
 if(i%%3==0){  
 sigma <- sigma + 2\*f(x)  
 }else{  
 sigma <- sigma + 3\*f(x)  
 }  
 }  
 }  
   
 return((3\*h/8)\*(I+sigma))  
}

Contoh 31: Hitung kembali integral persamaan yang disajikan pada Contoh 30 menggunakan fungsi simpson38() yang telah dibuat sebelumnya!

**Jawab**:

Berikut adalah sintaks yang digunakan untuk melakukan proses integrasi Simpson 3/8 menggunakan fungsi simpson38():

simpson38(function(x)1/(1+x), a=0, b=1, m=9)

## [1] 0.6931573

## Metode Integrasi Newton-Cotes Mengunakan Fungsi Lainnya

Terdapat sejumlah *library* yang menyediakan fungsi yang dapat digunakan untuk melakukan proses integrasi suatu fungsi pada R. *Library* pracma menyediakan fungsi-fungsi seperti trapz() dan cotes().

Fungsi trapz() merupakan fungsi yang digunakan untuk melakukan integrasi dengan pendekatan trapesium. Penggunaan fungsi tersebut untuk melakukan perhitungan integral tidak dapat dilakukan secara langsung, kita perlu membuat terlebih dahulu seri koordinat x dan koordinat y. Program selanjutnya akan melakukan interpolasi linier terhadap selang yang saling berdekatan. Luas masing-masing panel selang selanjutnya dihitung dan dijumlahkan untuk memperoleh nilai integral. Format fungsi tersebut adalah sebagai berikut:

trapz(x, y)

**Catatan:**

* **x**: vektor sumbu x.
* **y**: vektor sumbu y.

Untuk lebih memahami penerapannya berikut disajikan contoh untuk mencari integral persamaan pada Contoh 28:

library(pracma)  
  
f <- function(x)x^2  
x <- seq(0, 1, length.out = 101) # membuat subinterval panel sebanyak 100  
y <- f(x)  
  
trapz(x, y)

## [1] 0.33335

Fungsi lain yang dapat digunakan untuk menghitung integral suatu fungsi menggunakan metode Newton-Cotes adalah cotes(). Pada fungsi tersebut kita perlu menyatakan jumlah subinterval yang digunakan dan jumlah nodes interpolasi yang digunakan. Jumlah nodes akan menentukan fungsi polinomial pendekatan yang digunakan untuk menghitung luas di bawah suatu fungsi. Jika jumlah nodes diatur menjadi dua, maka fungsi pendekatannya berupa garis linier (metode trapezoida). Jika nodes diatur menjadi 3 maka pendekatannya berupa fungsi kuadrat (metode Simpson 1/3). Secara sederhana derajat polinomial memerlukan nodes. Format fungsi cotes() disajikan sebagai berikut:

cotes(f, a, b, n, nodes, ...)

**Catatan:**

* **f**: fungsi yang akan dicari integralnya
* **a**: batas atas.
* **b**: batas bawah.
* **n**: jumlah subinterval atau panel.
* **nodes**: jumlah nodes yang digunakan untuk interpolasi fungsi pada tiap subinterval.

Untuk lebih memahami penerapannya berikut adalah contoh perhitungan intergral menggunakan persamaan pada Contoh 30.

f <- function(x)1/(1+x)  
a <- 0; b <- 1  
  
# metode trapezoida  
cotes(f, a, b, n=8, nodes=2)

## [1] 0.6941219

# metode Simpson 1/3  
cotes(f, a, b, n=8, nodes=3)

## [1] 0.6931545

# metode Simpson 3/8  
cotes(f, a, b, n=9, nodes=4)

## [1] 0.6931573

## Metode Kuadratur Gauss

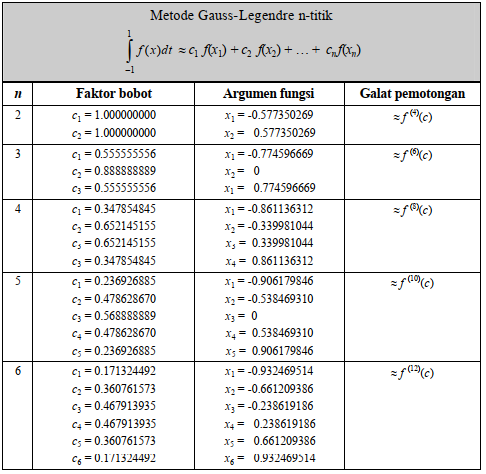
Metode Newton-Cotes sangat *powerful*, tetapi metode tersebut memiliki dua fitur yang kurang diinginkan. Pertama, kita harus menggunakan evaluasi fungsi sejumlah untuk hasil presisi dan pada polinomial berderajat . Hal tersebut mungkin tampak seperti rasio yang baik, tetapi dalam praktiknya, jumlah titik evaluasi akan sering ditingkatkan untuk memperoleh akurasi yang lebih tinggi, namun hasil yang diperoleh juga tidak presisi. Sebagai contoh pembaca dapat melakukan simulasi dengan memvariasikan jumlah penel yang digunakan untuk memperoleh nilai integral sebuah fungsi. Jika kita plotkan hasil yang kita peroleh, grafik yang muncul berupa garis yang berosilasi khusunya pada penggunaan polinomial berderajat tinggi yang menunjukkan hasil yang diperoleh menjadi kurang presisi.

Kelemahan kedua, metode Newton-Cotes memerlukan fungsi terintegrasi untuk dievaluasi pada node yang berjarak sama. Ini benar terlepas dari fungsi yang digunakan. Setiap panel membutuhkan node dengan jarak yang sama di dalamnya. Ini bisa menjadi masalah dengan fungsi periodik, di mana diskontinuitas periodik dapat secara kebetulan mendarat di titik evaluasi. Jika panel Newton-Cotes dapat menyebabkan masalah, kita dapat menggunakan integrasi Gaussian untuk menyelesaikan integral.

Secara umum integrasi Gauss berusaha memperoleh pendekatan luas dibawah kurva fungsi dengan memecah fungsi tersebut menjadi faktor bobot dan yang merupakan polinomial pendekatannya. Integral diperoleh melalui hasil kali dari bobot dan fungsi polinomial. Jumlah bobot dan fungsi yang digunakan bergantung pada orde polinomial yang akan digunakan untuk mengestimasi integral suatu fungsi. Bentuk umum dari kaidah Gauss tersebut ditampilkan pada Persamaan (101).

dimana merupakan faktor bobot dan merupakan titik evaluasi.

Nilai faktor bobot dan nilai masing-masing titik evaluasi disajikan pada Gambar 74.



Gambar 74: Tabulasi faktor bobot, titik evaluasi dan galat pemotongan.

Fungsi yang akan dicari nilai integrannya pada umumnya tidak hanya memiliki daerah batas , sehingga pendekatan kuadratur Gauss tidak dapat digunakan secara langsung pada fungsi yang tidak memiliki batas tersebut. Agar kuadratur Gauss tetap dapat digunakan, fungsi tersebut perlu dilakukan transformasi. Proses transformasi dituliskan pada Persamaan (102).

**Algoritma Kuadratur Gauss-Legendre**

1. Tentukan fungsi dan selang integrasinya .
2. Lakukan transformasi fungsi tersebut hingga diperoleh fungsi dengan selang menggunakan Persamaan (102).
3. Tentukan orde polinomial yang akan digunakan.
4. Lakukan proses integrasi dengan mengalikan faktor bobot dengan seperti yang ditujukkan pada Persamaan (101).

Kita dapat membangun suatu fungsi pada R untuk melakukan integrasi Gauss berdasarkan algoritma tersebut. Sintaks fungsi tersebut adalah sebagai berikut:

gauss\_legendre <- function(f, n){  
 if(n==2){  
 c <- c(1,1)  
 x <- c(-0.577350269, 0.577350269)  
 integral <- 0  
   
 for(i in 1:n){  
 integral <- integral + c[i]\*f(x[i])  
 }  
 }else if(n==3){  
 c <- c(0.555555556,0.888888889,0.555555556)  
 x <- c(-0.774596669,0,0.774596669)  
 integral <- 0  
   
 for(i in 1:n){  
 integral <- integral + c[i]\*f(x[i])  
 }  
 }else if(n==4){  
 c <- c(0.347854845,0.652145155,0.652145155,  
 0.347854845)  
 x <- c(-0.861136312,-0.339981044,0.339981044,  
 0.861136312)  
 integral <- 0  
   
 for(i in 1:n){  
 integral <- integral + c[i]\*f(x[i])  
 }  
 }else if(n==5){  
 c <- c(0.236926885,0.478628670,0.568888889,  
 0.478628670,0.236926885)  
 x <- c(-0.906179846,-0.538469310,0,  
 0.538469310, 0.906179846)  
 integral <- 0  
   
 for(i in 1:n){  
 integral <- integral + c[i]\*f(x[i])  
 }  
 }else if(n==6){  
 c <- c(0.171324492,0.360761573,0.467913935,  
 0.467913935,0.360761573,0.171324492)  
 x <- c(-0.932469514,-0.661209386,-0.238619186,  
 0.238619186, 0.661209386,0.932469514)  
 integral <- 0  
   
 for(i in 1:n){  
 integral <- integral + c[i]\*f(x[i])  
 }  
 }else{  
 stop("n harus ditentukan")  
 }  
 return(integral)  
}

Contoh 32: Hitunglah integral fungsi berikut menggunakan metode Gauss-Legendre 2 titik!

**Jawab**:

Agar fungsi tersebut dapat dicari nilai integralnya menggunakan metode Gauss-Legendre, fungsi tersebut perlu ditransformasi terlebih dahulu:

Jadi dalam hal ini

maka

Kita juga dapat menggunakan fungsi gauss\_legendre() untuk melakukan integrasi Gauss-Legendre pada dua titik. Berikut adalah sintaks yang digunakan:

0.5\*gauss\_legendre(function(x)((1.5+0.5\*x)^2)+1,  
 n=2)

## [1] 3.333333

## Metode Gauss-Legendre Menggunakan Fungsi legendre.quadrature()

Fungsi legendre.quadrature() dari *library* gaussquand dapat dijadikan alternatif untuk menghitung integral suatu fungsi menggunakan metode Gauss-Lagendre. Fortmat fungsi tersebut adalah sebagai berikut:

legendre.quadrature(functn, rule, lower=-1, upper=1,   
 weighted = TRUE, ...)

**Catatan:**

* **functn**: fungsi yang akan dicari integralnya
* **rule**: data frame yang terdiri atas orde n aturan kuadratur legendre
* **lower**: batas atas.
* **upper**: batas bawah.
* **weighted**: nilai boolean yang menyatakan apakah bobot fungsi disertakan dalam integran
* **…**: argumen tambahan functn

Untuk menentukan rule pada fungsi legendre.quadrature(), diperlukan fungsi lain yang tersedia pada *library* gaussquad. Fungsi tersebut adalah legendre.quadrature.rules(). Fungsi tersebut akan menampilkan list data frame berdasarkan orde polinomial yang digunakan sebagai taksiran yang terdiri atas faktor bobot dan titik evaluasi. Format fungsi tersebut adalah sebagai berikut:

legendre.quadrature.rules(n,normalized=FALSE)

**Catatan:**

* **n**: orde tertinggi polinomial yang akan ditampilkan
* **normalized**: nilai boolean. Jika bernilai TRUE, aturan digunakan untuk polinomial ortogonal.

Untuk memahami penerapan kedua fungsi tersebut berikut disajikan contoh penerapannya:

library(gaussquad)  
# menampilkan aturan untuk orde 4  
legendre.quadrature.rules(4)

## [[1]]  
## x w  
## 1 0 2  
##   
## [[2]]  
## x w  
## 1 0.5773503 1  
## 2 -0.5773503 1  
##   
## [[3]]  
## x w  
## 1 7.745967e-01 0.5555556  
## 2 7.771561e-16 0.8888889  
## 3 -7.745967e-01 0.5555556  
##   
## [[4]]  
## x w  
## 1 0.8611363 0.3478548  
## 2 0.3399810 0.6521452  
## 3 -0.3399810 0.6521452  
## 4 -0.8611363 0.3478548

# mencari integral suatu fungsi dengan orde  
# gauss-legendre sebesar 4  
f <- function(x)x^6  
rule <- legendre.quadrature.rules(4)[[4]]  
  
legendre.quadrature(f, rule, lower=-1, upper=1)

## [1] 0.2857143

## Metode Integrasi Adaptif

Integrasi adaptif menyediakan pendekatan yang berbeda untuk memperoleh nilai intergral suatu fungsi. Salah satu prinsip utama dari analisis numerik adalah bahwa kita harus berkomitmen pada semacam analisis manusia terhadap suatu masalah sebelum mencoba menyelesaikannya secara algoritmik. Metode analisis numerik umumnya tidak dapat menyelesaikan semua masalah dengan sangat baik. Jadi pengetahuan terhadap masalah yang hendak diselesaikan dapat memungkinkan kita memilih metode numerik yang lebih baik sesuai dengan masalah. Misalnya, dalam konteks integrasi numerik, diskontinuitas pada titik akhir tidak akan cocok untuk solusi Newton-Cotes yang bersifat tertutup.

Tentu saja, akan lebih baik jika kita bisa memprogram komputer untuk mempelajari sesuatu tentang masalah, daripada kita melakukan itu sendiri. Metode adaptif memberikan pendekatan untuk melakukan hal ini. Metode integrasi adaptif memeriksa integral yang mereka operasikan dan mengubah parameter mereka sendiri untuk meningkatkan kualitas integrasi. Algoritma adaptif yang paling sederhana memberikan pendekatan *brute force* untuk peningkatan kualitas dengan memeriksa *error* integrasi. Di sisi lain, jika kita tahu *error*-nya, secara teoritis kita bisa memperbaiki estimasi. Di situlah batas *error* pada algoritma Newton-Cotes dapat membantu.

Jika kita dapat menemukan sesuatu tentang *error* tersebut, kita dapat menggunakan informasi itu untuk memperbaiki estimasi pada proses integrasi. Bayangkan kita sedang mengintegrasikan suatu fungsi, pada batas . Jika kita menggunakan metode titik tengah (integral Riemann). Kita telah menetahui *error* maksimum yang mungkin terjadi pada metode tersebut melalui Persamaan (92). Dua pengamatan segera menjadi jelas. Terlepas dari apa fungsi itu, atau turunan keduanya, dua perubahan dapat dilakukan pada integrasi untuk meningkatkan kualitasnya. Pertama, *error* adalah proporsi terhadap kubik dari panjang domain integrasi. Mengurangi panjang meningkatkan kualitas dan memotong panjang menjadi dua memberikan peningkatan kualitas delapan kali lipat. Kedua, *error* berbanding terbalik dengan jumlah panel m yang digunakan . Meningkatkan panel mengurangi *error* dan menggandakan jumlah panel yang disediakan dapat meeningkatkan kualitas empat kali lipat.

Kita dapat merancang algoritma di sekitar pengamatan ini. Pertama, kita dapat memperkirakan nilai integral , menggunakan aturan titik tengah 1-point. Kedua, kita bisa memperkirakannya lagi menggunakan aturan titik tengah 2-point . Karena kita telah menggandakan jumlah titik dalam aturan, kita sekarang tahu bahwa perbedaan maksimum antara dan , nilai sebenarnya dari integral, tidak lebih dari 4 kali lebih besar dari perbedaan maksimum antara dan . Jika kurang dari toleransi tertentu, maka perbedaan antara dan harus kurang dari tiga kali toleransi yang sama.

Kita peeriksa perbedaan antara dua perkiraan. Jika perbedaannya lebih besar dari toleransi, kita mungkin masih berada dalam toleransi, tetapi kita belum pasti. Jadi proses membagi wilayah integrasi menjadi dua, dan menerapkan integrator adaptif untuk kedua bagian, secara terpisah, menjumlahkan hasilnya akan terus dilakukan.

**Algoritma Metode Riemann Adaptif**

1. Tentukan fungsi dan selang integrasinya .
2. Tentukan jumlah subinterval .
3. Jika , hitung luas area di bawah kurva dengan pendekatan metode Riemann dengan .
4. Jika ,

* Hitung dengan pendekatan metode Riemann dan
* Hitung dengan pendekatan metode Riemann dan

1. Jika ,

* Perkecil sebanyak 1,
* Perkecil menjadi setengahnya
* Bagi area integrasi menjadi dua bagian dengan menetapkan sebagai batas, sehingga terdapat dua batas yaitu: dan .
* Lakukan perhitungan kembali integral pada masing-masing batas tersebut menggunakan metode Riemann dan cek apakah .

1. Jika , luas integral = .

Kita dapat membangun sebuah fungsi integral adaptif menggunakan algoritma tersebut. Sintaks fungsi tersebut adalah sebagai berikut:

riemann\_adaptint <- function(f, a, b, m=10, tol=1e-8){  
 if(m<1){  
 stop("m harus >= 1")  
 }else if(m==1){  
 m <- 2  
 n\_width <- (b-a)/m  
 x <- seq(a, b-n\_width, length.out = m) + n\_width/2  
 y <- f(x)  
 area <- sum(y)\*abs(b-a)/m  
 }else{  
   
 m1 <- 1  
 n\_width1 <- (b-a)/m1  
 x1 <- seq(a, b-n\_width1, length.out = m1) + n\_width1/2  
 y1 <- f(x1)  
 q1 <- sum(y1)\*abs(b-a)/m1  
   
 m2 <- 2  
 n\_width2 <- (b-a)/m2  
 x2 <- seq(a, b-n\_width2, length.out = m2) + n\_width2/2  
 y2 <- f(x2)  
 q2 <- sum(y2)\*abs(b-a)/m2  
   
 if(abs(q1-q2)>3\*tol){  
 m <- m-1  
 tol <- tol/2  
 c <- (a+b)/2  
 lt <- riemann\_adaptint(f, a, c, m=m, tol=tol)  
 rt <- riemann\_adaptint(f, c, b, m=m, tol=tol)   
 area <- lt+rt  
 }else{  
 area <- q2  
 }  
 }  
   
 return(area)  
}

Contoh 33: Hitung integral fungsi di bawah ini dengan menggunakan integral Riemann adaptif dengan jumlah panel yang digunakan m=100!

**Jawab**:

Kita akan menghitung integral dari fungsi tersebut menggunakan fungsi R yang telah dibuat. Berikut adalah sintaks yang digunakan:

riemann\_adaptint(function(x)sin(x)^2 + log(x),  
 a=1, b=10, m=100)

## [1] 18.52494

## Metode Integral Adaptif Menggunakan Fungsi Lainnya Pada R

Terdapat dua buah fungsi yang hendak penulis kenalkan pada pembaca yang berfungsi untuk melakukan integrasi adaptif pada R. Fungsi-fungsi tersebut antara lain: integrate() dari *library* base dan integral() dari *library* pracma.

Fungsi integrate() merupakan fungsi yang akan melakukan integrasi numerik menggunakan metode kudratur adaptif untuk sebuah variabel dengan selang terbatas (*finite*) maupun tidak terbatas (*infinite*). Format fungsi tersebut secara umum adalah sebagai berikut:

integrate(f, lower, upper, ..., subdivisions = 100L,  
 rel.tol = .Machine$double.eps^0.25,   
 abs.tol = rel.tol)

**Catatan:**

* **f**: fungsi yang akan dicari integralnya
* **lower**: batas bawah.
* **upper**: batas atas.
* **…**: argumen tambahan functn
* **subdivision**: jumlah subinterval atau panel yang akan digunakan.
* **rel.tol**: nilai akurasi relatif yang hendak dicapai
* **abs.tol**: nilai akurasi absolut yang hendak dicapai

Contoh penerapan fungsi integrate() adalah sebagai berikut:

integrate(function(x)sin(x)^2 + log(x),  
 lower=1, upper=10,  
 rel.tol = 1e-8)

## 18.52494 with absolute error < 4.1e-10

Fungsi lainnya yang dapat digunakan untuk melakukan komputasi integral adaptif adalah fungsi integral() dari *library* pracma. Terdapat dua buah metode integrasi adaptif yang dapat digunakan pada fungsi tersebut yaitu: Gauss-Konrod dan Simpson. Metode Clenshaw-Curtis yang tersedia masih belum dapat melakukan integrasi adaptif melalui fungsi tersebut. Format umum fungsi integral() adalah sebagai berikut:

integral(fun, xmin, xmax,  
 method = c("Kronrod", "Clenshaw","Simpson"),  
 no\_intervals = 8, reltol = 1e-8,   
 abstol = 0, ...)

**Catatan:**

* **fun**: fungsi yang akan dicari integralnya
* **xmin**: batas bawah.
* **xmax**: batas atas.
* **method**: metode imtegrasi yang digunakan.
* **…**: argumen tambahan fun.
* **no\_intervals**: jumlah subinterval atau panel yang akan digunakan.
* **reltol**: nilai akurasi relatif yang hendak dicapai
* **abstol**: nilai akurasi absolut yang hendak dicapai

Berikut merupakan contoh penerapan fungsi integral():

pracma::integral(function(x)sin(x)^2 + log(x),  
 xmin=1, xmax=10, method="Simpson",  
 reltol = 1e-8)

## [1] 18.52494

## Metode Integrasi Romberg

Seperti halnya algoritma integrasi adaptif, integrasi Romberg adalah perluasan yang relatif mudah dari keluarga algoritma Newton-Cotes. Keduanya bekerja dengan menggunakan iterasi yang disempurnakan dari beberapa metode Newton-Cotes yang mendasarinya untuk memberikan perkiraan nilai integral yang lebih akurat. Tidak seperti proses komputasi fungsi riemann\_adapint(), integrasi Romberg bukanlah pendekatan adaptif terhadap integrasi. Hal tersebut berarti metode Romberg tidak mengubah perilakunya sendiri berdasarkan perilaku fungsi yang akan diintegrasikan. Sebaliknya, kita mengeksploitasi perilaku fungsi trapesium pada batas untuk menghasilkan estimasi integral.

Untuk memahami integrasi Romberg, kita harus mulai dengan implementasi rekursif dari aturan trapesium. Jika kita mulai dengan suatu fungsi, di mana adalah fungsi trapesium, adalah fungsi yang akan diintegrasikan, dan adalah jumlah panel untuk diintegrasikan, maka,

di mana adalah aturan Simpson. Kemudian, jika kita mendefinisikan , maka fungsi rekursif kita selesai, karena berdasarkan hubungan ini, fraksi yang diberikan dalam Persamaan (103) juga merupakan perkiraan untuk integral.

Secara umum,

di mana adalah aturan trapesium satu panel dan adalah aturan trapesium dengan panel . Dengan menggunakan fungsi-fungsi dasar ini, dapat ditemukan secara iteratif sebagai matriks segitiga-bawah di mana masing-masing nilai di kolom yang bukan paling kiri adalah fungsi dari nilai di sebelah kiri dan entri di atasnya.

Definisi rekursif ini muncul dari ekstrapolasi Richardson. Ketika diterapkan pada algoritma trapesium, yang konvergen menuju nilai sebenarnya dari integral sebagai (jumlah panel) meningkat, hubungan dalam Persamaan (104) muncul. Penting untuk dipahami bahwa pada batas ketika mendekati tak terhingga, nilai adalah nilai sejati integral. Untuk nilai yang lebih kecil dari , integral Romberg masih hanya perkiraan, meskipun hasil yang diperoleh sangat bagus.

**Algoritma Metode Integrasi Romberg**

1. Tentukan fungsi dan selang integrasinya .
2. Tentukan jumlah subinterval .
3. Bentuk matrik dengan ukuran yang akan menampung hasil perhitungan.
4. Untuk hitung integral fungsi menggunakan metode trapezoida dengan .
5. Untuk dan , hitung integral dengan jumlah panel
6. Untuk dan hitung nilai perbaikan nilai integrasi menggunakan Persamaan (104).
7. Solusi integrasi diperoleh pada .

Berdasarkan algoritma tersebut, kita akan menyusun suatu fungsi pada R untuk melakukan proses komputasi integrasi dengan metode Romberg. Berikut adalah sintaks fungsi yang dibuat:

romberg <- function(f, a, b, m, tab=FALSE){  
 R <- matrix(NA, nrow=m, ncol=m)  
   
 R[1,1] <- trap(f, a, b, m=1)  
 for(j in 2:m){  
 R[j,1] <- trap(f, a, b, m=2^(j-1))  
 for(k in 2:j){  
 k4 <- 4^(k-1)  
 R[j,k] <- (k4\*R[j,k-1]-R[j-1,k-1])/(k4-1)  
 }  
 }  
   
 if(tab==TRUE){  
 return(R)  
 }else{  
 return(R[m,m])  
 }  
}

Contoh 34: Hitung integral fungsi yang ditampilkan pada Contoh 33 dengan m=10!

**Jawab**:

Kita dapat menggunakan fungsi romberg() untuk melakukan proses integrasi menggunakan metode Romberg. Berikut adalah sintaks yang digunakan:

# menampilkan matriks proses perhitungan  
romberg(function(x)sin(x)^2 + log(x),  
 a=1, b=10, m=10, tab=TRUE)

## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 14.87978 NA NA NA NA NA NA  
## [2,] 17.35130 18.17514 NA NA NA NA NA  
## [3,] 18.18703 18.46561 18.48497 NA NA NA NA  
## [4,] 18.43393 18.51624 18.51961 18.52016 NA NA NA  
## [5,] 18.50156 18.52411 18.52463 18.52471 18.52473 NA NA  
## [6,] 18.51905 18.52488 18.52493 18.52493 18.52493 18.52493 NA  
## [7,] 18.52346 18.52493 18.52494 18.52494 18.52494 18.52494 18.52494  
## [8,] 18.52457 18.52494 18.52494 18.52494 18.52494 18.52494 18.52494  
## [9,] 18.52485 18.52494 18.52494 18.52494 18.52494 18.52494 18.52494  
## [10,] 18.52492 18.52494 18.52494 18.52494 18.52494 18.52494 18.52494  
## [,8] [,9] [,10]  
## [1,] NA NA NA  
## [2,] NA NA NA  
## [3,] NA NA NA  
## [4,] NA NA NA  
## [5,] NA NA NA  
## [6,] NA NA NA  
## [7,] NA NA NA  
## [8,] 18.52494 NA NA  
## [9,] 18.52494 18.52494 NA  
## [10,] 18.52494 18.52494 18.52494

Berdasarkan hasil perhitungan nilai integral fungsi tersebut adalah 18.524939.

## Metode Integrasi Romberg Menggunakan Fungsi Lainnya

Fungsi romberg() pada *library* pracma dapat digunakan untuk melakukan integrasi metode Romberg. Format umum fungsi tersebut adalah sebagai berikut:

romberg(f, a, b, maxit = 25, tol = 1e-12, ...)

**Catatan:**

* **f**: fungsi yang akan dicari integralnya
* **a**: batas bawah.
* **b**: batas atas.
* **…**: argumen tambahan functn
* **maxit**: jumlah iterasi maksimum.
* **tol**: nilai akurasi yang hendak dicapai

Berikut adalah contoh penerapan fungsi romberg():

pracma::romberg(function(x)sin(x)^2 + log(x),  
 a=1, b=10)

## $value  
## [1] 18.52494  
##   
## $iter  
## [1] 8  
##   
## $rel.error  
## [1] 7.105427e-15

## Metode Integrasi Monte Carlo

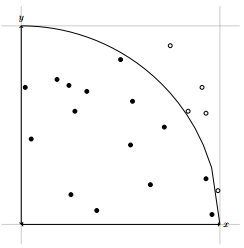
Nama Monte Carlo berasal dari daerah di Monako, yang terkenal karena aktikvitas kasino dan perjudiannya. Jelas, permainan kasino yang baik tergantung pada keacakan, seperti juga metode Monte Carlo. Nama ini menggambarkan pentingnya keacakan dalam proses karena algoritma Monte Carlo menggunakan generator angka acak untuk membedakan input ke suatu fungsi.

Angka acak harus berasal dari domain fungsi yang diharapkan. Selanjutnya, fungsi itu sendiri bersifat deterministik karena untuk diberikan dua input dari domain fungsi dan . Jika , maka . Generator angka acak digunakan untuk menghasilkan sejumlah besar input dan fungsinya dijalankan pada setiap input. Akhirnya, hasil yang diperoleh dikumpulkan sesuai dengan model logika yang sesuai dengan analisis yang dilakukan.

Metode Monte Carlo dapat digunakan untuk integrasi numerik dalam jumlah dimensi apa pun yang diberikan. Pendekatan mendasar metode ini adalah menempatkan beberapa titik secara acak di atas domain untuk diintegrasikan. Jika titik terletak “di bawah” garis fungsi, maka titik tersebut dianggap dalam area integrasi. Jika titiknya “di atas” garis fungsi, maka titik tersebut bukan berada diluar garis integrasi. Area di bawah perkiraan kurva adalah persentase titik di bawah garis.

Beberapa algoritma Monte Carlo yang paling awal digunakan untuk menemukan area di bawah kurva atau untuk memperkirakan nilai sebuah hobi favorit matematikawan sejak dahulu. Satu pendekatan menciptakan seperempat lingkaran, menggunakan fungsi . Melalui domain , ini adalah fungsi dan merupakan hasil dari penyelesaian persamaan standar untuk lingkaran, untuk di mana .

Gambar 75 menunjukkan plot fungsi ini. Selain itu, 20 titik acak dipilih. Jika titik di bawah kurva dilambangkan dengan lingkaran hitam terisi dan titik-titik kurva dilambangkan dengan titik bulat kosong. Dalam contoh ini, terdapat 15 titik berada di bawah kurva, mengarah ke estimasi area luas area 0,75. Karena kurva mewakili seperempat lingkaran, estimasi untuk adalah 3. Meningkatkan jumlah tes titik acak meningkatkan ketepatan estimasi dan akurasi.



Gambar 75: Visualisasi metode Monte-Carlo untuk fungsi setengah lingkaran dengan jumlah bilangan acak 20 (sumber: Jones et.al., 2014).

Bentuk umum metode Monte-Carlo disajikan pada Persamaan (105).

dimana merupakan jumlah titik yang akan dievaluasi.

**Algoritma Metode Monte Carlo**

1. Tentukan fungsi dan selang integrasinya .
2. Tentukan jumlah titik acak yang akan digunakan .
3. Lakukan produksi titik acak dengan selang sejumlah
4. Hitung
5. Hitung estimasi area menggunakan Persamaan (105)

Berdasarkan algoritma tersebut, fungsi R dapat dibangun untuk melakukan integrasi numerik menggunakan metode Monte Carlo. Berikut adalah sintaks yang digunakan:

monte\_int <- function(f, a, b, m=1e6){  
 x <- runif(m, min=a, max=b)  
   
 return((b-a)\*sum(f(x))/m)  
}

Contoh 35: Hitung integral fungsi yang ditampilkan pada Contoh 33 menggunakan metode Monte-Carlo dengan m=1e6!

**Jawab**:

Integrasi Monte Carlo menggunakan menggunakan fungsi monte\_int() disajikan pada sintaks berikut:

monte\_int(function(x)sin(x)^2 + log(x), a=1, b=10)

## [1] 18.51559

Hasil yang diperoleh sedikit berbeda dengan yang dihasilkan oleh metode lainnya. Hal ini disebabkan oleh penggunaan bilangan acak pada proses integrasi. Selain itu, metode ini juga menghasilkan kualitas hasil yang rendah dengan tingkat komputasi yang tinggi dibandingkan dengan metode Newton-Cotes.

Keunggulan metode Monte Carlo dibandingkan metode sebelumnya adalah kemampuan untuk menangani proses integrasi berganda. Berikut adalah bentuk umum proses integrasi bivariat menggunakan metode Monte Carlo:

dimana merupakan area perpotongan dimana fungsi diintegrasikan.

**Algoritma Metode Monte Carlo Bivariat**

1. Tentukan fungsi dan domain integrasinya pada masing-masing sumbu dan
2. Tentukan jumlah titik acak yang akan digunakan .
3. Lakukan produksi titik acak dan masing-masing domain sumbunya sejumlah
4. Hitung
5. Hitung estimasi volume menggunakan Persamaan (106)

monte\_int2 <- function(f, xdom, ydom, m=1000){  
 xmin <- min(xdom)  
 xmax <- max(xdom)  
 ymin <- min(ydom)  
 ymax <- max(ydom)  
   
 x <- runif(m, min=xmin, max=xmax)  
 y <- runif(m, min=ymin, max=ymax)  
   
 V <- (xmax-xmin)\*(ymax-ymin)  
   
 return(V\*sum(f(x,y))/m)  
}

Contoh 36: Hitung volume melalui integrasi persamaan berikut menggunakan metode Monte Carlo dengan domain x = [0,1] dan y=[0,1]!

**Jawab**:

Berikut adalah sintaks yang digunakan untuk melakukan integrasi persamaan tersebut menggunakan metode Monte Carlo bivariat:

monte\_int2(function(x,y)x^2\*y,  
 xdom=c(0,1), ydom=c(0,1))

## [1] 0.1679889

Karena metode Monte Carlo tidak deterministik, *error* integrasi Monte Carlo tidak dibatasi dalam pengertian yang telah kita lihat sejauh ini. Namun, kita dapat memperkirakan varians dari estimasi yang dihasilkan, yang berkurang dengan meningkatnya jumlah poin:

dimana . Definisi ini juga digunakan untuk proses integrasi dengan dimensi yang lebih tinggi.

Perlu diketahui pula bahwa metode Monte Carlo hanya dapat digunakan jika nilai terendah dari variabel bebas yang digunakan tidak negatif. Hal ini dilakukan untuk mencegah adanya pengurangan dengan nilai negatif sehingga hasil integrasi jauh lebih besar dari yang seharusnya.

## Studi Kasus

### Penerjung Payung

Pada studi kasus kali ini, penulis akan memberikan contoh penerapan integrasi numerik dalam menganalisa jarak jatuh seorang penerjung payung yang melompat dari sebuah pesawat. Kecepatan penerjun payung dapat dituliskan ke dalam sebuah fungsi dari waktu,

dimana adalah kecepatan penerjun payung dalam , adalah percepatan gravitasi sebesar , adalah massa penerjun payung sebesar , dan adalah koefisien tahanan udara sebesar .

Misalkan kita ingin mengetahui seberapa jauh penerjun telah jatuh setelah waktu tertentu . Karena kecepatan merupakan turunan pertama dari fungsi jarak, maka jarak penerjun dari titik terjun adalah:

Jika kita ingin mengetahui jarak yang telah ditempuh saat , kita dapat melakukan integrasi pada persamaan tersebut dengan domain . Persamaan (109) dapat dinyatakan menjadi Persamaan (110) dengan memasukkan semua komponen yang telah diketahui sebelumnya.

Kita dapat menyelesaikan Persamaan (110) dengan menggunakan metode-metode integrasi numerik yang telah dijabarkan sebelumnya. Berikut adalah sintaks untuk masing-masing metode tersebut:

f <- function(x)((9.8\*68.1)/12.5)\*(1-exp(-(12.5/68.1)\*x))  
a <- 0; b <- 10

**Newton-Cotes**

# Metode Riemann  
riemann(f, a, b)

## [1] 289.4386

# Metode Trapezoida  
trap(f, a, b)

## [1] 289.4283

# Metode Simpson 1/3  
simpson(f, a, b)

## [1] 289.4351

# Metode Simpson 3/8  
simpson38(f, a, b)

## [1] 289.4351

**Metode Adaptif**

riemann\_adaptint(f, a, b, m=100)

## [1] 289.4351

**Metode Romberg**

romberg(f, a, b, m=10)

## [1] 289.4351

**Metode Monte Carlo**

monte\_int(f, a, b)

## [1] 289.6182

## Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press.
2. Chapra, S.C. Canale, R.P. 2015. **Numerical Methods For Engineers, Seventh Edition**. Mc Graw Hill.
3. Howard, J.P. 2017. **Computational Methods for Numerical Analysis with R**. CRC Press.
4. Kreyszig, E. 2011. **Advanced Engineering Mathematics, 10th Edition**. John Wiley & Sons.
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.

## Latihan

1. Hitung integral fungsi pada domain !
2. Tuliskan fungsi R yang dapat melakukan integrasi Riemann dengan aturan titik kiri !
3. Buatlah sebuah fungsi R yang dapat melakukan integrasi adaptif menggunakan metode Simpson 1/3 !
4. Kerjakan kembali soal 3 dengan menggunakan metode Simpson 3/8! (**Note**: pembaca dapat melakukan pencarian algoritma di internet dan mentrasformasikannya menjadi sintaks R)
5. Fungsi monte\_int() hanya mampu melakukan integrasi pada domain positif. Buatlah algoritma baru sehingga metode ini dapat melakukan integrasi pada domain negatif !

# Persamaan Diferensial

Chapter ini dalam proses pembuatan

# Analisis Data

Pada Chapter 11, kita akan membahas mengenai cara melakukan analisis data pada R. Pada *Chapter* ini penulis akan memperkenalkan fungsi-fungsi yang ada pada R yang dapat membantu kita menganalisis data dan melakukan sejumlah uji statistik.

Pada *Chapter* ini kita tidak akan berfokus pada persamaan-persamaan matematika yang menjadi dasar suatu uji statistik. *Chapter* ini menitik beratkan pada bagaimana pembaca dapat melakukan sejumlah uji statistik pada R dan gambaran metode yang digunakan.

## Import Data

Kita dapat melakukan import data dalam berbagai format pada R. Namun, pada *sub-chapter* ini hanya akan dibahas bagaimana cara mengimport data dari file dengan format .csvdan .txt. Secara umum fungsi-fungsi yang digunakan untuk membaca data pada file dengan format tersebut adalah sebagai berikut:

read.table(file, header = FALSE, sep = "", dec = ".",  
 stringsAsFactors = default.stringsAsFactors())  
  
read.csv(file, header = TRUE, sep = ",", dec = ".")  
  
read.csv2(file, header = TRUE, sep = ";", dec = ",")  
  
read.delim(file, header = TRUE, sep = "\t", dec = ".")  
  
read.delim2(file, header = TRUE, sep = "\t", dec = ",")

**Catatan:**

* file : lokasi dan nama file yang akan dibaca diakhiri dengan format file. Secara *default* fungsi akan membaca file yang ada pada *working directory*. Untuk mengetahui lokasi *working directory*, jalankan fungsi getwd(). Salin file yang akan dibaca pada lokasi *working directory*.
* header : nilai logik yang menunjukkan apakah baris pertama pada file yang dibaca akan dibaca sebagai nama kolom.
* sep : simbol yang menujukkan pemisah antar data. Pemisah antar data dapat berupa “",”;“,”.", dll.
* dec : simbol yang menujukkan desimal. Pemisah desimal dapat berupa “.” atau “,”.
* stringsAsFactors : nilai logik yang menunjukkan apakah jenis data string akan dikonversi menjadi factor.

Kelima fungsi tersebut digunakan untuk membaca data tabular atau data yang disusun kedalam format tabel. Fungsi read.table() merupakan bentuk umum dari keempat fungsi lainnya. Fungsi tersebut dapat digunakan untuk membaca data dalam kedua format yang telah disebutkan sebelumnya. Fungsi lainnya lebih spesifi, dimana fungsi read.csv() dan read.csv2() digunakan untuk membaca data dengan ekstensi .csv, sedangkan read.delim() dan read.delim2() untuk membaca data dengan ekstensi .txt. Berikut adalah contoh bagaimana cara membaca data dengan nama data.csv yang ada pada *working directory* dengan pemisah antar data berupa ; dan tanda koma berupa ,:

data <- read.table(file="data.csv", sep=";", dec=",")

## Membaca Data Dari *Library*

Untuk keperluan pendidikan atau pengujian sebuah fungsi biasanya dalam sebuah *library* disediakan dataset yang siap digunakan. R melalui *library* datasets menyediakan sejumlah data yang dapat digunakan untuk berlatih menggunakan R. Berikut adalah fungsi yang digunakan untuk mengecek dataset apa saja yang tersedia pada sebuah *library*:

data(package=.packages(all.available = TRUE))

**Catatan:**

* package: nama *library* yang hendak dicek dataset yang tersedia.

Berikut adalah contoh cara melakukan pengecekan pada dataset yang tersedia pada *library* datasets:

data(package="datasets")  
  
# cek seluruh dataset dari seluruh library yg telah dimuat  
data()

## Ringkasan Data

Terdapat sejumlah fungsi yang akan pembaca sering gunakan untuk mengecek dataset yang akan pembaca analisa. Fungsi-fungsi tersebut antara lain:

* head(): mengecek (*default* 6) observasi teratas.
* tail(): mengecek (*default* 6) observasi terbawah.
* str(): mengecek struktur data atau jenis data pada masing-masing kolom. Jenis data yang ada pada R dapat berupa num (numerik), int (integer), Factor(factor), date (tanggal), dan chr (karakter atau string).
* summary(): ringkasan data.

Berikut adalah contoh penerapan fungsi-fungsi tersebut pada dataset iris:

# cek 10 observasi teratas  
head(iris, 10)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa  
## 7 4.6 3.4 1.4 0.3 setosa  
## 8 5.0 3.4 1.5 0.2 setosa  
## 9 4.4 2.9 1.4 0.2 setosa  
## 10 4.9 3.1 1.5 0.1 setosa

# cek 10 observasi terbawah  
tail(iris, 10)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 141 6.7 3.1 5.6 2.4 virginica  
## 142 6.9 3.1 5.1 2.3 virginica  
## 143 5.8 2.7 5.1 1.9 virginica  
## 144 6.8 3.2 5.9 2.3 virginica  
## 145 6.7 3.3 5.7 2.5 virginica  
## 146 6.7 3.0 5.2 2.3 virginica  
## 147 6.3 2.5 5.0 1.9 virginica  
## 148 6.5 3.0 5.2 2.0 virginica  
## 149 6.2 3.4 5.4 2.3 virginica  
## 150 5.9 3.0 5.1 1.8 virginica

# cek struktur data  
str(iris)

## 'data.frame': 150 obs. of 5 variables:  
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
## $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

# ringkasan data  
summary(iris)

## 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   
##   
##   
##

Fungsi-fungsi lainnya yang dapat digunakan untuk melakukan analisis statistika deskriptif adalah sebagai berikut:

* mean() : menghitung nilai rata-rata variabel numerik.
* sd() : menghitung simpangan baku variabel numerik.
* var() : menghitung varians variabel numerik.
* median() : menghitung median suatu variabel numerik.
* range() : memperoleh nilai minimum dan maksimum suatu variabel numerik.
* IQR() : memperoleh nilai jarak antar kuartil.
* quantile() : memperoleh kuantil variabel numerik.

Berikut adalah contoh penerapan fungsi-fungsi tersebut:

attach(airquality)  
  
# rata-rata konsentrasi ozon  
mean(Ozone, na.rm = TRUE)

## [1] 42.12931

# median konsentrasi ozon  
median(Ozone, na.rm = TRUE)

## [1] 31.5

# simpangan baku konsentrasi ozon  
sd(Ozone, na.rm = TRUE)

## [1] 32.98788

# varians konsentrasi ozon  
var(Ozone, na.rm = TRUE)

## [1] 1088.201

# range konsentrasi ozon  
range(Ozone, na.rm = TRUE)

## [1] 1 168

# IQR konsentrasi ozon  
IQR(Ozone, na.rm = TRUE)

## [1] 45.25

# kuartil 1, 2 dan 3 konsentrasi ozon  
quantile(Ozone, probs = c(0.25, 0.5, 0.75), na.rm = TRUE)

## 25% 50% 75%   
## 18.00 31.50 63.25

detach(airquality)

## Uji Normalitas Data Tunggal

Pada analisis statistik inferensial khususnya pada pengujian hipotesis, asumsi normalitas merupakan sesuatu yang harus terpenuhi jika prosedur uji yang digunakan merupakan prosedur uji parametrik. Terdapat dua buah cara untuk melakukan uji tersebut, antara lain:

1. Metode grafis (qq-plot, ECDF, plot densitas, histogram, dan boxplot).
2. Metode matematis (Shapiro-Wilk, Cramer-von Mises, Shapiro-Francia, Anderson-Darling, Liliefors, Pearson Chi-square, dll).

Pada *Chapter* ini, kita akan berfokus pada uji matematis karena cara pengujian dengan menggunakan metode grafis telah penulis jabarkan pada *Chapter* visualisasi data.

Metode uji normalitas yang sering digunakan pada R adalah metode Shapiro-Wilk. Metode ini merupakan metode uji yang memiliki power yang besar khusunya untuk ukuran sampel yang relatif kecil. Versi awal metode ini terbatas dengan jumlah sampel 3 sampai 50 sampel. Versi selanjutnya mengalami modifikasi sehingga dapat menangani sampel sampai dengan 5000 sampel bahkan lebih.

Untuk melakukan uji SHapiro-Wilk pada R, pembaca dapat menggunakan fungsi shapiro.test(). Format fungsi tersebut adalah sebagai beriku:

shapiro.test(x)

**Catatan:**

* x : vektor numerik.

Untuk lebih memahami impelementasi fungsi tersebut pada data, berikut adalah contoh penerapan fungsi tersebut untuk menguji normalitas distribusi konsentrasi ozon pada dataset airquality:

shapiro.test(airquality$Ozone)

##   
## Shapiro-Wilk normality test  
##   
## data: airquality$Ozone  
## W = 0.87867, p-value = 2.79e-08

Berdasarkan hasil uji diperoleh nilai p-value < 0,05, sehingga ditolak dan dapat disimpulkan bahwa distribusi konsentrasi ozon tidak mengikuti distribusi normal. Untuk lebih memahami prosedur pengujian normalitas distribusi suatu data pembaca dapat membaca lebih lanjut pada tautan [Environmental Data Modelin](https://environmental-data-modeling.netlify.com/tutorial/11_uji_hipotesis/#11-4-uji-asumsi-normalitas-distribusi-data).

## Uji Rata-Rata Satu dan Dua Sampel

Uji rata-rata satu sampel merupakan uji statistik untuk menguji apakah rata-rata suatu sampel berasal dari suatu populasi yang telah diketahui nilai rata-ratanya. Sedangkan uji rata-rata untuk dua populasi dilakukan untuk menguji apakah kedua selisis rata-rata populasi tersebut bernilai nol yang menujukkan bahwa kedua populasi tersebut memiliki nilai rata-rata yang sama. Uji rata-rata dua populasi dapat dilakukan untuk sampel independen (contoh: uji rata-rata performa dua buah IPAL) dan berpasangan (contoh: uji rata-rata input dan output IPAL).

Untuk melakukan uji rata-rata pada R dapat digunakan fungsi t.test() untuk uji parametrik dan wilcox.test() untuk melakukan uji non-parametrik *sign rank test*. Format fungsi-fungsi tersebut adalah sebagai berikut:

t.test(x, y = NULL,  
 alternative = c("two.sided", "less", "greater"),  
 mu = 0, paired = FALSE, var.equal = FALSE,  
 conf.level = 0.95, ...)  
  
wilcox.test(x, y = NULL,  
 alternative = c("two.sided", "less", "greater"),  
 mu = 0, paired = FALSE, conf.level = 0.95, ...)

**Catatan:**

* x,y : vektor numerik. Jika argumen x dan y diisikan maka uji hipotesis dilakukan untuk dua buah populasi.
* alternative: digunakan untuk menentukan jenis uji hipotesis apakah satu sisi(“less” dan “greater”), atau dua sisi (“two.sided”).
* mu : nilai rata-rata populasi atau nilai rata-rata selisih antar populasi jika dilakukan uji hipotesis terhadap dua populasi. Secara default nilainya 0.
* paired : nilai logikal yang menentukan apakah uji dua populasi digunakan untuk sampel berpasangan (TRUE) atau tidak (FALSE).
* var.equal : nilai logikal yang menunjukkan apakah varians kedua populasi diasumsikan sama atau berbeda.
* conf.level : tingkat kepercayaan. Secara default tingkat kepercayaan yang digunakan adalah 95%.

Berikut adalah contoh penerapan fungsi tersebut untuk uji hipotesis satu dan dua populasi:

# Uji hipotesis konsentrasi ozon = 40 ppm  
# parametrik  
t.test(x=airquality$Ozone, alternative = "two.sided",  
 mu = 40)

##   
## One Sample t-test  
##   
## data: airquality$Ozone  
## t = 0.69521, df = 115, p-value = 0.4883  
## alternative hypothesis: true mean is not equal to 40  
## 95 percent confidence interval:  
## 36.06240 48.19622  
## sample estimates:  
## mean of x   
## 42.12931

# nonparametrik  
wilcox.test(x=airquality$Ozone, alternative = "two.sided",  
 mu = 40)

##   
## Wilcoxon signed rank test with continuity correction  
##   
## data: airquality$Ozone  
## V = 3188, p-value = 0.6826  
## alternative hypothesis: true location is not equal to 40

# Uji hipotesis dua populasi  
dni3 <- dimnames(iris3)  
ii <- data.frame(matrix(aperm(iris3, c(1,3,2)), ncol = 4,  
 dimnames = list(NULL, sub(" L.",".Length",  
 sub(" W.",".Width", dni3[[2]])))),  
 Species = gl(3, 50, labels = sub("S", "s", sub("V", "v", dni3[[3]]))))  
# parametrik  
t.test(x=iris$Sepal.Length[iris$Species=="setosa"],   
 y=ii$Sepal.Length[iris$Species=="versicolor"])

##   
## Welch Two Sample t-test  
##   
## data: iris$Sepal.Length[iris$Species == "setosa"] and ii$Sepal.Length[iris$Species == "versicolor"]  
## t = -10.521, df = 86.538, p-value < 2.2e-16  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -1.1057074 -0.7542926  
## sample estimates:  
## mean of x mean of y   
## 5.006 5.936

# nonparametrik  
wilcox.test(x=iris$Sepal.Length[iris$Species=="setosa"],   
 y=ii$Sepal.Length[iris$Species=="versicolor"])

##   
## Wilcoxon rank sum test with continuity correction  
##   
## data: iris$Sepal.Length[iris$Species == "setosa"] and ii$Sepal.Length[iris$Species == "versicolor"]  
## W = 168.5, p-value = 8.346e-14  
## alternative hypothesis: true location shift is not equal to 0

Fungsi t.test() akan menghasilkan output berupa nilai t uji, derajat kebebasan (df), nilai p-value, rentang estimasi nilai rata-rata berdasarkan tingkat kepercayaan yang digunakan, serta estimasi nilai rata-rata sampel. Fungsi wilcox.test() akan menghasilkan dua buah output yaitu nilai W dan p-value berdasarkan nilai W yang dihasilkan.

## Korelasi Antar Variabel

Pada sebuah analisa, kita sering kali tertarik untuk menganalisa hubungan atau korelasi antara satu variabel terhadap variabel lainnya. Pengamatan adanya korelasi antar variabel dapat dilakukan melalui visualisasi menggunakan *scatterplot* dan perhitungan matematis menggunakan metode Pearson untuk metode parametrik dan metode rangking Spearman dan Kendall untuk metode non-parametrik. Pada *Chapter* ini kita akan berfokus untuk melakukan uji korelasi menggunakan R menggunakan metode matematis.

Pada R uji korelasi dapat dilakukan dengan menggunakan fungsi cor.test(). Format fungsi tersebut adalah sebagai berikut:

cor.test(x, y,  
 alternative = c("two.sided", "less", "greater"),  
 method = c("pearson", "kendall", "spearman"),  
 conf.level = 0.95)

**Catatan:**

* x,y : vektor numerik.
* alternative: digunakan untuk menentukan jenis uji hipotesis apakah satu sisi(“less” dan “greater”), atau dua sisi (“two.sided”).
* method : metode perhitungan korelasi yang digunakan.
* conf.level : tingkat kepercayaan. Secara default tingkat kepercayaan yang digunakan adalah 95%.

Berikut adalah penerapan fungsi cor.test() berdasarkan metode-metode yang telah disediakan pada fungsi tersebut:

# Pearson  
cor.test(x = airquality$Ozone, y = airquality$Solar.R,  
 alternative = "two.sided",  
 method = "pearson")

##   
## Pearson's product-moment correlation  
##   
## data: airquality$Ozone and airquality$Solar.R  
## t = 3.8798, df = 109, p-value = 0.0001793  
## alternative hypothesis: true correlation is not equal to 0  
## 95 percent confidence interval:  
## 0.173194 0.502132  
## sample estimates:  
## cor   
## 0.3483417

# Kendall  
cor.test(x = airquality$Ozone, y = airquality$Solar.R,  
 alternative = "two.sided",  
 method = "kendall")

##   
## Kendall's rank correlation tau  
##   
## data: airquality$Ozone and airquality$Solar.R  
## z = 3.7096, p-value = 0.0002076  
## alternative hypothesis: true tau is not equal to 0  
## sample estimates:  
## tau   
## 0.2403194

# Spearman  
cor.test(x = airquality$Ozone, y = airquality$Solar.R,  
 alternative = "two.sided",  
 method = "spearman")

## Warning in cor.test.default(x = airquality$Ozone, y = airquality$Solar.R, :  
## Cannot compute exact p-value with ties

##   
## Spearman's rank correlation rho  
##   
## data: airquality$Ozone and airquality$Solar.R  
## S = 148560, p-value = 0.0001806  
## alternative hypothesis: true rho is not equal to 0  
## sample estimates:  
## rho   
## 0.3481865

Berdasarkan output yang dihasilkan, metode Pearson menghasilkan output berupa nilai t uji, derajat kebebasan, nilai p-value, rentang estimasi nilai korelasi berdasarkan tingkat kepercayaan, dan estimasi nilai korelasi. Metode Kendall dan Spearman disisi lai menghasilkan output berupa nilai z uji dan S untuk masing-masing metode serta nilai p-value berdasarkan nilai statistika uji dan estimasi koefisien korelasi.

## Analisis Varians

Pada *sub-Chapter* sebelumnya penulis telah menjelaskan uji rata-rata untuk satu sampel dan dua sampel. Pada kenyataannya dalam sebuah percobaan laboratorium, kita tidak hanya membandingkan dua buah grup sampel saja, namun beberapa grup dan sejumlah faktor. Untuk menganalisa apakah variasi perlakuan pada kelompok sampel akan memberikan hasil yang berbeda-beda pada rata-rata tiap grup atau tidak diperlukan analisis varians untuk menganilisa variasi perlakuan atau faktor pada masing-masing grup. Analisis varians dapat dilakukan baik untuk satu faktor maupun dua faktor atau lebih. Untuk melakukannya pada R, kita dapat menggunakan fungsi aov() untuk analisis varians dengan metode parametrik dan kruskal.test() untuk analisis varians dengan menggunakan metode nonparametrik. Berikut adalah format kedua fungsi tersebut:

aov(formula, data = NULL)  
  
kruskal.test(formula, data)

**Catatan:**

* formula : formula model yang digunakan.
* data: dataset yang akan digunakan.

Berikut adalah contoh penerapan kedua fungsi tersebut untuk melihat apakah terdapat beda pada rata-rata konsentrasi bulanan ozon menggunakan dataset airquality:

summary(aov(Ozone~Month, airquality))

## Df Sum Sq Mean Sq F value Pr(>F)   
## Month 1 3387 3387 3.171 0.0776 .  
## Residuals 114 121756 1068   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
## 37 observations deleted due to missingness

kruskal.test(Ozone~Month, airquality)

##   
## Kruskal-Wallis rank sum test  
##   
## data: Ozone by Month  
## Kruskal-Wallis chi-squared = 29.267, df = 4, p-value = 6.901e-06

Berdasarkan hasil yang diperoleh diketahui bahwa rata-rata konsentrasi bulanan ozon tidak sama tiap bulannya atau minimal terdapat satu bulan dimana konsentrasi ozonnya berbeda secara signifikan dengan konsentrasi ozon pada bulan-bulan lainnya. Untuk lebih memahami terkait analisis varians pada R dan cara membaca output kedua fungsi tersebut, pembaca dapat membaca tulisan pada halaman situs [sthda](http://www.sthda.com/english/wiki/comparing-means-in-r).

## Analisis Komponen Utama

Analisis komponen utama menggunakan transformasi ortogonal (umumnya nilai singular atau dekomposisi nilai eigen) untuk mengubah seperangkat variabel pengamatan yang mungkin berkorelasi menjadi seperangkat variabel tidak berkorelasi (ortogonal) yang disebut komponen utama. Transformasi didefinisikan sedemikian rupa sehingga komponen utama pertama memiliki varians setinggi mungkin (menyumbang variabilitas pada data sebanyak mungkin), dan masing-masing komponen berikutnya pada gilirannya memiliki varians tertinggi yang mungkin di bawah kendala, dimana komponen tersebut menjadi ortogonal ke komponen sebelumnya.

Dalam R, analisis komponen utama umumnya dilakukan dengan fungsi prcomp (). Format fungsi tersebut adalah sebagai berikut:

prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE,  
 tol = NULL)

**Catatan:**

* x : data frame atau matriks kompleks numerik.
* retx : nilai logik yang mengindikasikan apakah variabel hasil rotasi perlu ditampilkan.
* center : nilai logik yang mengidikasikan apakah variabel perlu dilakukan pergeseran sehingga nilai rata-ratanya berpusat pada nilai nol.
* scale : nilai logik yang mengidikasikan apakah variabel perlu dilakukan penskalaan sebelum dilakukan analisis.
* tol : nilai toleransi yang menunjukkan batas nilai bagi komponen yang akan dipertahankan. Komponen yang dihilangkan jika simpangan bakunya kurang dari atau sama dengan tol x simpangan baku pc1.

Untuk memahami penerapan fungsi tersebut, kita akan melakukan simulasi menggunakan dataset iris. Output yang dihasilkan di bawah ini menunjukkan bagaimana empat variabel numerik ditransformasikan menjadi empat komponen utama. Penskalaan data mungkin tidak diperlukan dalam kasus ini, karena keempat pengukuran memiliki unit yang sama dan besarnya sama. Namun, penskalaan umumnya merupakan praktik yang baik.

iris\_use <- iris[,-5] # menghilangkan variabel non-numerik  
iris\_pca <- prcomp(iris\_use, scale. = TRUE)  
iris\_pca

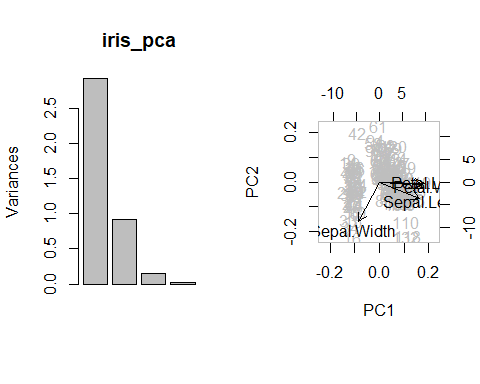
## Standard deviations (1, .., p=4):  
## [1] 1.7083611 0.9560494 0.3830886 0.1439265  
##   
## Rotation (n x k) = (4 x 4):  
## PC1 PC2 PC3 PC4  
## Sepal.Length 0.5210659 -0.37741762 0.7195664 0.2612863  
## Sepal.Width -0.2693474 -0.92329566 -0.2443818 -0.1235096  
## Petal.Length 0.5804131 -0.02449161 -0.1421264 -0.8014492  
## Petal.Width 0.5648565 -0.06694199 -0.6342727 0.5235971

Fungsi summary memberikan proporsi varians total yang dikaitkan dengan masing-masing komponen utama, dan proporsi kumulatif ketika masing-masing komponen ditambahkan. Kita melihat bahwa dua komponen pertama berperan lebih dari 95% dari total varians.

summary(iris\_pca)

## Importance of components:  
## PC1 PC2 PC3 PC4  
## Standard deviation 1.7084 0.9560 0.38309 0.14393  
## Proportion of Variance 0.7296 0.2285 0.03669 0.00518  
## Cumulative Proportion 0.7296 0.9581 0.99482 1.00000

Histogram (hasil plot dalam analisis prcomp) secara grafis merekapitulasi proporsi varian yang disumbangkan oleh masing-masing komponen utama, sementara biplot menunjukkan bagaimana variabel awal diproyeksikan pada dua komponen utama pertama (Gambar 76). Ini juga menunjukkan koordinat dari masing-masing sampel dalam ruang (PC1, PC2). Satu spesies iris (yang berubahmenjadi setosa dari analisis kluster di bawah ini) secara jelas dipisahkan dari dua spesies lain dalam ruang koordinat ini.



Gambar 76: Analisis komponen utama data iris.

Untuk informasi lebih lanjut terkait metode analisis komponen utama, pembaca dapat membacanya pada laman [Little Book of R for Multivariate Analysis](https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/).

## Analisis Cluster

Analisis cluster mencoba untuk mengurutkan satu set objek ke dalam kelompok (cluster) sedemikian rupa sehingga objek dalam cluster yang sama lebih mirip satu sama lain dibandingkan objek pada cluster lainnya. Ini digunakan untuk analisis eksplorasi melalui proses penambangan data (*data mining*) di banyak bidang, seperti bioinformatika, biologi evolusi, analisis gambar, lingkungan, dan pembelajaran mesin.

Menurut Wikipedia: “Analisis Cluster itu sendiri bukanlah salah satu algoritma spesifik, tetapi tugas umum yang harus dipecahkan. Ini dapat dicapai dengan berbagai algoritma yang berbeda secara signifikan dalam pengertian mereka tentang apa yang merupakan sebuah cluster dan bagaimana cara menemukannya secara efisien. Gagasan populer mengenai cluster termasuk kelompok dengan jarak rendah di antara anggota cluster, area padat ruang data, interval atau distribusi statistik tertentu. Algoritma pengelompokan dan pengaturan parameter yang sesuai (termasuk nilai-nilai seperti fungsi jarak yang akan digunakan, ambang kepadatan atau jumlah cluster yang diharapkan) tergantung pada dataset individual dan tujuan penggunaan hasil. Analisis cluster seperti itu bukan tugas otomatis, tetapi proses berulang penemuan pengetahuan yang melibatkan *trial and error*. Seringkali diperlukan untuk memodifikasi preprocessing dan parameter sampai hasilnya mencapai properti yang diinginkan.

### Analisis Cluster Menggunakan Algoritma Pengelompokan Hierarkis Aglomeratif

Hierarchical clustering membangun hierarki cluster, di mana metrik hierarki adalah suatu ukuran ketidaksamaan antar cluster. Menurut halaman bantuan untuk hclust(), metode pengelompokan hierarkis aglomeratif, “Fungsi ini melakukan analisis hierarki cluster menggunakan seperangkat ketidaksamaan untuk n objek yang dikelompokkan. Awalnya, masing-masing objek ditugaskan ke cluster sendiri dan kemudian algoritma melanjutkan secara iteratif, pada setiap tahap bergabung dengan dua cluster yang paling mirip, terus sampai hanya ada satu cluster. Pada setiap tahap, jarak antar kluster dihitung ulang dengan formula pembaruan ketidaksamaan Lance Williams sesuai dengan metode pengelompokan tertentu yang digunakan.”Ada tujuh metode aglomerasi yang tersedia, dengan lengkap — yang mencari kluster kompak, bola — sebagai default. Format umum fungsi hclust() adalah sebagai berikut:

hclust(d, method = "complete", members = NULL)

**Catatan:**

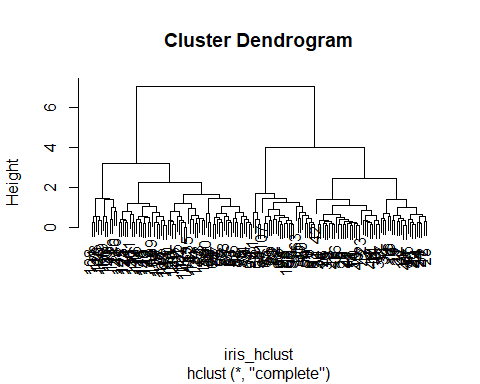
* d : struktur ketidaksamaan yang dihasilkan dengan fungsi dist.
* method : metode alglomerasi yang digunakan. Metode yang dapat digunakan antara lain: “ward.D”, “ward.D2”, “single”, “complete”, “average” (= UPGMA), “mcquitty” (= WPGMA), “median” (= WPGMC) atau “centroid” (= UPGMC)
* members: NULL atau vektor dengan ukuran sama dengan d. Untuk info lebih lanjut jalankan sintaks ?hclust.

Untuk memahami penerapan fungsi hclust(), kita akan kembali menggunakan data iris\_use. Berikut adalah sintaks yang digunakan:

# menghitung jarak antar observasi  
iris\_hclust <- dist(iris\_use)  
  
# Pembentukan Cluster  
hc <- hclust(iris\_hclust)  
hc

##   
## Call:  
## hclust(d = iris\_hclust)  
##   
## Cluster method : complete   
## Distance : euclidean   
## Number of objects: 150

Visualisasi cluster dibentuk melalui dendogram yang ditampilkan pada Gambar 77.



Gambar 77: Analisis pengelompokan hierarkis alglomeratif data iris.

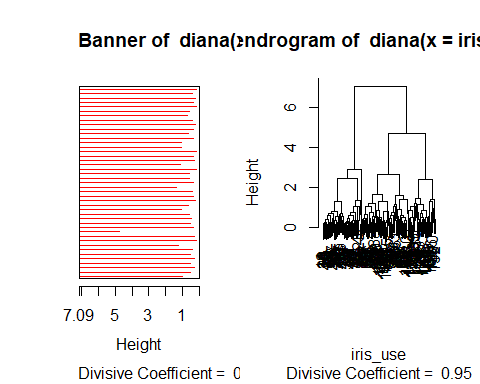
### Pengelompokan Hierarkis Divisif

Menurut halaman bantuan diana() (*DIvisive ANAlysis Clustering*) dalam *library* cluster, “Algoritma diana membangun hierarki pengelompokan, dimulai dengan satu kluster besar yang berisi semua n pengamatan. Cluster dibagi sampai masing-masing cluster hanya berisi satu pengamatan. Pada setiap tahap, cluster dengan diameter terbesar dipilih. Format fungsi diana() adalah sebagai berikut:

diana(x, metric = "euclidean", stand = FALSE)

**Catatan:**

* x : struktur ketidaksamaan yang dihasilkan dengan fungsi dist atau data frame.
* metric : karakter string yang menyatakan metode pengukuran jarak yang digunakan. Metode pengukuran jarak dapat berupa “euclidean” dan “manhattan”.
* stand: vektor logik yang menyatakan apakah data akan dilakukan standardisasi terlebih dahulu sebelum dilakukan analisis.



Gambar 78: Analisis pengelompokan divisif data iris.

### Pengelompokan Menggunakan Algoritma K-Mean

k-means melakukan pengelompokan n pengamatan ke dalam k cluster di mana setiap pengamatan akan tergabung dengan pusat cluster terdekat. Pengguna harus menentukan jumlah pusat (cluster) yang diinginkan sebagai output. Untuk melakukan pengelompokan dengan algoritma k-means pada R dapat menggunakan fungsi kmeans(). Format fungsi tersebut secara umum adalah sebagai berikut:

kmeans(x, centers, iter.max = 10,  
 algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",  
 "MacQueen"))

**Catatan:**

* x : data frame.
* centers : jumlah cluster yang ingin di buat.
* iter.max: jumlah iterasi maksimum yang diijinkan.
* algorithm : algoritma pengelompokan yang digunakan. Untuk informasi lebih lanjut jalankan sintaks ?kmeans.

iris\_kmeans <- kmeans(iris\_use, centers = 3)  
iris\_kmeans

## K-means clustering with 3 clusters of sizes 33, 96, 21  
##   
## Cluster means:  
## Sepal.Length Sepal.Width Petal.Length Petal.Width  
## 1 5.175758 3.624242 1.472727 0.2727273  
## 2 6.314583 2.895833 4.973958 1.7031250  
## 3 4.738095 2.904762 1.790476 0.3523810  
##   
## Clustering vector:  
## [1] 1 3 3 3 1 1 1 1 3 3 1 1 3 3 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 3 3 1 1 1 3  
## [36] 1 1 1 3 1 1 3 3 1 1 3 1 3 1 1 2 2 2 2 2 2 2 3 2 2 3 2 2 2 2 2 2 2 2 2  
## [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2 2 2 2  
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
## [141] 2 2 2 2 2 2 2 2 2 2  
##   
## Within cluster sum of squares by cluster:  
## [1] 6.432121 118.651875 17.669524  
## (between\_SS / total\_SS = 79.0 %)  
##   
## Available components:  
##   
## [1] "cluster" "centers" "totss" "withinss"   
## [5] "tot.withinss" "betweenss" "size" "iter"   
## [9] "ifault"

# menghitung lokasi pusat cluster  
ccent = function(cl) {  
 f = function(i) colMeans(iris\_use[cl==i,])  
 x = sapply(sort(unique(cl)), f)  
 colnames(x) = sort(unique(cl))  
 return(x)  
}  
  
ccent(iris\_kmeans$cluster)

## 1 2 3  
## Sepal.Length 5.1757576 6.314583 4.738095  
## Sepal.Width 3.6242424 2.895833 2.904762  
## Petal.Length 1.4727273 4.973958 1.790476  
## Petal.Width 0.2727273 1.703125 0.352381

### Pengelompokan Menggunakan Algoritma PAM

pam mem-partisi data menjadi k cluster di sekitar medoid. Medoid dari set data yang terbatas merupakan titik data dengan nilai ketidaksamaan rata-rata untuk semua titik data adalah minimum. Hal tersebut menujukkan bahwa medoid merupakan pusat dari set cluster. Menurut halaman bantuan pam(), pendekatan k-medoid lebih kuat daripada pendekatan k-means “karena meminimalkan jumlah ketidaksamaan daripada jumlah jarak euclidean kuadrat”. Format umum fungsi pam() adalah sebagai berikut:

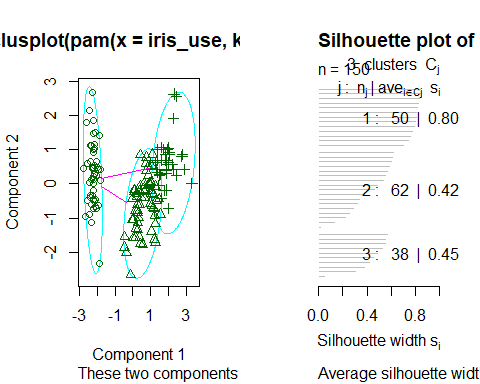
pam(x, k, diss = inherits(x, "dist"),  
 metric = c("euclidean", "manhattan"))

**Catatan:**

* x : data frame.
* k : jumlah cluster yang ingin di buat.
* method: metode perhitungan jarak yang digunakan. Untuk informasi lebih lanjut jalankan sintaks ?pam.

library(cluster)  
iris\_pam <- pam(iris\_use, k=3)  
iris\_pam

## Medoids:  
## ID Sepal.Length Sepal.Width Petal.Length Petal.Width  
## [1,] 8 5.0 3.4 1.5 0.2  
## [2,] 79 6.0 2.9 4.5 1.5  
## [3,] 113 6.8 3.0 5.5 2.1  
## Clustering vector:  
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
## [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
## [71] 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3  
## [106] 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3  
## [141] 3 3 2 3 3 3 2 3 3 2  
## Objective function:  
## build swap   
## 0.6709391 0.6542077   
##   
## Available components:  
## [1] "medoids" "id.med" "clustering" "objective" "isolation"   
## [6] "clusinfo" "silinfo" "diss" "call" "data"



Gambar 79: Analisis pengelompokan metode pam data iris.

## Referensi

1. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press.
2. Coqhlan, A. Tanpa Tahun. **Using R for Multivariate Analysis**. <https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/src/multivariateanalysis.html#principal-component-analysis>.
3. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung
4. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.
5. Rosidi, M. 2019. **Uji Hipotesis**. <https://environmental-data-modeling.netlify.com/tutorial/11_uji_hipotesis/>.
6. STHDA. Tanpa Tahun. **Comparing Means in R**. <http://www.sthda.com/english/wiki/comparing-means-in-r>.

# Pemodelan Data

Pada Chapter 12, kita akan membahas cara membentuk model statistik menggunakan R. Terdapat 2 buah jenis model yang akan dibahas pada *Chapter* ini, yaitu: regresi dan klasifikasi. Untuk informasi terkait cara untuk melakukan inferensi berdasarkan hasil yang diperoleh dan cara untuk melakukan prediksi menggunakan model yang terbentuk tidak akan dijelaskan dalam buku ini. Pembaca dapat membaca lebih lanjut pada referensi berikut:

* [Introduction to Probability and Statistics Using R](http://ipsur.r-forge.r-project.org/book/download/IPSUR.pdf)
* [STHDA](http://www.sthda.com/english/)
* [An Introduction to Statistical Learning](https://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf)

## Regresi Linier

Regresi linier merupakan model sederhana yang paling sering dibahas dalam buku-buku statistika. Modelnya cukup sederhana dimana kita berusaha membentuk model dengan pendekatan garis linier dengan prinsip meminimalkan jumlah kuadrat residual pada data. Model yang tebentuk akan menghasilkan dua buah nilai yaitu nilai konstanta (titik potong sumbu y) dan nilai slope kurva. Model yang terbentuk secara umum haruslah memenuhi asumsi dasar model linier berikut:

1. **Asumsi liniearitas**: kurva relasi yang terbentuk antara variabel independen terhadap variabel dependen harus linier. Asumsi ini dapat dipelajari melalui plot residual terhadap nilai *fitted value*. Jika asumsi liniearitas terpenuhi, maka titik-titik residual yang di plotkan akan membentuk pola acak. Jika pada plot yang dihasilkan terbentuk pola tidak linear maka transformasi data pada variabel prediktor atau independen diperlukan.
2. **Error atau residu berdristribusi normal**: normalitas error di cek menggunakan qq-plot atau uji normalitas yang telah dibahas pada Chapter 11.4.
3. ***Outlier* dan *high influence point***: kedua pengamatan tersebut dideteksi melalui qq-plot, plot residual terhadap nilai *fitted value*, dan plot *residuals vs leverage*. Jika *outlier* terjadi akibat adanya error selama pengukuran maka *outlier* dapat dihilangkan.
4. **Error bersifat independen**: independensi residual dapat dideteksi melaui plot korelasi serial dengan mengeplotkan vs .
5. **Varians bersifat konstan**: Varians bersifat konstan dicek melalui plot **square root standardize residual vs fitted value**. Pada kasus dimana varians tidak bersifat konstan, kita dapat memberikan bobot pada model yang akan kita bentuk (*weighted least square*), dimana bobot yang diberikan proporsional dengan invers varians.
6. **multikolinearitas**: tidak ada variabel dependen yang saling berfkorelasi. Multikolinearitas dapat dideteksi melalui plot matriks korelasi. Pada model adanya kolinearitas ditunjukkan dari nilai *variance inflation factor* (VIF) yang tinggi. Secara umum nilai VIF terkecil sebesar 1 dan jika kolinearitas terjadi nilainya dapat lebih besar dari 5 atau 10. Untuk mengatasi kolinearitas pada model dapat dilakukan dengan dua cara, yaitu: mengeluarkan variabel dengan nilai VIF yang tinggi pada model atau menggabungkan dua variabel prediktor yang saling berkorelasi menjadi satu variabel baru.

Pembentukan model linier pada R dilakukan dengan menggunakan fungsi lm(). Format umum fungsi tersebut adalah sebagai berikut:

lm(formula, data, subset, weights)

**Catatan:**

* formula : formula model yang hendak dibentuk.
* data: data yang digunakan untuk membentuk model.
* subset : subset data yang akan digunakan dalam pembentukan model.
* weight : nilai pembobotan dalam pembentukan model.

### Regrasi Linier Sederhana (*Simple Linear Regression*)

Pada Chapter 12.1.1 akan diberikan contoh pembentukan model linier sederhana menggunakan dataset Boston dari *library* MASS dengan jumlah observasi sebesar 506 observasi. Pada contoh kali ini kita akan mencoba membentuk model dengan variabel dependen berupa medv (median harga rumah) dan variabel independen berupa lstat (persen rumah tangga dengan status ekonomi menengah ke bawah). Berikut adalh sintaks untuk membentuk model tersebut:

library(MASS)

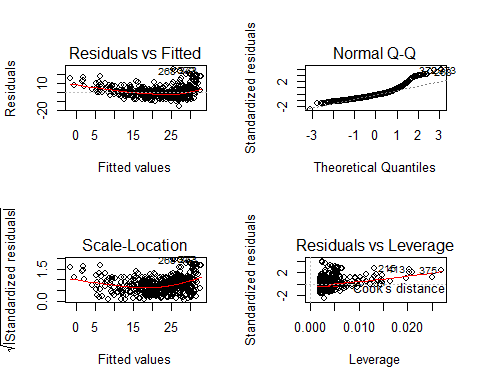
lm.fit <- lm(medv~lstat, data=Boston)  
anova(lm.fit)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## lstat 1 23244 23243.9 601.62 < 2.2e-16 \*\*\*  
## Residuals 504 19472 38.6   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(lm.fit)

##   
## Call:  
## lm(formula = medv ~ lstat, data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.168 -3.990 -1.318 2.034 24.500   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 34.55384 0.56263 61.41 <2e-16 \*\*\*  
## lstat -0.95005 0.03873 -24.53 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 6.216 on 504 degrees of freedom  
## Multiple R-squared: 0.5441, Adjusted R-squared: 0.5432   
## F-statistic: 601.6 on 1 and 504 DF, p-value: < 2.2e-16

Plot residual disajikan pada Gambar 80.



Gambar 80: Analisis residual model linier medv vs lstat pada dataset Boston.

Berdasarkan hasil plot dapat dilihat bahwa seluruh asumsi model linier tidak terpenuhi. Selain melalui plot residual, uji asumsi model linier dapat juga dilakukan secara matematis. Berikut adalah sintaks yang digunakan:

# error berdistribusi normal   
# (data tidak berdistribusi normal)  
shapiro.test(residuals(lm.fit))

##   
## Shapiro-Wilk normality test  
##   
## data: residuals(lm.fit)  
## W = 0.87857, p-value < 2.2e-16

# varians bersifat konstan   
# (varians tidak konstan)  
library(lmtest)  
bptest(lm.fit)

##   
## studentized Breusch-Pagan test  
##   
## data: lm.fit  
## BP = 15.497, df = 1, p-value = 8.262e-05

# error bersifat independen  
# (error tidak bersifat independen)  
dwtest(lm.fit, alternative = "two.sided")

##   
## Durbin-Watson test  
##   
## data: lm.fit  
## DW = 0.8915, p-value < 2.2e-16  
## alternative hypothesis: true autocorrelation is not 0

# deteksi outlier (stdres > 2)  
sres <- rstandard(lm.fit)  
sres[which(abs(sres)>2)] # nomor observasi outlier

## 99 142 162 163 164 167 181   
## 2.038479 2.037877 2.758553 2.787466 3.000496 3.058318 2.002509   
## 187 196 204 205 215 225 226   
## 3.172448 2.947239 2.833127 2.933544 2.788670 2.286679 3.199842   
## 229 234 257 258 262 263 268   
## 2.559571 2.822187 2.000582 3.274417 2.488441 3.201252 3.627671   
## 281 283 284 369 370 371 372   
## 2.325519 2.307994 2.976149 2.991366 3.062883 2.945717 3.946264   
## 373 375 413 506   
## 3.847133 2.498416 2.600516 -2.443658

# influential observation  
# observasi > percentil 50  
# tidak ada observasi dengan jarak cook yang extrim  
cooksD <- cooks.distance(lm.fit)  
p50 <- qf(0.5, df1=2, df2=560-2)  
any(cooksD>p50)

## [1] FALSE

### Regresi Linier Berganda (*Multiple Linier Regression*)

Pada Chapter 12.1.2, kita akan membuat tiga buah model regresi linier. Model pertama akan menambahkan variabel age (usia bangunan) pada model sebelumnya, model kedua akan menggunakan seluruh ariabel yang ada, dan model ketiga akan melakukan pembaharuan dengan mengeluarkan variabel dengan VIF paling tinggi dari model kedua. Berikut adalah sintaks untuk membentuk ketiag model tersebut:

library(car)  
# Model pertama  
lm.fit1 <- lm(medv ~ lstat+age, data=Boston)  
anova(lm.fit1)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## lstat 1 23243.9 23243.9 609.955 < 2.2e-16 \*\*\*  
## age 1 304.3 304.3 7.984 0.004907 \*\*   
## Residuals 503 19168.1 38.1   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

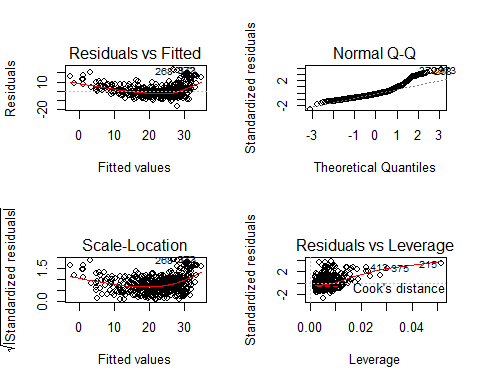
summary(lm.fit1)

##   
## Call:  
## lm(formula = medv ~ lstat + age, data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.981 -3.978 -1.283 1.968 23.158   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 33.22276 0.73085 45.458 < 2e-16 \*\*\*  
## lstat -1.03207 0.04819 -21.416 < 2e-16 \*\*\*  
## age 0.03454 0.01223 2.826 0.00491 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 6.173 on 503 degrees of freedom  
## Multiple R-squared: 0.5513, Adjusted R-squared: 0.5495   
## F-statistic: 309 on 2 and 503 DF, p-value: < 2.2e-16

vif(lm.fit1)

## lstat age   
## 1.569395 1.569395

Berdasarkan hasil perhitungan diketahui nilai VIF dari model < 10, sehingga asumsi multikolinearitas terpenuhi. Untuk asumsi lainnya dapat dicek pada plot residual yang ditampilkan pada Gambar 81.



Gambar 81: Analisis residual model linier 1 pada dataset Boston.

# Model 2  
lm.fit2 <- lm(medv~., data=Boston)  
anova(lm.fit2)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## crim 1 6440.8 6440.8 286.0300 < 2.2e-16 \*\*\*  
## zn 1 3554.3 3554.3 157.8452 < 2.2e-16 \*\*\*  
## indus 1 2551.2 2551.2 113.2984 < 2.2e-16 \*\*\*  
## chas 1 1529.8 1529.8 67.9393 1.543e-15 \*\*\*  
## nox 1 76.2 76.2 3.3861 0.0663505 .   
## rm 1 10938.1 10938.1 485.7530 < 2.2e-16 \*\*\*  
## age 1 90.3 90.3 4.0087 0.0458137 \*   
## dis 1 1779.5 1779.5 79.0262 < 2.2e-16 \*\*\*  
## rad 1 34.1 34.1 1.5159 0.2188325   
## tax 1 329.6 329.6 14.6352 0.0001472 \*\*\*  
## ptratio 1 1309.3 1309.3 58.1454 1.266e-13 \*\*\*  
## black 1 593.3 593.3 26.3496 4.109e-07 \*\*\*  
## lstat 1 2410.8 2410.8 107.0634 < 2.2e-16 \*\*\*  
## Residuals 492 11078.8 22.5   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

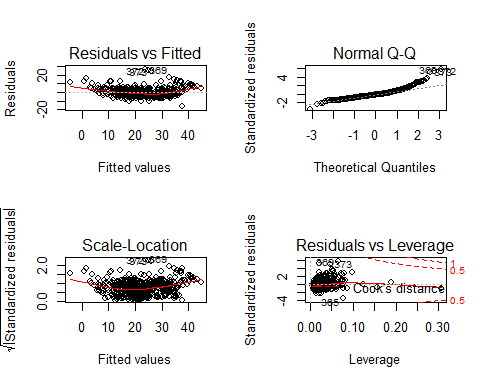
summary(lm.fit2)

##   
## Call:  
## lm(formula = medv ~ ., data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.595 -2.730 -0.518 1.777 26.199   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 3.646e+01 5.103e+00 7.144 3.28e-12 \*\*\*  
## crim -1.080e-01 3.286e-02 -3.287 0.001087 \*\*   
## zn 4.642e-02 1.373e-02 3.382 0.000778 \*\*\*  
## indus 2.056e-02 6.150e-02 0.334 0.738288   
## chas 2.687e+00 8.616e-01 3.118 0.001925 \*\*   
## nox -1.777e+01 3.820e+00 -4.651 4.25e-06 \*\*\*  
## rm 3.810e+00 4.179e-01 9.116 < 2e-16 \*\*\*  
## age 6.922e-04 1.321e-02 0.052 0.958229   
## dis -1.476e+00 1.995e-01 -7.398 6.01e-13 \*\*\*  
## rad 3.060e-01 6.635e-02 4.613 5.07e-06 \*\*\*  
## tax -1.233e-02 3.760e-03 -3.280 0.001112 \*\*   
## ptratio -9.527e-01 1.308e-01 -7.283 1.31e-12 \*\*\*  
## black 9.312e-03 2.686e-03 3.467 0.000573 \*\*\*  
## lstat -5.248e-01 5.072e-02 -10.347 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 4.745 on 492 degrees of freedom  
## Multiple R-squared: 0.7406, Adjusted R-squared: 0.7338   
## F-statistic: 108.1 on 13 and 492 DF, p-value: < 2.2e-16

vif(lm.fit2)

## crim zn indus chas nox rm age dis   
## 1.792192 2.298758 3.991596 1.073995 4.393720 1.933744 3.100826 3.955945   
## rad tax ptratio black lstat   
## 7.484496 9.008554 1.799084 1.348521 2.941491

Berdasarkan hasil perhitungan diperoleh nilai VIF untuk seluruh varaibel prediktor dalam model < 10, sehingga asumsi multikolinearitas terpenuhi. Untuk asumsi lainnya dapat dicek pada plot residual yang ditampilkan pada Gambar 82.



Gambar 82: Analisis residual model linier 2 pada dataset Boston.

Pada model ketiga, kita akan mencoba untuk melakukan pembaharuan pada model kedua dengan melakukan drop variabel dengan vif yang paling tinggi. Pada hasil perhitungan sebelumnya, variabel tax (pajak) memiliki nilai VIF yang paling tinggi, sehingga pada model ketiga variabel tersebut tidak disertakan. Terdapat dua cara untuk melakukannya berikut adalah sintaks yang digunakan:

# Model 3 (cara 1)  
lm.fit3 <- lm(medv~.-tax, data=Boston)  
  
# Model 3 (cara 2)  
lm.fit3 <- update(lm.fit2, ~.-tax)  
  
anova(lm.fit3)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## crim 1 6440.8 6440.8 280.4782 < 2.2e-16 \*\*\*  
## zn 1 3554.3 3554.3 154.7815 < 2.2e-16 \*\*\*  
## indus 1 2551.2 2551.2 111.0993 < 2.2e-16 \*\*\*  
## chas 1 1529.8 1529.8 66.6206 2.768e-15 \*\*\*  
## nox 1 76.2 76.2 3.3204 0.06903 .   
## rm 1 10938.1 10938.1 476.3247 < 2.2e-16 \*\*\*  
## age 1 90.3 90.3 3.9309 0.04796 \*   
## dis 1 1779.5 1779.5 77.4923 < 2.2e-16 \*\*\*  
## rad 1 34.1 34.1 1.4865 0.22335   
## ptratio 1 1401.2 1401.2 61.0172 3.434e-14 \*\*\*  
## black 1 611.6 611.6 26.6323 3.574e-07 \*\*\*  
## lstat 1 2388.0 2388.0 103.9924 < 2.2e-16 \*\*\*  
## Residuals 493 11321.0 23.0   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

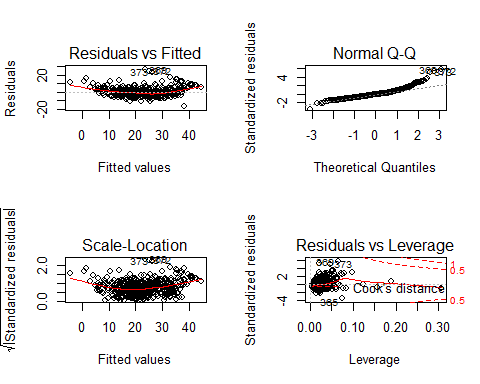
summary(lm.fit3)

##   
## Call:  
## lm(formula = medv ~ crim + zn + indus + chas + nox + rm + age +   
## dis + rad + ptratio + black + lstat, data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -16.1449 -2.9143 -0.5661 1.7438 26.3113   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 3.463e+01 5.123e+00 6.760 3.92e-11 \*\*\*  
## crim -1.067e-01 3.319e-02 -3.216 0.001384 \*\*   
## zn 3.637e-02 1.351e-02 2.692 0.007354 \*\*   
## indus -6.778e-02 5.583e-02 -1.214 0.225317   
## chas 3.029e+00 8.637e-01 3.507 0.000494 \*\*\*  
## nox -1.870e+01 3.847e+00 -4.862 1.57e-06 \*\*\*  
## rm 3.912e+00 4.209e-01 9.294 < 2e-16 \*\*\*  
## age -6.054e-04 1.333e-02 -0.045 0.963804   
## dis -1.488e+00 2.014e-01 -7.390 6.31e-13 \*\*\*  
## rad 1.346e-01 4.125e-02 3.262 0.001182 \*\*   
## ptratio -9.851e-01 1.317e-01 -7.478 3.48e-13 \*\*\*  
## black 9.546e-03 2.711e-03 3.521 0.000470 \*\*\*  
## lstat -5.222e-01 5.121e-02 -10.198 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 4.792 on 493 degrees of freedom  
## Multiple R-squared: 0.735, Adjusted R-squared: 0.7285   
## F-statistic: 113.9 on 12 and 493 DF, p-value: < 2.2e-16

vif(lm.fit3)

## crim zn indus chas nox rm age dis   
## 1.791940 2.184240 3.226015 1.058220 4.369271 1.923075 3.098044 3.954446   
## rad ptratio black lstat   
## 2.837494 1.788839 1.347564 2.940800

Plot residual ditampilkan pada Gambar 83.



Gambar 83: Analisis residual model linier 3 pada dataset Boston.

### Model Linier dengan Interaksi Antar Variabel Prediktor

Interaksi antar variabel pada model linier dapat dengan mudah dimasukkan kedalam fungsi lm(). Terdapat dua buah cara untuk melakukannya. Cara pertama dengan menggunakan tanda : pada formula (contoh: ). Tanda : menyatakan formula persamaan linier memasukkan interaksi antar variabel prediktor di dalamnya. Cara kedua adalah dengan menggunakan tanda \*. Cara ini lebih sederhana, dimana fungsi lm() akan secara otomatis menerjemahkannya sebagai serangkaian variabel tunggal dan interaksinya. Berikut adalah contoh penerapannya menggunakan kedua cara tersebut:

# cara 1  
lm.inter <- lm(medv~lstat+age+lstat:age, data=Boston)  
anova(lm.inter)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## lstat 1 23243.9 23243.9 614.8498 < 2.2e-16 \*\*\*  
## age 1 304.3 304.3 8.0481 0.004739 \*\*   
## lstat:age 1 190.4 190.4 5.0368 0.025249 \*   
## Residuals 502 18977.7 37.8   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(lm.inter)

##   
## Call:  
## lm(formula = medv ~ lstat + age + lstat:age, data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.806 -4.045 -1.333 2.085 27.552   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 36.0885359 1.4698355 24.553 < 2e-16 \*\*\*  
## lstat -1.3921168 0.1674555 -8.313 8.78e-16 \*\*\*  
## age -0.0007209 0.0198792 -0.036 0.9711   
## lstat:age 0.0041560 0.0018518 2.244 0.0252 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 6.149 on 502 degrees of freedom  
## Multiple R-squared: 0.5557, Adjusted R-squared: 0.5531   
## F-statistic: 209.3 on 3 and 502 DF, p-value: < 2.2e-16

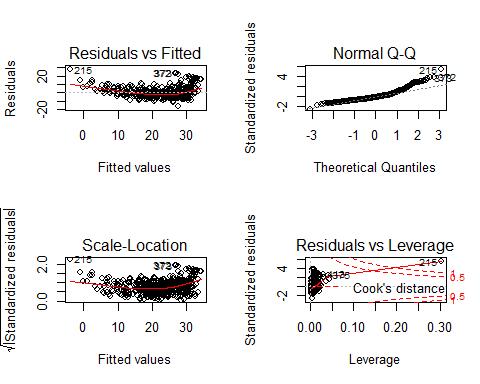
# Cara 2  
lm.inter <- lm(medv~lstat\*age, data=Boston)  
anova(lm.inter)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## lstat 1 23243.9 23243.9 614.8498 < 2.2e-16 \*\*\*  
## age 1 304.3 304.3 8.0481 0.004739 \*\*   
## lstat:age 1 190.4 190.4 5.0368 0.025249 \*   
## Residuals 502 18977.7 37.8   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(lm.inter)

##   
## Call:  
## lm(formula = medv ~ lstat \* age, data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.806 -4.045 -1.333 2.085 27.552   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 36.0885359 1.4698355 24.553 < 2e-16 \*\*\*  
## lstat -1.3921168 0.1674555 -8.313 8.78e-16 \*\*\*  
## age -0.0007209 0.0198792 -0.036 0.9711   
## lstat:age 0.0041560 0.0018518 2.244 0.0252 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 6.149 on 502 degrees of freedom  
## Multiple R-squared: 0.5557, Adjusted R-squared: 0.5531   
## F-statistic: 209.3 on 3 and 502 DF, p-value: < 2.2e-16

Plot residual ditampilkan pada Gambar 84.



Gambar 84: Analisis residual model dengan melibatkan interaksi antar variabel pada dataset Boston.

### Transformasi Non-linier Pada Prediktor

Fungsi lm() juga dapat melibatkan transformasi non-linier prediktor pada argumen formula-nya. Transformasi non-linier dilakukan dengan menambahkan fungsi identitas I(). Sebagai contoh model berikut melibatkan transformasi kuadrat pada variabel lstat:

lm.trans <- lm(medv~lstat+I(lstat^2), data=Boston)  
anova(lm.trans)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## lstat 1 23243.9 23243.9 761.81 < 2.2e-16 \*\*\*  
## I(lstat^2) 1 4125.1 4125.1 135.20 < 2.2e-16 \*\*\*  
## Residuals 503 15347.2 30.5   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(lm.trans)

##   
## Call:  
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.2834 -3.8313 -0.5295 2.3095 25.4148   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 42.862007 0.872084 49.15 <2e-16 \*\*\*  
## lstat -2.332821 0.123803 -18.84 <2e-16 \*\*\*  
## I(lstat^2) 0.043547 0.003745 11.63 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 5.524 on 503 degrees of freedom  
## Multiple R-squared: 0.6407, Adjusted R-squared: 0.6393   
## F-statistic: 448.5 on 2 and 503 DF, p-value: < 2.2e-16

Cara yang lebih sederhana untuk melibatkan tranformasi polinomial kedalam model linier adalah dengan menggunakan fungsi poly(). Berikut adalah contoh penerapannya:

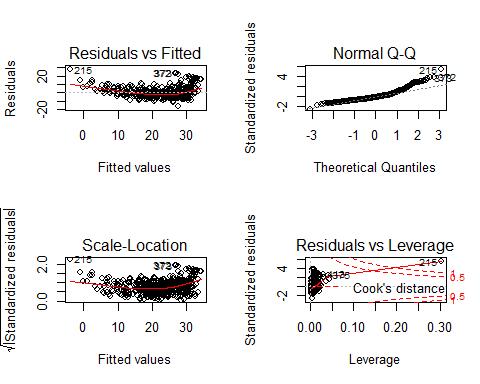
lm.trans <- lm(medv~poly(lstat,2), data=Boston)  
anova(lm.trans)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## poly(lstat, 2) 2 27369 13684.5 448.51 < 2.2e-16 \*\*\*  
## Residuals 503 15347 30.5   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(lm.trans)

##   
## Call:  
## lm(formula = medv ~ poly(lstat, 2), data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.2834 -3.8313 -0.5295 2.3095 25.4148   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 22.5328 0.2456 91.76 <2e-16 \*\*\*  
## poly(lstat, 2)1 -152.4595 5.5237 -27.60 <2e-16 \*\*\*  
## poly(lstat, 2)2 64.2272 5.5237 11.63 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 5.524 on 503 degrees of freedom  
## Multiple R-squared: 0.6407, Adjusted R-squared: 0.6393   
## F-statistic: 448.5 on 2 and 503 DF, p-value: < 2.2e-16

Plot residual ditampilkan pada Gambar 85.



Gambar 85: Analisis residual model dengan melibatkan transformasi non-linier variabel prediktor pada dataset Boston.

FUngsi trasnformasi lainnya juga dapat digunakan pada pembentukan model linier. Berikut adalah contoh penerapan transformasi logaritmik dan eksponensial pada model linier:

summary(lm(medv~log(lstat), data=Boston))

##   
## Call:  
## lm(formula = medv ~ log(lstat), data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -14.4599 -3.5006 -0.6686 2.1688 26.0129   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 52.1248 0.9652 54.00 <2e-16 \*\*\*  
## log(lstat) -12.4810 0.3946 -31.63 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 5.329 on 504 degrees of freedom  
## Multiple R-squared: 0.6649, Adjusted R-squared: 0.6643   
## F-statistic: 1000 on 1 and 504 DF, p-value: < 2.2e-16

summary(lm(medv~exp(lstat), data=Boston))

##   
## Call:  
## lm(formula = medv ~ exp(lstat), data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -17.573 -5.473 -1.373 2.427 27.427   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 2.257e+01 4.090e-01 55.187 <2e-16 \*\*\*  
## exp(lstat) -4.436e-16 2.785e-16 -1.592 0.112   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 9.183 on 504 degrees of freedom  
## Multiple R-squared: 0.005006, Adjusted R-squared: 0.003032   
## F-statistic: 2.536 on 1 and 504 DF, p-value: 0.1119

### Model Linier dengan Prediktor Kategorikal

Regresi linier dapat pula dilakukan dengan jenis variabel prediktor berupa variabel kategorikal. Untuk dapat melakukannya jenis data variabel kategorikal terlebih dahulu dirubah kedalam factor.

Pada contoh kali ini, kita akan menggunakan dataset utds dari *library* edtaR untuk memodelkan konsentrasi uranium dalam air tanah menggunakan variabel TDS pada berbagai kondisi kesadahan. Berikut adalah sintaks yang digunakan

library(edtaR)  
str(utds)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 44 obs. of 3 variables:  
## $ TDS : num 683 819 304 1151 582 ...  
## $ Uranium : num 0.931 1.938 0.292 11.904 1.567 ...  
## $ Bicarbonate: num 0 0 0 0 0 0 0 0 0 0 ...

# ubah variabe Bicarbonat menjadi factor  
utds$Bicarbonate <- factor(utds$Bicarbonate,  
 levels = c(0,1),  
 labels = c(" <= 50%",  
 " > 50%"))  
  
str(utds)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 44 obs. of 3 variables:  
## $ TDS : num 683 819 304 1151 582 ...  
## $ Uranium : num 0.931 1.938 0.292 11.904 1.567 ...  
## $ Bicarbonate: Factor w/ 2 levels " <= 50%"," > 50%": 1 1 1 1 1 1 1 1 1 1 ...

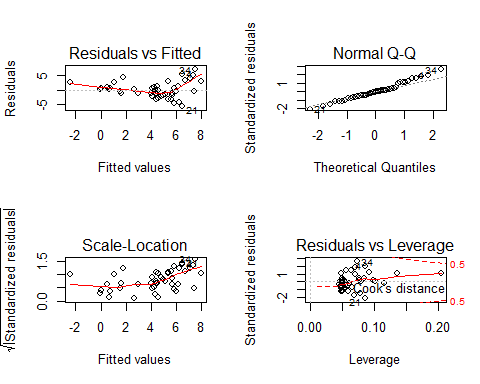
lm.fit <- lm(Uranium~., data=utds)  
anova(lm.fit)

## Analysis of Variance Table  
##   
## Response: Uranium  
## Df Sum Sq Mean Sq F value Pr(>F)   
## TDS 1 27.44 27.442 3.3106 0.07614 .   
## Bicarbonate 1 220.19 220.187 26.5637 6.825e-06 \*\*\*  
## Residuals 41 339.85 8.289   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(lm.fit)

##   
## Call:  
## lm(formula = Uranium ~ ., data = utds)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -5.7102 -1.6818 -0.1214 1.1444 7.1177   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -5.611878 1.876167 -2.991 0.00469 \*\*   
## TDS 0.010512 0.002057 5.111 7.84e-06 \*\*\*  
## Bicarbonate > 50% 6.938445 1.346227 5.154 6.83e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 2.879 on 41 degrees of freedom  
## Multiple R-squared: 0.4215, Adjusted R-squared: 0.3933   
## F-statistic: 14.94 on 2 and 41 DF, p-value: 1.34e-05

Plot residual yang ditampilkan pada Gambar 86.



Gambar 86: Analisis residual model dengan melibatkan variabel kategorikal pada dataset utds.

### Regresi linier dengan Pembobotan

Pada pembahasan sebelumnya kita telah menyaksikan bahwa sebagian model yang telah terbentuk tidak memenuhi asumsi varians yang konstan. Untuk mengatasi hal tersebut, kita dapat membentuk regresi dengan memberikan bobot sebesar invers variansnya. Untuk melakukannya diperlukan beberapa tahapan, antara lain:

1. Membentuk model linier dari variabel dataset: fit <- lm(y~(variabel prediktor)).
2. Menghitung error absolut (abse <- abs(resid(fit))) dan nilai \*fitted value dari model (yhat <- fitted(fit).
3. Membentuk kembali model menggunakan data residual absolut (efit <- lm(abse~poly(yhat,2))) dan menghitung *residual fitted value* (shat <- fitted(efit)).
4. Gunakan nilai bobot w <- 1/shat^2 untuk membentuk model regresi dengan pembobotan (fitw <-lm(y~(variabel prediktor), weights=w)).

Kita akan membentuk kembali model menggunakan dataset Boston dengan menggunakan seluruh variabel, namun pada model kali ini kita akan memberikan bobot pada model yang terbentuk. Berikut adalah sintaks yang digunakan:

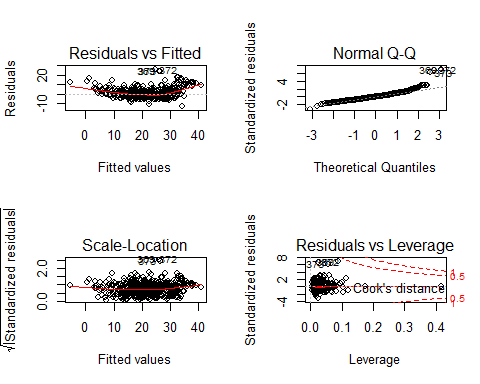
# langkah 1  
fit <- lm(medv~., data=Boston)  
  
# langkah 2  
abse <- abs(resid(fit))  
yhat <- fitted(fit)  
  
# langkah 3  
efit <- lm(abse~poly(yhat,2))  
shat <- fitted(efit)  
  
# langkah 4  
fitw <- lm(medv~., data = Boston, weights = 1/(shat^2))  
anova(fitw)

## Analysis of Variance Table  
##   
## Response: medv  
## Df Sum Sq Mean Sq F value Pr(>F)   
## crim 1 367.13 367.13 188.5179 < 2.2e-16 \*\*\*  
## zn 1 148.25 148.25 76.1235 < 2.2e-16 \*\*\*  
## indus 1 124.51 124.51 63.9328 9.264e-15 \*\*\*  
## chas 1 41.31 41.31 21.2115 5.244e-06 \*\*\*  
## nox 1 38.33 38.33 19.6841 1.128e-05 \*\*\*  
## rm 1 363.47 363.47 186.6396 < 2.2e-16 \*\*\*  
## age 1 30.37 30.37 15.5945 8.994e-05 \*\*\*  
## dis 1 105.70 105.70 54.2755 7.407e-13 \*\*\*  
## rad 1 0.00 0.00 0.0003 0.9854   
## tax 1 30.01 30.01 15.4080 9.897e-05 \*\*\*  
## ptratio 1 73.76 73.76 37.8761 1.565e-09 \*\*\*  
## black 1 64.38 64.38 33.0607 1.570e-08 \*\*\*  
## lstat 1 334.30 334.30 171.6631 < 2.2e-16 \*\*\*  
## Residuals 492 958.14 1.95   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(fitw)

##   
## Call:  
## lm(formula = medv ~ ., data = Boston, weights = 1/(shat^2))  
##   
## Weighted Residuals:  
## Min 1Q Median 3Q Max   
## -4.2522 -0.7946 -0.0367 0.6976 9.2986   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 39.158445 4.474180 8.752 < 2e-16 \*\*\*  
## crim -0.137382 0.032536 -4.222 2.88e-05 \*\*\*  
## zn 0.039751 0.013668 2.908 0.003797 \*\*   
## indus 0.031072 0.052112 0.596 0.551285   
## chas 2.886787 0.816335 3.536 0.000444 \*\*\*  
## nox -15.844212 3.221769 -4.918 1.19e-06 \*\*\*  
## rm 2.322467 0.414237 5.607 3.44e-08 \*\*\*  
## age 0.005865 0.011469 0.511 0.609298   
## dis -1.137778 0.177740 -6.401 3.60e-10 \*\*\*  
## rad 0.302548 0.058053 5.212 2.76e-07 \*\*\*  
## tax -0.011376 0.003277 -3.471 0.000564 \*\*\*  
## ptratio -0.721781 0.115976 -6.224 1.04e-09 \*\*\*  
## black 0.008784 0.002342 3.751 0.000197 \*\*\*  
## lstat -0.621847 0.047462 -13.102 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.396 on 492 degrees of freedom  
## Multiple R-squared: 0.6424, Adjusted R-squared: 0.633   
## F-statistic: 68 on 13 and 492 DF, p-value: < 2.2e-16

Plot residual ditampilkan pada Gambar 87.



Gambar 87: Analisis residual model regresi linier dengan pembobotan pada dataset Boston.

## Regresi Logistik

Pada Chapter 12.2, kita telah membahas cara untuk membangun model dengan output berupa variabel dengan nilai numerik. Pada Chapter 12.2, kita akan belajar cara membentuk model regresi dengan 2 respons (0 dan 1). Pada regresi ini pembentukan model didasarkan oleh kurva logistik, dimana melalui kurva tersebut nilai yang dihasilkan akan memiliki rentang dari 0 sampai 1. Karena model yang dibuat bertujuan untuk memprediksi dua buah kemungkinan (0 atau 1), maka diperlukan suatu nilai ambang (y<0,5 = 0 dan y >= 0,5 = 1).

Fungsi glm() dapat digunakan untuk membentuk model regresi logistik. Format umum fungsi tersebut adalah sebagai berikut:

glm(formula, family = gaussian, data, weights, subset,  
 )

**Catatan:**

* formula : formula model yang hendak dibentuk.
* family : distribusi yang digunakan. Untuk regresi logistik digunakan argumen family=binomial
* data: data yang digunakan untuk membentuk model.
* subset : subset data yang akan digunakan dalam pembentukan model.
* weight : nilai pembobotan dalam pembentukan model.

Pada contoh berikut, kita akan membuat model untuk memprediksi *contamination rating* menggunakan dataset contamination dari *library* edtaR. Berikut adalah sintaks yang digunakan:

log.model <- glm(CR ~ ., family=binomial, data=contamination)  
anova(log.model)

## Analysis of Deviance Table  
##   
## Model: binomial, link: logit  
##   
## Response: CR  
##   
## Terms added sequentially (first to last)  
##   
##   
## Df Deviance Resid. Df Resid. Dev  
## NULL 123 91.474  
## UZT 1 12.3823 122 79.092  
## AY 1 4.3228 121 74.769  
## GWQ 1 0.0147 120 74.755  
## HWM 1 25.0503 119 49.704

summary(log.model)

##   
## Call:  
## glm(formula = CR ~ ., family = binomial, data = contamination)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.09450 -0.33494 -0.08567 -0.04624 2.13239   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) -13.20539 3.55814 -3.711 0.000206 \*\*\*  
## UZT 0.51527 0.15094 3.414 0.000641 \*\*\*  
## AY 0.42909 0.27487 1.561 0.118506   
## GWQ 0.03035 0.32461 0.093 0.925515   
## HWM 1.08951 0.29863 3.648 0.000264 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 91.474 on 123 degrees of freedom  
## Residual deviance: 49.704 on 119 degrees of freedom  
## AIC: 59.704  
##   
## Number of Fisher Scoring iterations: 7

## Referensi

1. Akritas, M. 2016. **PROBABILITY & STATISTICS WITH R FOR ENGINEERS AND SCIENTISTS**. Pearson.
2. Bloomfield, V.A. 2014. **Using R for Numerical Analysis in Science and Engineering**. CRC Press.
3. James, G., Witten, D., Hastie, T., Tibshirani, R. 2013. **An Introduction to Statistical Learning**. Springer.
4. Kerns, G.J., 2018. **Introduction to Probability and Statistics Using R**. Course notes for University of Auckland Paper STATS 330. <http://ipsur.r-forge.r-project.org/book/download/IPSUR.pdf>.
5. Lee, A., Ihaka, R., Triggs, C. 2012. **ADVANCED STATISTICAL MODELLING**.
6. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
7. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.
8. STHDA. <(<http://www.sthda.com/english/>>