UNIVERSITAS SULTAN AGENG TIRTAYASA

UNTIRTA Jawara
Jujur Adil Wibawa Amanah Religius Akuntabel

Komputasi Statistika

#9 Meeting

Numerical Accuracy

Ferdian Bangkit Wijaya, S.Stat., M.Si
NIP. 19900520202406100 1

Machine Representation

Pengertian dan ilustrasi

- Definisi: "Representasi Mesin" adalah cara komputer menerjemahkan dan menyimpan angka yang kita pahami (sistem desimal/basis 10) ke dalam bahasa internalnya (sistem biner/basis 2, yaitu 0 dan 1) di dalam memori atau prosesor.
- Contoh Sederhana (Analogi): Bayangkan kita diminta menulis angka "satu per tiga" (1/3) sebagai desimal. Kita akan menulis 0.33333... Tentu tidak bisa ditulis selamanya, bukan? Kita pasti akan berhenti dan membulatkan di satu titik, misal 0.333.
- Komputer mengalami masalah yang sama persis, tetapi dengan angka yang berbeda. Bagi komputer, angka "nol koma satu" (0.1) adalah angka desimal yang berulang dan tak terbatas dalam bahasa biner. Ia juga harus membulatkan 0.1 saat menyimpannya.

Machine Representation

Pengertian dan ilustrasi

- Komputer tidak menyimpan angka desimal (seperti 0.1) secara sempurna. Komputer menggunakan sistem biner (basis 2) dan representasi floating-point (standar IEEE 754).
- Akibatnya, banyak angka desimal yang sederhana bagi kita, menjadi angka biner yang berulang dan tak terbatas, sehingga harus dibulatkan. Ini disebut floating-point error atau representation error.
- 0.1 (Desimal) = 0.0001100110011... (Biner berulang)
- Karena 0.1 yang disimpan komputer adalah pembulatan, dan 0.2 yang disimpan juga pembulatan, penjumlahannya tidak akan sama persis dengan 0.3 yang juga disimpan sebagai pembulatan.

Machine Representation

Pengertian dan ilustrasi

Contoh: Ubah 13 (desimal) ke biner
13 / 2 = 6 → Sisa 1
6 / 2 = 3 → Sisa 0
3 / 2 = 1 → Sisa 1
1 / 2 = 0 → Sisa 1 (Berhenti karena hasil sudah 0)
Baca sisanya dari bawah ke atas: 1101. Jadi, 13 (desimal) = 1101 (biner).
(Verifikasi: $(1*2^3) + (1*2^2) + (0*2^1) + (1*2^0) = 8 + 4 + 0 + 1 = 13$)

Machine Representation

Pengertian dan ilustrasi

Ubah 0.625 (desimal) ke biner

- 0.625 * 2 = 1.25 → Catat 1, ambil sisa pecahan 0.25
- 0.25 * 2 = 0.50 → Catat 0, ambil sisa pecahan 0.50
- 0.50 * 2 = 1.00 → Catat 1, ambil sisa pecahan 0.00 (Berhenti)

Baca catatannya dari atas ke bawah: 101. Jadi, 0.625 (desimal) = 0.101 (biner).
(Verifikasi: (1 * 1/2) + (0 * 1/4) + (1 * 1/8) = 0.5 + 0 + 0.125 = 0.625)

Machine Representation

Pengertian dan ilustrasi

Ubah 0.1 (desimal) ke Biner

Sekarang kita terapkan Metode Kali Dua pada 0.1

- 0.1 * 2 = 0.2 → Catat 0, ambil sisa 0.2
- 0.2 * 2 = 0.4 → Catat 0, ambil sisa 0.4
- 0.4 * 2 = 0.8 → Catat 0, ambil sisa 0.8
- 0.8 * 2 = 1.6 → Catat 1, ambil sisa 0.6
- 0.6 * 2 = 1.2 → Catat 1, ambil sisa 0.2

Kita kembali mendapatkan sisa pecahan 0.2, yang sama persis dengan di langkah pertama. Jika kita lanjutkan, prosesnya akan berulang selamanya:

- 0.2 * 2 = 0.4 → Catat 0...
- 0.4 * 2 = 0.8 → Catat 0...
- 0.8 * 2 = 1.6 → Catat 1...
- 0.6 * 2 = 1.2 → Catat 1...
- (dan seterusnya...)

0.0001100110011001100110011... (biner).

Komputer tidak bisa menyimpan angka yang berulang tak terbatas. Komputer harus memotong (membulatkan) biner tersebut pada batas presisi tertentu (misal, 52 bit).

Machine Representation

Pengertian dan ilustrasi

Ubah 13.625 (desimal) ke dalam angka Biner !!!.

Machine Representation

Pengertian dan ilustrasi

```
# Penjumlahan sederhana
a <- 0.1
b <- 0.2
print(a + b)
[1] 0.3

# Mari kita uji kesamaannya dengan 0.3
print(a + b == 0.3)
 [1] FALSE
```

Mengapa FALSE?

Mari kita lihat representasi mesin yang sebenarnya dengan presisi tinggi:

Machine Representation

Pengertian dan ilustrasi

```
# Tampilkan dengan 22 digit presisi
print(a + b, digits = 22)
[1] 0.3000000000000000444089

print(0.3, digits = 22)
[1] 0.2999999999999999888978
```

```
# Cara yang Benar
all.equal(a + b, 0.3)
[1] TRUE
```

Jangan pernah menggunakan operator == untuk membandingkan dua angka floating-point (desimal). Selalu gunakan fungsi all.equal() yang memberikan toleransi, atau periksa apakah selisih absolutnya lebih kecil dari angka yang sangat kecil (epsilon).

R (1.49×10^{-8})

Significant Digit

Pengertian dan ilustrasi

- Definisi: Angka signifikan adalah digit-digit dalam sebuah angka yang dianggap "bermakna" atau "tepercaya" dalam menunjukkan presisi suatu pengukuran.
- Contoh Sederhana: Bayangkan menimbang berat badan.
- Timbangan A (murah): Menampilkan 70 kg. Ini punya 1 angka signifikan (hanya angka 7, angka 0 di sini tidak pasti).
- Timbangan B (mahal): Menampilkan 70.1kg. Ini punya 3 angka signifikan (7, 0, dan 1).
- Angka 70.1 lebih presisi daripada 70. Ia membawa lebih banyak "informasi yang bermakna".

Significant Digit

Pengertian dan ilustrasi

- Dalam komputasi, masalah muncul saat kita melakukan operasi matematika (terutama pengurangan) pada dua angka yang nilainya sangat berdekatan. Fenomena ini disebut Catastrophic Cancellation (Pembatalan Katastropik).
- Misalkan kita memiliki dua angka presisi tinggi: 123.456 dan 123.450. Keduanya memiliki 6 angka signifikan.
- Jika dikurangkan:
- 123.456 - 123.450 = 0.006
- Hasilnya, 0.006. hanya memiliki 1 angka signifikan (yaitu 6). Lima digit presisi (1, 2, 3, 4, 5) saling "membatalkan" dan hilang. Telah terjadi kehilangan 5 digit informasi!

Significant Digit

Pengertian dan ilustrasi

```
# Angka besar
x <- 1e16  # Ini adalah 1 dengan 16 angka nol

# Tambahkan 1
y <- x + 1

# Sekarang kita kurangi lagi dengan x
# Secara matematis, (1e16 + 1) - 1e16 = 1
print(y - x)
[1] 0
```

- Hasilnya adalah 0, bukan 1.
- Mengapa? Karena dalam presisi standar R, angka 1e16 dan 1e16 + 1 dianggap identik.
- Angka 1 "hilang" karena terlalu kecil dibandingkan 1e16 (sudah di luar jangkauan angka signifikan yang bisa disimpan).

Waktu Eksekusi (Time)

Pengertian dan ilustrasi

- Definisi: Proses mengukur berapa lama waktu (durasi) yang dibutuhkan oleh komputer untuk menjalankan sebuah perintah atau skrip kode.
- Contoh Sederhana: Ini persis seperti menggunakan stopwatch saat Anda memasak. Anda ingin tahu, "Berapa lama waktu untuk merebus air?" Anda tekan "mulai" saat menyalakan kompor dan "berhenti" saat air mendidih.
- Dalam R, kita "membungkus" kode kita dengan stopwatch yang sama.

Waktu Eksekusi (Time)

Pengertian dan ilustrasi

- Dalam komputasi, terutama dengan data besar atau simulasi, kecepatan sangat penting. Kita perlu tahu berapa lama kode kita berjalan untuk mengidentifikasi bagian mana yang lambat (bottleneck).

- Fungsi system.time() di R mengembalikan 3 nilai:

1. user: Waktu CPU yang dihabiskan oleh kode Anda.
2. system: Waktu CPU yang dihabiskan oleh OS (misal, membaca/menulis file).
3. elapsed: Waktu total (dunia nyata) dari awal sampai akhir. Ini yang biasanya kita pedulikan.

Waktu Eksekusi (Time)

Pengertian dan ilustrasi

```
# Kita ukur waktu untuk membuat vektor 10 juta angka acak
data_size <- 1e7 # 10 juta

time_taken <- system.time({
  # Kode yang ingin diukur ada di dalam kurung kurawal {}
  random_numbers <- rnorm(data_size)
  mean(random_numbers)
})

print(time_taken)
```

Loop vs Vectorize

Pengertian dan ilustrasi

- Definisi: Ini adalah dua strategi berbeda untuk melakukan tugas berulang.
- Contoh Sederhana: Bayangkan Anda ingin memberi tahu 50 mahasiswa untuk "mengerjakan PR".
- Loop (satu per satu): Anda panggil Budi, "Budi, kerjakan PR." Lalu Anda panggil Ani, "Ani, kerjakan PR." Lalu Cici... (Anda melakukan 50 percakapan terpisah). Ini lambat.
- Vektorisasi (sekaligus): Anda masuk ke kelas dan berteriak, "Semuanya, kerjakan PR!" (Anda memberikan satu perintah yang berlaku untuk semua 50 mahasiswa sekaligus). Ini cepat.

Loop vs Vectorize

Pengertian dan ilustrasi

- Ini adalah konsep paling penting untuk efisiensi di R.
- Loop: Seperti for loop, memproses data satu elemen pada satu waktu. Di R, ini sangat lambat karena R adalah bahasa interpreted (setiap perintah "diterjemahkan" satu per satu saat berjalan).
- Vektorisasi: Melakukan operasi pada seluruh vektor (atau matriks) sekaligus. Di balik layar, R menggunakan kode C atau Fortran yang sudah terkompilasi dan sangat cepat.

Loop vs Vectorize

Pengertian dan ilustrasi

```
n <- 1e6 # 1 juta elemen
x <- 1:n
y <- 1:n

# --- Metode 1: Menggunakan 'for' loop (LAMBAT) ---
time_loop <- system.time({
  z_loop <- numeric(n) # Buat wadah kosong dulu
  for (i in 1:n) {
    z_loop[i] <- x[i] + y[i]
  }
})

# --- Metode 2: Menggunakan Vektorisasi (CEPAT) ---
time_vectorized <- system.time({
  z_vectorized <- x + y
})

# --- Bandingkan Hasil ---
cat("Waktu menggunakan loop:", time_loop['elapsed'], "detik\n")
cat("Waktu menggunakan vektorisasi:", time_vectorized['elapsed'], "detik\n")
```

Parallel Processing

Pengertian dan ilustrasi

- Definisi: Menggunakan beberapa "otak" (CPU core) komputer secara bersamaan untuk menyelesaikan satu tugas besar.
- Contoh Sederhana: Bayangkan Anda harus mencuci 4 piring kotor.
- Serial (1 inti): Ibu mencuci piring 1, lalu piring 2, lalu piring 3, lalu piring 4.
- Paralel (4 inti): Ibu, Kakak, Adik, dan Ayah berdiri di wastafel dan masing-masing mencuci 1 piring secara bersamaan. Pekerjaan selesai 4 kali lebih cepat.

Parallel Processing

Pengertian dan ilustrasi

- Komputer modern memiliki banyak CPU core (misal, 4, 8, atau 16). Secara default, R hanya menggunakan satu inti. Jika Anda punya 8 inti, 7 inti lainnya menganggur. Pemrosesan paralel memungkinkan kita memecah tugas besar menjadi bagian-bagian kecil dan menjalankannya di semua inti secara bersamaan.
- Simulasi bootstrap, validasi silang (cross-validation), atau grid search sangat cocok untuk pemrosesan ini.

Parallel Processing

Pengertian dan ilustrasi

```
library(parallel)

# 1. Definisikan fungsi "berat" kita
# (Fungsi ini butuh 0.5 detik untuk selesai)
heavy_function <- function(i) {
  Sys.sleep(0.5) # Pura-pura melakukan perhitungan berat
  return(i * 2)
}
```

```
# --------------------------------
# --- Metode 1: Serial (lapply) ---
# --------------------------------
# Ini akan berjalan 10 x 0.5 detik = 5 detik
time_serial <- system.time({
  results_serial <- lapply(1:10, heavy_function)
})

cat("Waktu Serial (lapply):", time_serial['elapsed'], "detik\n")
```

Parallel Processing

Pengertian dan ilustrasi

```
# ----------------------------------
# --- Metode 2: Paralel (parLapply) ---
# ----------------------------------
# Deteksi berapa inti CPU yang kita punya (misal, kurangi 1 agar OS tetap jalan)
num_cores <- detectCores() - 1

# 1. Buat "Cluster" (kumpulan pekerja R)
cl <- makeCluster(num_cores)

# 2. Jalankan fungsi secara paralel
time_parallel <- system.time({
  results_parallel <- parLapply(cl, 1:10, heavy_function)
})

# 3. Selalu hentikan cluster setelah selesai!
stopCluster(cl)

cat("Waktu Paralel (parLapply):", time_parallel['elapsed'], "detik\n")
```

Memory

Pengertian dan ilustrasi

- Definisi: Mengacu pada bagaimana program (seperti R) menggunakan RAM (Random Access Memory) komputer untuk menyimpan objek, data, dan variabel yang sedang digunakan.
- Contoh Sederhana: Anggap RAM adalah meja kerja.
- Membaca data 1GB (data <- read.csv(...)) sama seperti mengambil buku setebal 1GB dan meletakkannya di atas meja.
- Jika meja (RAM) hanya 8GB, mustahil meletakkan buku 10GB di atasnya.
- Jika buku itu disalin (data2 <- data), sekarang ada dua buku di atas meja, menghabiskan 2GB. Mengelola memori adalah tentang membersihkan buku (data) yang tidak lagi diperlukan dari meja.

Memory

Pengertian dan ilustrasi

- R adalah bahasa yang "rakus" memori karena R menyimpan semua objek (data frame, vektor, model) di dalam RAM. Jika data lebih besar dari RAM yang tersedia, R akan crash.
- Dua hal penting di R:

1. Ukuran Objek: Kita harus sadar seberapa besar data kita.
2. Copy-on-Modify: Ini adalah perilaku khusus R. Jika kita membuat y <- x, R tidak langsung menyalin. y hanya "menunjuk" ke x. Tapi begitu 1 elemen saja di y diubah (misal y[1] <- 100), R langsung membuat salinan penuh dari x untuk y. Ini bisa tiba-tiba menghabiskan memori dua kali lipat.
3. Kita bisa menggunakan object.size() untuk melihat ukuran satu objek dan pryr::mem_used() untuk melihat total memori yang dipakai R.
4. rm() lalu gc() untuk membersihkan memory.

Memory

Pengertian dan ilustrasi

```
# Install dulu jika belum punya
# install.packages("pryr")
library(pryr)

# 1. Melihat ukuran objek
my_vector <- 1:1e7 # 10 juta angka integer
print(object.size(my_vector))
# 40,000,048 bytes (sekitar 40 MB)

# 2. Melihat perilaku "Copy-on-Modify"
cat("Memori terpakai A:", mem_used(), "\n")

# Buat objek besar
x <- 1:1e8 # 100 juta angka (sekitar 400 MB)
cat("Memori terpakai B (setelah 'x'):",
mem_used(), "\n")

# Buat 'y' menunjuk ke 'x'. Memori TIDAK
bertambah.
y <- x
cat("Memori terpakai C (setelah 'y <- x'):",
mem_used(), "\n")

# Ubah 1 elemen saja di 'y'
y[1] <- 100
```

Memory

Pengertian dan ilustrasi

```
# R 'dipaksa' membuat salinan penuh. Memori bertambah 400MB lagi.
cat("Memori terpakai D (setelah 'y[1] <- 100'):", mem_used(), "\n")

# 3. Membersihkan memori
# Hapus objek yang tidak perlu
rm(x, y)

# Jalankan "Garbage Collector" (gc) secara manual
gc()
cat("Memori terpakai E (setelah rm dan gc):", mem_used(), "\n")
```

UNIVERSITAS
SULTAN AGENG
TIRTAYASA

UNTIRTA
Jawara
Jujur Adil Wibawa Amanah Religius Akuntabel

SEE YOU NEXT WEEK !

Ferdian Bangkit Wijaya, S.Stat., M.Si
NIP. 19900520202406100 1
ferdian.bangkit@untirta.ac.id