Modul Praktikum 7 Komputasi Statistik

Numerical Integration 1 – Metode Riemann, Trapesium, dan Simpson 1/3

Tim Dosen Komputasi Statistik – Sains Data - ITERA

1. Pendahuluan

Dalam banyak bidang sains dan statistika, kita sering kali ingin mengetahui **luas di bawah kurva** dari suatu fungsi. Luas tersebut secara matematis dinyatakan sebagai **integral tentu**:

$$I = \int_a^b f(x)\,dx.$$

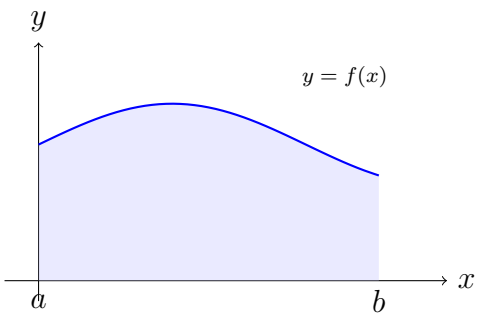
Jika fungsi $f(x)$ tidak memiliki bentuk integral yang bisa diselesaikan secara analitik, kita perlu menghitungnya dengan pendekatan numerik, yang dikenal sebagai **integrasi numerik (numerical integration)**.

Analogi Geometris

Bayangkan kamu ingin menghitung luas daerah yang dibatasi oleh kurva $y = f(x)$, sumbu-x, dan garis vertikal $x = a$ sampai $x = b$. Jika bentuk daerahnya persegi atau segitiga, kita bisa langsung pakai rumus luas. Namun bentuk kurva umumnya *melengkung* dan tidak beraturan.

Ide pentingnya adalah:

- Bagi daerah tersebut menjadi potongan-potongan kecil.
- Setiap potongan didekati dengan **bentuk sederhana** (persegi panjang, trapesium, atau parabola).
- Luas total $\approx$ jumlah luas semua potongan kecil.



Dengan membagi interval $[a, b]$ menjadi n bagian kecil, luas ini dapat dihampiri secara numerik menggunakan metode Riemann, Trapesium, dan Simpson.

2. Ide Dasar Aproksimasi Integral

Secara umum, langkah-langkah aproksimasi integral adalah:

1. Bagi interval $[a, b]$ menjadi n sub-selang dengan panjang sama:

$$h = \frac{b-a}{n}.$$

2. Bentuk titik-titik pembagi:

$$x_0 = a, \quad x_1 = a + h, \quad x_2 = a + 2h, \ \ldots, \ x_n = b.$$

3. Pada setiap sub-selang $[x_{i-1}, x_i]$, pilih titik x_i^* di dalamnya dan ambil nilai fungsi $f(x_i^*)$ sebagai **tinggi**.

4. Luas di setiap panel $\approx$ (tinggi) $\times$ (lebar). Lebar panel $= h$.

5. Jumlahkan semua luas panel untuk menghampiri nilai integral.

Ketika n sangat besar (panel sangat tipis), jumlah ini mendekati nilai integral yang sebenarnya.

3. Metode Riemann

3.1. Asal-usul Rumus Riemann dari Luas Persegi Panjang

Kita mulai dari konsep paling dasar: **luas persegi panjang**.

Untuk satu persegi panjang:

$$\text{Luas} = \text{lebar} \times \text{tinggi}.$$

Pada integrasi numerik, kita mendekati kurva dengan banyak persegi panjang:

- Lebar setiap persegi panjang $= h = \dfrac{b-a}{n}$.
- Tinggi setiap persegi panjang = nilai fungsi pada titik tertentu di sub-selang, misalnya $f(x_i^*)$.

Sub-selang ke-i adalah:

$$[x_{i-1}, x_i] = [a + (i-1)h, \ a + ih].$$

Jika kita pilih suatu titik di dalam sub-selang itu, x_i^*, maka luas panel ke-i *diaproksimasi* sebagai:

$$A_i \approx f(x_i^*) \times h.$$

Jumlahkan untuk semua panel:

$$I \approx \sum_{i=1}^{n} f(x_i^*)\, h.$$

Ketika $n \to \infty$ dan $h \to 0$, definisi integral tentu secara formal adalah:

$$\int_a^b f(x)\, dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i^*)\, h,$$

yang disebut **Riemann sum**.

Dalam praktik komputasi, kita tidak mengambil limit, tetapi memilih n yang cukup besar sehingga:

$$\int_a^b f(x)\, dx \approx \sum_{i=1}^{n} f(x_i^*)\, h.$$

Bedanya Riemann kiri, kanan, dan tengah hanyalah **pilihan titik** x_i^*:

$$\begin{aligned}
\text{Riemann kiri: } & x_i^* = x_{i-1} = a + (i-1)h, \\
\text{Riemann kanan: } & x_i^* = x_i = a + ih, \\
\text{Riemann tengah: } & x_i^* = \frac{x_{i-1} + x_i}{2} = a + \left(i - \frac{1}{2}\right) h.
\end{aligned}$$

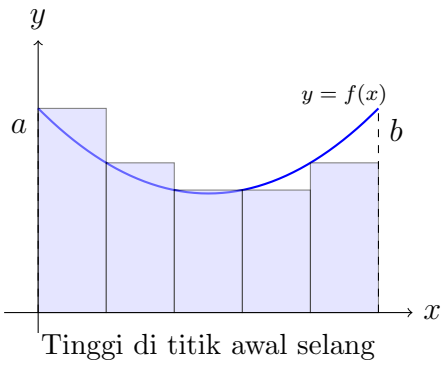
Inilah yang nanti menjadi dasar **algoritma dan kode R**: kita cukup menghitung jumlah

```
sum <- sum + f(x_i)
```

di dalam sebuah `for-loop`, lalu mengalikan totalnya dengan h.

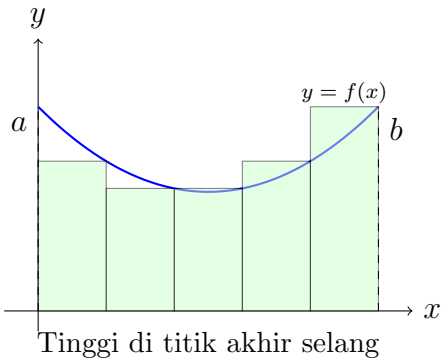
Ilustrasi Tiga Jenis Metode Riemann

Riemann Kiri



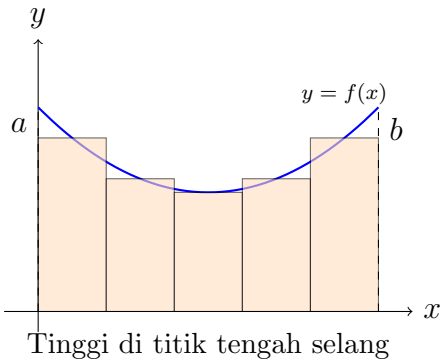
Tinggi di titik awal selang

Riemann Kanan



Tinggi di titik akhir selang

Riemann Tengah



Tinggi di titik tengah selang

3.2. Algoritma Riemann Fleksibel

Dari rumus:

$$I \approx \sum_{i=1}^{n} f(x_i^*)\, h,$$

dengan x_i^* dipilih sesuai tipe (kiri, kanan, tengah), kita dapat algoritma:

1. Masukkan fungsi $f(x)$, batas a dan b.
2. Pilih jumlah panel n dan tipe Riemann (`"left"`, `"right"`, `"mid"`).

3. Hitung $h = (b - a)/n$.

4. Inisialisasi `sum = 0`.

5. Untuk $i = 1$ sampai n:

   (a) Hitung x_i^* sesuai tipe:

$$x_i^* = \begin{cases} a + (i-1)h & \text{(kiri)} \\ a + ih & \text{(kanan)} \\ a + (i - \frac{1}{2})h & \text{(tengah)} \end{cases}$$

   (b) Tambahkan `sum = sum + f(`x_i^*`)`.

6. Hasil aproksimasi integral: `sum * h`.

3.3. Kode R: Riemann Fleksibel

```r
# Fungsi Riemann Fleksibel
riemann <- function(f, a, b, n = 10, type = "left") {
  h <- (b - a) / n        # panjang tiap panel
  sum <- 0                # inisialisasi penjumlahan

  for (i in 1:n) {
    # Tentukan titik x_i sesuai tipe
    if (type == "left") {
      x_i <- a + (i - 1) * h
    } else if (type == "right") {
      x_i <- a + i * h
    } else if (type == "mid") {
      x_i <- a + (i - 0.5) * h
    } else {
      stop("Tipe harus salah satu dari: 'left', 'right', atau '
         mid'.")
    }

    sum <- sum + f(x_i)    # akumulasi nilai fungsi
  }

  return(sum * h)          # total luas aproksimasi
}

```

```r
# Fungsi uji (sedang/medium)
f <- function(x) exp(-x^2) + 2*x

# Batas integrasi dan jumlah panel
a <- 0
b <- 2
n <- 20

# Hitung integral dengan tiga tipe
riemann_left  <- riemann(f, a, b, n, type = "left")
riemann_right <- riemann(f, a, b, n, type = "right")
riemann_mid   <- riemann(f, a, b, n, type = "mid")

# Bandingkan dengan hasil eksak
result_exact <- integrate(f, lower = a, upper = b)
cat("Riemann Kiri   :", riemann_left,  "\n")
cat("Riemann Kanan  :", riemann_right, "\n")
cat("Riemann Tengah :", riemann_mid,   "\n")
cat("Nilai Eksak    :", result_exact$value, "\n")
```

3.4. Penjelasan Kode

- `riemann()` mengimplementasikan langsung rumus

$$\sum_{i=1}^{n} f(x_i^*)\, h.$$

- `h` adalah lebar panel: $(b-a)/n$.
- `sum` adalah variabel akumulator untuk $\sum f(x_i^*)$.
- Blok `if` memilih rumus x_i^* sesuai tipe:
  - `"left"` $\rightarrow x_i^* = a + (i-1)h$.
  - `"right"` $\rightarrow x_i^* = a + ih$.
  - `"mid"` $\rightarrow x_i^* = a + (i-0.5)h$.
- Setelah loop selesai, `return(sum * h)` mengalikan jumlah tinggi dengan lebar untuk mendapatkan luas total.
- Fungsi `integrate()` dipakai sebagai pembanding nilai integral eksak secara numerik (bawaan R).

3.5. Hasil Contoh

```
Riemann Kiri : 4.6307
Riemann Kanan: 4.8735
Riemann Tengah: 4.7520
Eksak : 4.7561
```

Hasil menunjukkan bahwa metode Riemann tengah memberikan aproksimasi paling mendekati nilai eksak.

4. Metode Newton–Cotes

Setelah memahami metode Riemann yang mendekati fungsi dengan persegi panjang, kita dapat meningkatkan akurasi dengan cara mengganti pendekatan bentuk panelnya. Metode Newton–Cotes memperkirakan fungsi $f(x)$ dalam tiap sub-selang dengan polinomial sederhana, lalu mengintegrasikannya secara eksak.

4.1. Ide Dasar

Pada setiap selang kecil $[x_i, x_{i+1}]$, kita hampiri fungsi $f(x)$ dengan polinomial Lagrange yang melewati titik-titik fungsi pada batas-batas selang tersebut.

Misalnya, jika kita hanya pakai dua titik (x_i, f_i) dan (x_{i+1}, f_{i+1}), maka interpolasi linear di antara keduanya adalah:

$$P_1(x) = f_i + \frac{f_{i+1} - f_i}{h}(x - x_i),$$

dengan $h = x_{i+1} - x_i$.

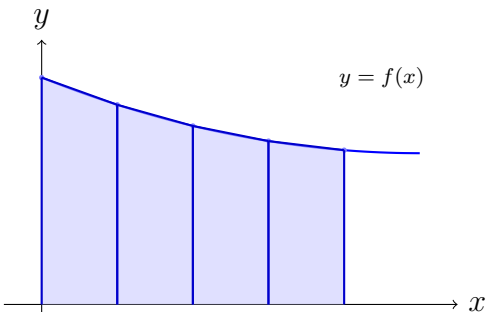
Kemudian kita integralkan polinomial ini pada selang $[x_i, x_{i+1}]$:

$$\int_{x_i}^{x_{i+1}} P_1(x)\,dx = \frac{h}{2}\big(f_i + f_{i+1}\big).$$

Hasil inilah yang dikenal sebagai **aturan Trapesium sederhana**. Dengan menjumlahkan hasil untuk semua sub-selang, kita memperoleh aturan **Trapesium Majemuk (Composite Trapezoid Rule)**:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\Big[f(a) + 2\sum_{i=1}^{n-1} f(a + ih) + f(b)\Big].$$

4.2. Ilustrasi Aturan Trapesium



Setiap panel berbentuk trapesium

Perhatikan bahwa garis atas tiap panel mengikuti garis lurus yang menghubungkan dua titik fungsi — inilah sebabnya disebut aturan **Trapesium**.

4.3. Algoritma Aturan Trapesium Majemuk

1. Bagi interval $[a, b]$ menjadi n panel dengan lebar $h = (b - a)/n$.
2. Hitung nilai fungsi pada setiap titik partisi: $f(a), f(a + h), f(a + 2h), \ldots, f(b)$.
3. Hitung jumlah:
$$S = f(a) + f(b) + 2\sum_{i=1}^{n-1} f(a + ih).$$
4. Nilai aproksimasi integral:
$$I_T = \frac{h}{2} S.$$

4.4. Kode R: Trapesium Majemuk

```r
# ================================================================

trapezoid <- function(f, a, b, n = 10, show_steps = FALSE) {
  # --- Validasi input ---
  if (n < 1) stop("n harus lebih besar dari 0.")
  if (!is.function(f)) stop("Argumen f harus berupa fungsi.")

  # --- Panjang panel ---
  h <- (b - a) / n

  # --- Inisialisasi hasil penjumlahan ---
  S <- f(a) + f(b)

  # --- Perulangan untuk titik tengah ---
```

```r
  for (i in 1:(n - 1)) {
    x_i <- a + i * h
    S <- S + 2 * f(x_i)
    if (show_steps) cat(sprintf("i=%2d | x_i = %.4f | f(x_i)=%.4f
       \n", i, x_i, f(x_i)))
  }

  # --- Hasil akhir ---
  I <- (h / 2) * S
  return(I)
}

# ================================================================
# Contoh penggunaan:
# ================================================================
# Fungsi yang akan diintegralkan
f <- function(x) exp(-x^2) + 2*x

# Batas integrasi
a <- 0
b <- 2

# Uji beberapa nilai n
for (n in c(4, 8, 16, 32, 64)) {
  I_approx <- trapezoid(f, a, b, n)
  cat(sprintf("n = %2d | Hasil aproksimasi: %.6f\n", n, I_approx)
     )
}

# ================================================================
# Perbandingan dengan hasil eksak bawaan R
# ================================================================
I_exact <- integrate(f, lower = a, upper = b)$value
cat(sprintf("\nNilai integral eksak (fungsi integrate): %.6f\n",
   I_exact))
```

4.5. Penjelasan Kode

- Variabel `h` menyatakan panjang panel $(b-a)/n$.
- Baris `S <- f(a) + f(b)` memulai jumlah dengan nilai fungsi di ujung-ujung selang.

- Perulangan `for` menambahkan dua kali lipat semua titik tengah (koefisien 2).
- Hasil akhir diperoleh dengan mengalikan $(h/2)$.

4.6. Hasil Contoh

```
n =  4 | Hasil aproksimasi: 4.746483
n =  8 | Hasil aproksimasi: 4.754310
n = 16 | Hasil aproksimasi: 4.755753
n = 32 | Hasil aproksimasi: 4.756102
n = 64 | Hasil aproksimasi: 4.756184

Nilai integral eksak (fungsi integrate): 4.756200
```

Hasilnya semakin mendekati nilai eksak seiring bertambahnya jumlah panel n

4.7. Asal-usul Simpson 1/3

Jika kita ingin hasil yang lebih akurat, kita bisa menghampiri $f(x)$ di tiap dua panel (tiga titik berurutan) dengan **parabola kuadratik** alih-alih garis lurus.

Gunakan tiga titik (x_0, f_0), (x_1, f_1), dan (x_2, f_2) yang berjarak sama h. Interpolasi Lagrange orde dua:

$$P_2(x) = f_0\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f_1\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f_2\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}.$$

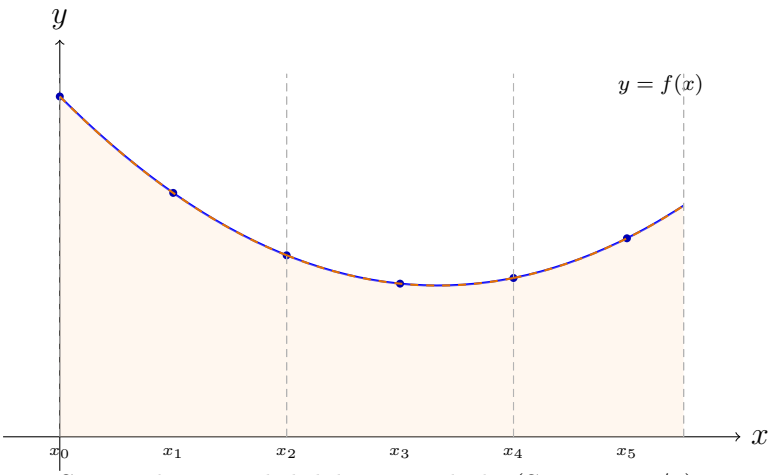
Jika diintegralkan dari x_0 ke x_2 dengan jarak antar titik h, akan diperoleh:

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}\big[f_0 + 4f_1 + f_2\big].$$

Dengan menjumlahkan semua pasangan panel (jumlah n harus genap), diperoleh **Simpson 1/3 Majemuk**:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + f(b) + 4\sum_{i=1,3,\ldots}^{n-1} f(a+ih) + 2\sum_{i=2,4,\ldots}^{n-2} f(a+ih)\right].$$

4.8. Ilustrasi Simpson 1/3



Setiap dua panel didekati parabola (Simpson 1/3)

4.9. Algoritma Simpson 1/3 Majemuk

1. Pastikan jumlah panel n genap.
2. Hitung $h = (b - a)/n$.
3. Inisialisasi $S = f(a) + f(b)$.
4. Untuk $i = 1$ sampai $n - 1$:
   - Jika i ganjil, tambahkan $4f(a + ih)$.
   - Jika i genap, tambahkan $2f(a + ih)$.
5. Hasil akhir: $(h/3) \times S$.

4.10. Kode R: Simpson 1/3 Majemuk

```r
# ================================================================

simpson13 <- function(f, a, b, n = 10, show_steps = FALSE) {
  # --- Validasi input ---
  if (!is.function(f)) stop("Argumen f harus berupa fungsi.")
  if (n %% 2 != 0) stop("Untuk Simpson 1/3, n harus GENAP.")
  if (n < 2) stop("n minimal 2 (dan harus genap).")

  # --- Panjang panel ---
  h <- (b - a) / n

  # --- Inisialisasi jumlah ---
```

```r
  S <- f(a) + f(b)

  # --- Perulangan untuk titik dalam ---
  for (i in 1:(n - 1)) {
    x_i <- a + i * h
    coef <- if (i %% 2 == 0) 2 else 4
    S <- S + coef * f(x_i)
    if (show_steps) {
      cat(sprintf("i=%2d | x_i=%.4f | f(x_i)=%.6f | koef= %d\n",
                  i, x_i, f(x_i), coef))
    }
  }

  # --- Rumus akhir Simpson 1/3 ---
  I <- (h / 3) * S
  return(I)
}

# ================================================================
# Contoh penggunaan:
# ================================================================
# Fungsi yang akan diintegralkan
f <- function(x) exp(-x^2) + 2*x

# Batas integrasi
a <- 0
b <- 2

# Uji beberapa nilai panel (harus genap)
for (n in c(4, 6, 8, 10, 20, 40)) {
  I_approx <- simpson13(f, a, b, n)
  cat(sprintf("n = %2d | Hasil : %.6f\n", n, I_approx))
}

# ================================================================
# Perbandingan dengan hasil eksak bawaan R
# ================================================================
I_exact <- integrate(f, lower = a, upper = b)$value
cat(sprintf("\nNilai integral eksak (fungsi integrate): %.6f\n",
   I_exact))
```

4.11. Hasil Contoh

```
n =  4 | Hasil aproksimasi: 4.756048
n =  6 | Hasil aproksimasi: 4.756162
n =  8 | Hasil aproksimasi: 4.756189
n = 10 | Hasil aproksimasi: 4.756195
n = 20 | Hasil aproksimasi: 4.756199
n = 40 | Hasil aproksimasi: 4.756200

Nilai integral eksak (fungsi integrate): 4.756200
```

Semakin banyak panel n, hasilnya makin mendekati nilai eksak.

4.12. Penjelasan Kode

- Jumlah panel n harus genap karena tiap parabola mencakup dua sub-selang.
- menentukan apakah indeks genap atau ganjil untuk pemberian koefisien 2 atau 4.
- Rumus akhir `return((h/3) * S)` sesuai langsung dengan rumus matematik Simpson 1/3.

4.13. Perbandingan Akurasi

Untuk fungsi halus, metode Simpson 1/3 memberikan tingkat akurasi yang jauh lebih baik daripada Trapesium, karena menggunakan parabola sebagai pendekatan lokal terhadap fungsi. Namun, metode ini memerlukan jumlah panel genap.