SK5002 ALGORITMA DAN RANCANGAN PERANGKAT LUNAK

Ujian Tengah Semester

Mohammad Rizka Fadhli
NIM: 20921004

14 October 2021

Contents

List of Figures

1 PENDAHULUAN

1.1 Bahasa Pemrograman yang Dipakai

Bahasa pemrograman yang digunakan pada tugas ini adalah **R** versi `4.1.1`. Format tugas ini ditulis menggunakan *LaTex* **R Markdown** di *software* **R Studio**.

1.2 *Libraries* R yang Digunakan

Berikut adalah beberapa *libraries* yang digunakan dalam mengerjakan dan menuliskan tugas ini:

1. `dplyr`: untuk *data carpentry*.
2. `ggplot2`: sebagai visualisasi data (grafik).

1.3 Program

Program yang digunakan untuk menjawab soal akan di kirimkan sebagai lampiran dan ditunjukkan dalam format `code` *markdown*. Agar bisa dieksekusi dengan baik, pastikan *libraries* yang terlibat sudah ter-*install* terlebih dahulu.

1.4 Pembulatan

Seluruh jawaban numerik akan ditampilkan menggunakan **delapan angka berarti**.

1.5 Lampiran Program

1.5.1 Program untuk Menggambar Fungsi

Berikut adalah program yang saya buat untuk menggambar fungsi:

```
gambar_grafik = function(x_lower,x_upper, # selang x
                         y_lower,y_upper, # selang y
                         delta, # selang gambar
                         f){ # fungsi f(x,y)
  # generate selang
  selang_x = seq(x_lower,x_upper,by = delta)
  selang_y = seq(y_lower,y_upper,by = delta)

  # menghitung (x,y) yang memenuhi f(x,y) = 1
  df =
```

```
    # mengeluarkan semua kombinasi yang mungkin dari selang
    expand.grid(selang_x,selang_y) %>%
    as.data.frame() %>%
    # mengubah nama variabel menjadi x,y
    rename(x = Var1,
           y = Var2) %>%
    # menghitung nilai f(x,y)
    mutate(f = f(x,y)) %>%
    # hanya mengambil (x,y) yang memenuhi f(x,y) = 1
    filter(round(f,2) == 1)
  # membuat grafik
  df %>%
    ggplot(aes(x,y)) +
    geom_point(size = .1,
               color = "steelblue") +
    theme_minimal() +
    geom_vline(xintercept = 0,color = "black") +
    geom_hline(yintercept = 0,color = "black") +
    labs(title = "Grafik f(x,y)",
         caption = "Digambar dengan R\n20921004@mahasiswa.itb.ac.id")
}
```

1.5.2 Program `luastrap`

Program untuk menghitung luas area di bawah kurva dengan metode trapesium saya lampirkan pada jawaban **Sub Soal Ia i**.

1.5.3 Program `luasmc`

Program untuk menghitung luas area di bawah kurva dengan metode Monte Carlo saya lampirkan pada jawaban **Sub Soal Ia ii**.

1.5.4 Program `numode`

Program `numode` merupakan program untuk menemukan solusi numerik dari persamaan diferensial. Ditulis berdasarkan metode **Runge Kutta** order 4. Program ini dilampirkan pada jawaban **Sub Soal Ib i**.

ALGORITMA KOMPUTASI NUMERIK

2 SOAL 1a

2.1 Soal Utama

Diketahui sebuah fungsi:

$$\frac{(x+1)^2}{16} + \frac{(y-1)^2}{8} = 1$$

2.1.1 Sub Soal Ia i

Gambarlah fungsi tersebut. Hitunglah luas area di bawah kurva pada kuadran pertama untuk nilai $x \in [0, \sqrt{14} - 1]$ dengan metode partisi trapesium.

2.1.2 Jawaban Sub Soal Ia i

Gambar Fungsi Berikut adalah gambar fungsi yang saya buat dengan **R**.

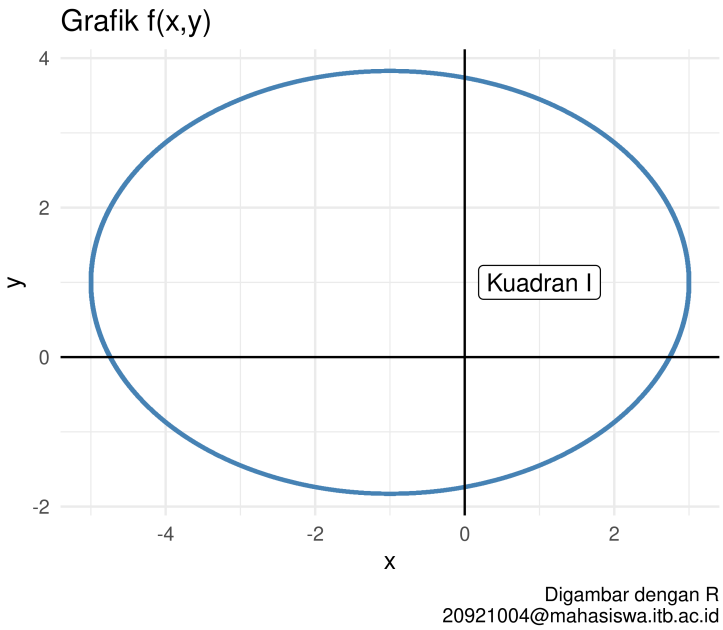

Figure 1: Grafik f(x,y)

Sekarang kita akan menghitung luas area pada kuadran I di selang $x \in [0, \sqrt(14) - 1]$. Saya akan gambarkan selang tersebut dengan garis merah sebagai berikut:

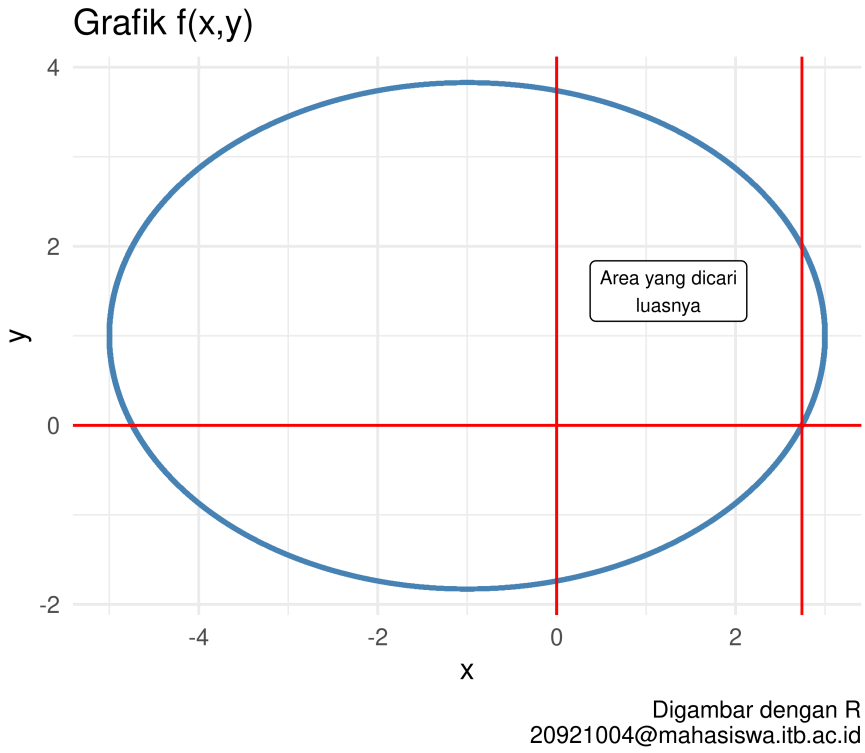

Figure 2: Area Luas

Mengubah Fungsi Untuk memudahkan, kita perlu memodifikasi fungsi $f(x, y)$ ke dalam bentuk $y = g(x)$ yang lebih sederhana.

$$y = 1 + \sqrt{8 - \frac{(x+1)^2}{2}}$$

Karena kita akan menghitung luas area di kuadran I, maka nilai akar yang dihasilkan kita akan ambil hanya yang bernilai **positif** saja.

Luas Area di Bawah Kurva Ide dasar untuk menghitung luas area di bawah kurva adalah:

$$L = alas \times tinggi$$

Pada partisi trapesium, tinggi yang akan digunakan adalah: $tinggi = \frac{f(x_1)+f(x_2)}{2}$

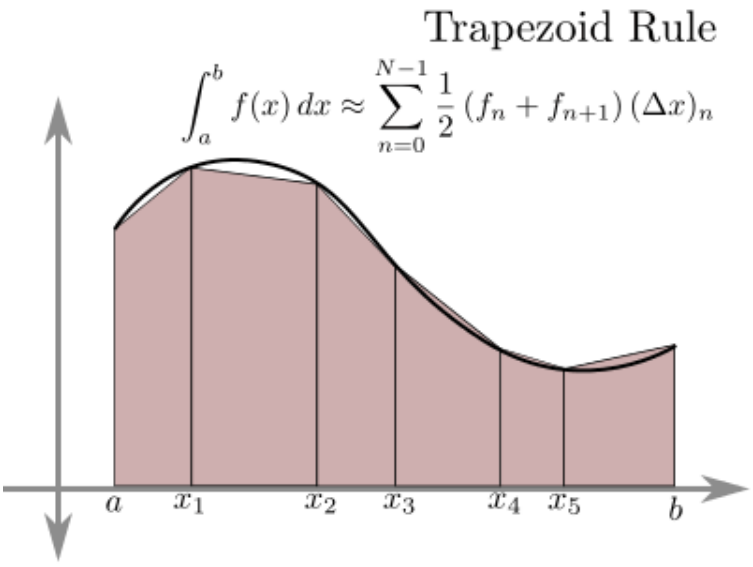

Figure 3: sumber: bragitoff.com

Pada metode trapesium ini, penentuan berapa banyak selang akan mempengaruhi seberapa akurat hasilnya.

Barikut adalah program **luastrap** yang saya buat di **R**:

```
luas_trap = function(x0, # titik awal
                     xn, # titik akhir
                     n,  # banyak selang
                     f){ # fungsi y = f(x)
  # menghitung delta x
  h = (xn - x0) / n
  # menghitung f di x0
  f0 = f(x0)
  # selang pertama
  i = 1
  k = x0 + i*h
  fn = f(k)
  integration = (f0+fn)/2
  # iterasi untuk selang berikutnya hingga selesai
  for(i in 2:n){
```

```
    f0 = fn
    k = x0 + i*h
    fn = f(k)
    temp = (f0+fn)/2
    integration = integration + temp
  }
  # menghitung hampiran luas
  integration = integration * h
  return(integration)
}
```

Sekarang kita akan bandingkan hasilnya untuk berbagai banyak selang.

```
N = c(10,50,100,200,1000,2500,5000,100000,250000,500000,750000,1000000)
Luas = c()
for(i in 1:length(N)){
  Luas[i] = luas_trap(0,
                      sqrt(14)-1,
                      N[i],
                      g)
}
```

Table 1: Hasil Perhitungan Luas Trapesium

n banyak selang	Luas aproksimasi
10	8.64494288
50	8.65494631
100	8.65526330
200	8.65534260
1000	8.65536798
2500	8.65536887
5000	8.65536899
100000	8.65536904
250000	8.65536904
500000	8.65536904
750000	8.65536904
1000000	8.65536904

Terlihat bahwa semakin banyak selangnya, hasilnya konvergen ke suatu nilai yang sama yakni: **8.65536904**.

2.1.3 Sub Soal Ia ii

Buatlah algoritma dan *flowchart* untuk menghitung luas soal sebelumnya dengan metode Monte Carlo. Lakukan analisa hasil yang diperoleh dengan jumlah sampling yang diberikan. Anggaplah perhitungan analitis adalah yang benar sehingga merupakan rujukan nilai.

2.1.4 Jawaban Sub Soal Ia ii

Perhitungan Analitis Kita bisa menghitung secara analitis luas area di bawah kurva dengan cara melakukan integral tentu berikut ini:

$$\int_0^{\sqrt{14}-1} 1 + \sqrt{8 - \frac{(x+1)^2}{2}}\ \mathrm{dx} \simeq 8.65536904$$

Metode Monte Carlo Analogi dari metode ini adalah seperti melempar sekian banyak *darts* ke suatu target. Luas area di bawah kurva didefinisikan sebagai rasio dari banyaknya *darts* yang jatuh di bawah kurva dengan total semua *darts* yang dilempar.

Berikut adalah *flowchart*-nya:

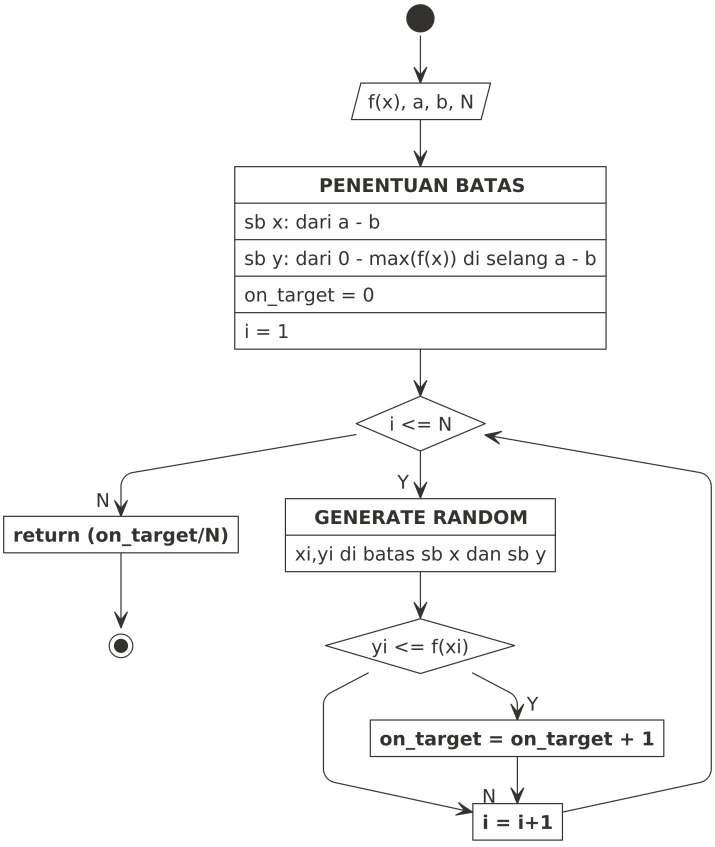

Figure 4: Flowchart Brute Force Monte Carlo

Hal terpenting dalam metode ini adalah **mendefinisikan batas titik** x, y untuk di-*random*. Kenapa?

> Kita tidak ingin *darts* yang kita lempar jatuh ke area sembarang! Kita harus definisikan di mana **area bermain** *darts*.

Perhatikan kembali grafik di bawah ini:

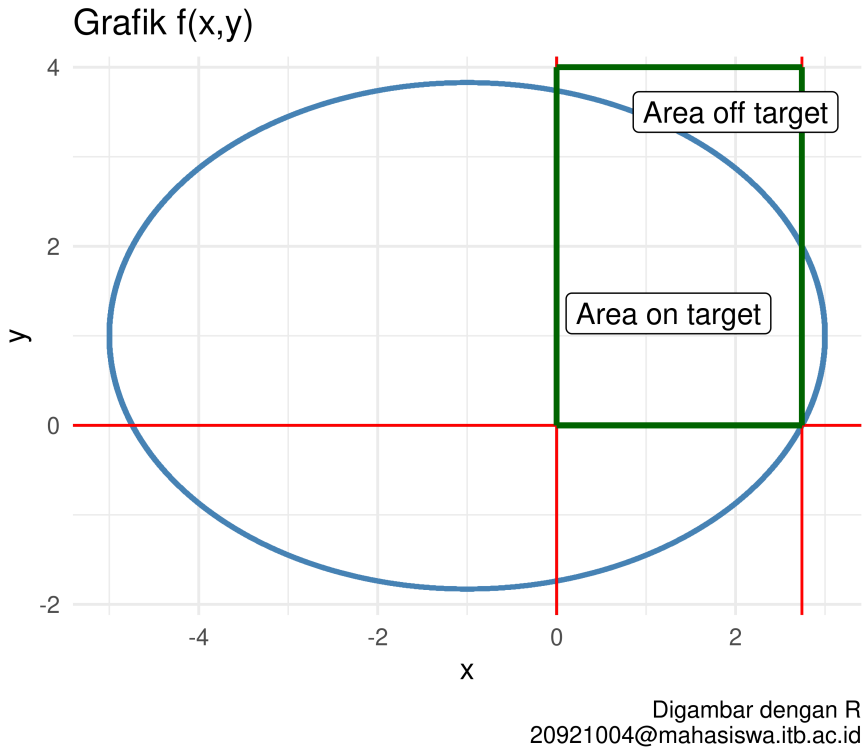

Figure 5: Area Monte Carlo

Saya akan menjadikan area di **dalam kotak warna hijau** sebagai area random titik metode Monte Carlo. Jika titik tersebut jatuh ke bawah kurva, maka akan dihitung sebagai **on target**. Jika jatuh di atasnya, berarti **off target**.

Kelak luas akan dihitung dengan cara:

$$L = 4 \times (\sqrt{14} - 1) \times ratio$$

Berikut adalah programnya dalam **R**:

```
brute_force = function(f,x1,x2,y1,y2,N){
  # generating random number
  x = runif(N,x1,x2)
  y = runif(N,y1,y2)

  # pengecekan y <= f(x)
  rekap =
    data.frame(x,y) %>%
    mutate(f_x = f(x),
           on_target = ifelse(y <= f_x,1,0))

  # hitung rasio on target vs all dots
  rasio = sum(rekap$on_target) / N
  # hitung luas
  luas = (x2-x1)*(y2-y1)*rasio

  # perbandingan dengan eksak
  eksak = 8.65536904
  delta = abs(eksak - luas)

  # output plot
  plot_sim =
    soal +
    geom_point(data = rekap,aes(x,y,color = on_target)) +
    theme(legend.position = "none") +
    labs(subtitle = paste0("Didapat nilai rasio sebesar ",rasio))

  # output
  output = list(
    "Plot Brute Force" = plot_sim,
    "Luas area di bawah kurva" = luas,
    "Absolute selisih dg solusi eksak" = delta
    )

  return(output)
}
```

Saya menghitung *error* atau **selisih solusi numerik dengan solusi eksak** dengan cara:

$$\Delta = |eksak - numerik|$$

Sekarang kita akan coba program tersebut untuk N = 100

```
## $`Plot Brute Force`
```

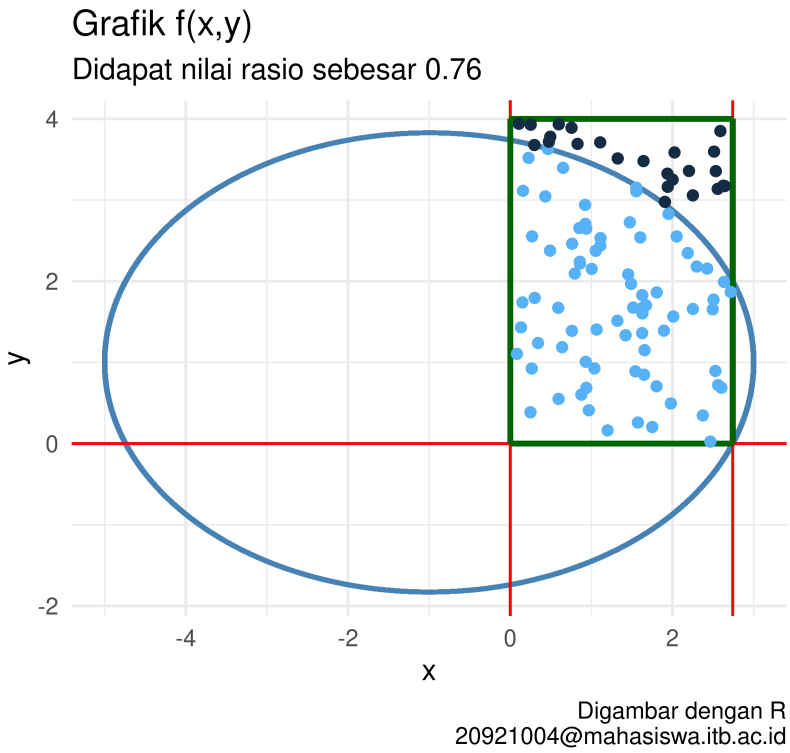

Figure 6: Brute force Monte Carlo dengan N = 100

```
##
## $`Luas area di bawah kurva`
## [1] 8.33463846
##
## $`Absolute selisih dg solusi eksak`
## [1] 0.320730584
```

Sekarang kita akan coba program tersebut untuk N = 200

```
## $`Plot Brute Force`
```

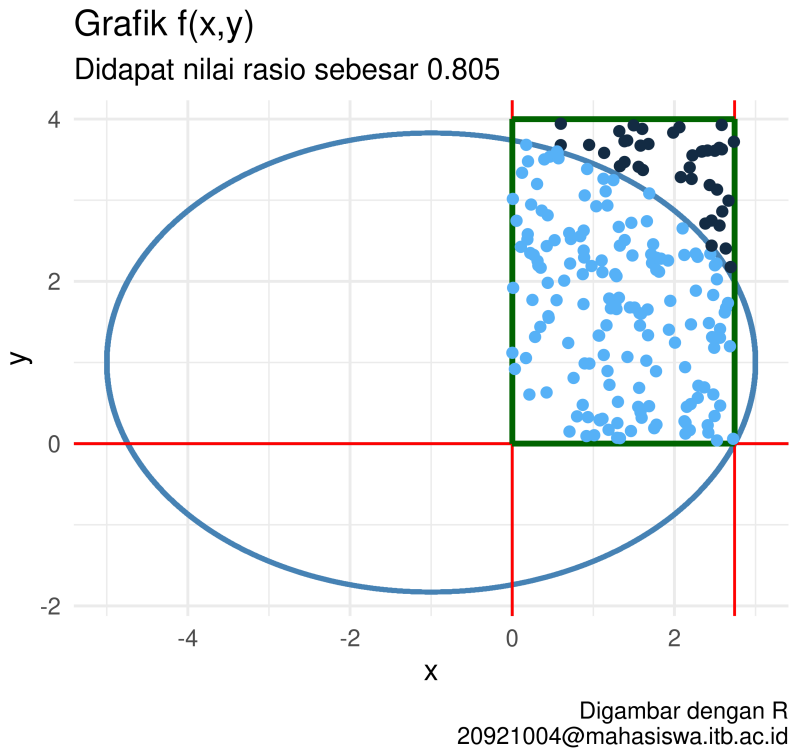

Figure 7: Brute force Monte Carlo dengan N = 200

```
##
## $`Luas area di bawah kurva`
## [1] 8.82813679
##
## $`Absolute selisih dg solusi eksak`
## [1] 0.172767745
```

Sekarang kita akan coba program tersebut untuk N = 500

```
## $`Plot Brute Force`
```

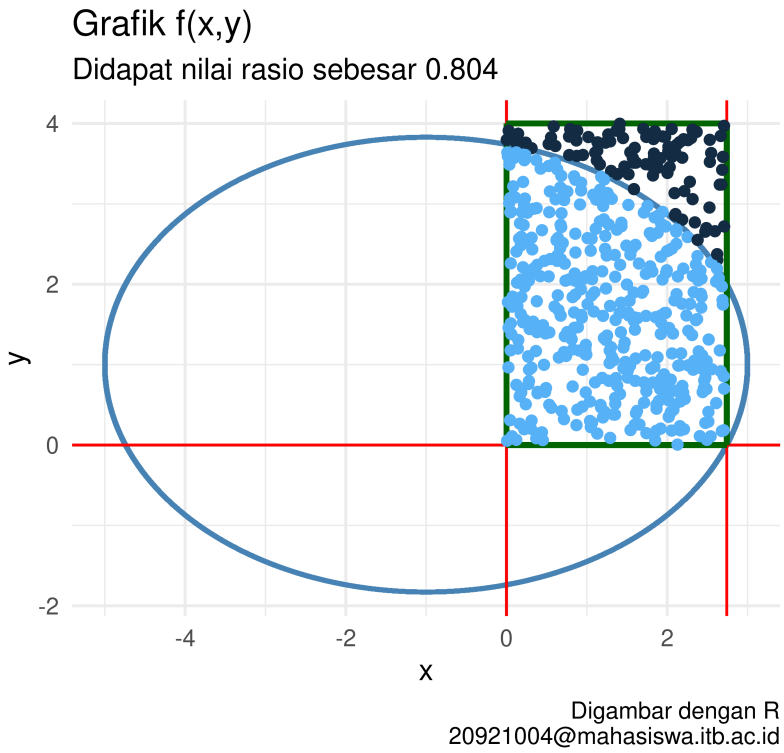

Figure 8: Brute force Monte Carlo dengan N = 500

```
##
## $`Luas area di bawah kurva`
## [1] 8.81717016
##
## $`Absolute selisih dg solusi eksak`
## [1] 0.161801116
```

Masalah pada Brute Force Salah satu prinsip metode ini adalah *random bumber generator*. Oleh karena itu, bisa jadi pada **N** besar hasilnya tidak lebih baik pada **N** kecil **dalam sekali run**.

Untuk mengatasi hal tersebut kita harus melakukan *run* berulang-ulang dan menghitung nilai rata-ratanya sebagai *output* finalnya. Saya akan coba melakukan pengulangan sebanyak **100 kali** setiap kali *run* untuk mendapatkan aproksimasi yang lebih baik.

Berikut adalah modifikasi program sebelumnya yang saya berikan nama **luasmc**.

```
luas_mc = function(f,x1,x2,y1,y2,N){
  # membuat template vector luas
  luas = c()
  # lakukan 100 x pengulangan
  for(ikanx in 1:100){
    # generating random number
    x = runif(N,x1,x2)
    y = runif(N,y1,y2)

    # pengecekan y <= f(x)
    rekap =
      data.frame(x,y) %>%
      rowwise() %>%
      mutate(f_x = f(x),
             on_target = ifelse(y <= f_x,1,0)) %>%
      ungroup()

    # hitung rasio on target vs all dots
    rasio = sum(rekap$on_target) / N
    # hitung luas
    luas_temp = (x2-x1)*(y2-y1)*rasio
    # memasukkan ke dalam template
    luas = c(luas,luas_temp)
  }

  # menghitung rata-rata luas
  return(mean(luas))
}
```

Kita akan coba hitung untuk nilai **N** yang berbeda-beda sebagai berikut:

```
N = c(10,50,100,200,500,1000,2500,5000,10000,20000)
Luas = c()
for(i in 1:length(N)){
  Luas[i] = luas_mc(g,0,sqrt(14)-1,0,4,N[i])
}
```

Table 2: Hasil Perhitungan Luas Monte Carlo

n banyak titik	Luas aproksimasi	Absolute selisih dengan eksak
10	8.65267071	0.002698327
50	8.71847049	0.063101450
100	8.66583067	0.010461628
200	8.69379557	0.038426533
500	8.67855196	0.023182918
1000	8.63402744	0.021341598
2500	8.65622390	0.000854861
5000	8.64681453	0.008554508
10000	8.65381124	0.001557798
20000	8.65319711	0.002171929

Kita bisa lihat bahwa ada indikasi semakin tinggi **N** yag digunakan, nilai absolut selisih aproksiasi dengan eksak relatif semakin kecil.

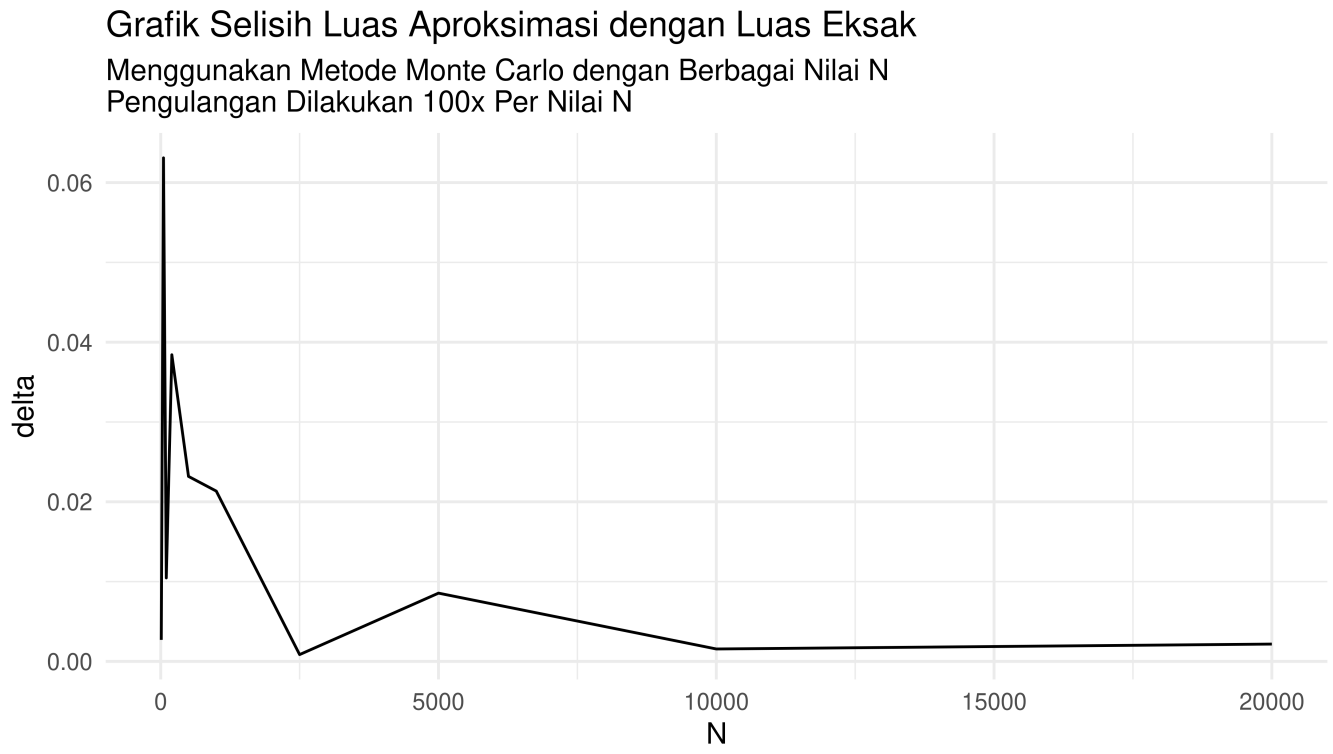

Figure 9: Plot Selisih Aproksimasi vs Eksak

Namun agar metode ini lebih **stabil** dan konvergen, kita bisa mengulang komputasi lebih banyak per nilai **N** dengan konsekuensi *runtime* yang semakin panjang. Pada soal ini, saya menggunakan 100 x pengulangan.

3 SOAL 1b

3.1 Soal Utama

Diketahui persamaan diferensial sebagai berikut:

$$\frac{dy}{dx} = (x^2 + y)\sin(x^2 y)$$

3.1.1 Sub Soal Ib i

Pilih sebuah metode numerik untuk menyelesaikan suatu persamaan diferensial dan rancanglah algoritma dari metode tersebut!

3.1.2 Jawaban Sub Soal Ib i

Dari suatu persamaan diferensial dengan bentuk:

$$\frac{dy}{dx} = f(x, y)$$

dan memiliki *initial condition* $y(x_0) = y_0$. Kita bisa menyelesaikannya dengan metode **Runge-Kutta** order 4. Bentuk umumnya adalah sebagai berikut:

$$y_{n+1} = y_n + h\sum_{i=1}^{n} b_i k_i$$

dimana $k_i, i = 1, 2, 3, 4$ adalah konstanta yang harus dicari.

$$k_1 = f(x_0, y_0)$$

$$k_2 = f(x_0 + 0.5h, y_0 + 0.5k_1 h)$$

$$k_3 = f(x_0 + 0.5h, y_0 + 0.5k_2 h)$$

$$k4 = f(x_0 + h, y_0 + k_3 h)$$

Bentuk algoritmanya dalam *pseudocode* adalah sebagai berikut:

```
INPUT x0,y0,xmax

COMPUTE n = (xmax-x0)/h

FOR i 1:n
    k1 = f(x0,y0)
    k2 = f(x0 + 0.5*h,y0 + 0.5*k1*h)
    k3 = f(x0 + 0.5*h,y0 + 0.5*k2*h)
    k4 = f(x0 + h,y0 + k3*h)
    y0 = y0 + (1/6)*(k1 + 2*k2 + 2*k3 + k4) * h
    x0 = x0 + h
```

Jika dibuat dalam program **R**:

```r
rk_4order = function(f,       # dy/dx
                     x0, y0, # init condition
                     h,       # selang
                     xmax){  # x max
  # initial condition
  x = x0
  y = y0
  n = (xmax-x0)/h
  # proses iterasi
  for(i in 1:n){
    k1 = f(x0,y0)
    k2 = f(x0 + 0.5*h,y0 + 0.5*k1*h)
    k3 = f(x0 + 0.5*h,y0 + 0.5*k2*h)
    k4 = f(x0 + h,y0 + k3*h)
    y0 = y0 + (1/6)*(k1 + 2*k2 + 2*k3 + k4) * h
    x0 = x0 + h
    x = c(x, x0)
    y = c(y, y0)
  }
  # output
  output = data.frame(x = x,
                      y = y)
  return(output)
}
```

Saya *save* program tersebut dengan nama **numode**.

3.1.3 Sub Soal Ib ii

Tentukanlah solusi persamaan diferensial di atas untuk rentang $0 \leq x \leq 2$ dengan kondisi awal $y(0) = 5$ dan rentang partisi $h = 0.2$.

3.1.4 Jawaban Sub Soal Ib ii

Mari kita selesaikan:

```
dydx = function(x,y){(x^2 + y)*sin((x^2) * y)}
x0 = 0
y0 = 5
xmax = 2
h = 0.2
solusi = rk_4order(dydx,x0,y0,h,xmax)
```

Table 3: Solusi Persamaan Diferensial dengan RK4

x	y
0.0	5.00000000
0.2	5.06760128
0.4	5.57269164
0.6	6.71789477
0.8	6.45613761
1.0	5.18752961
1.2	4.67362560
1.4	5.16720280
1.6	4.05194257
1.8	3.20113263
2.0	2.66358548

Dalam bentuk grafik:

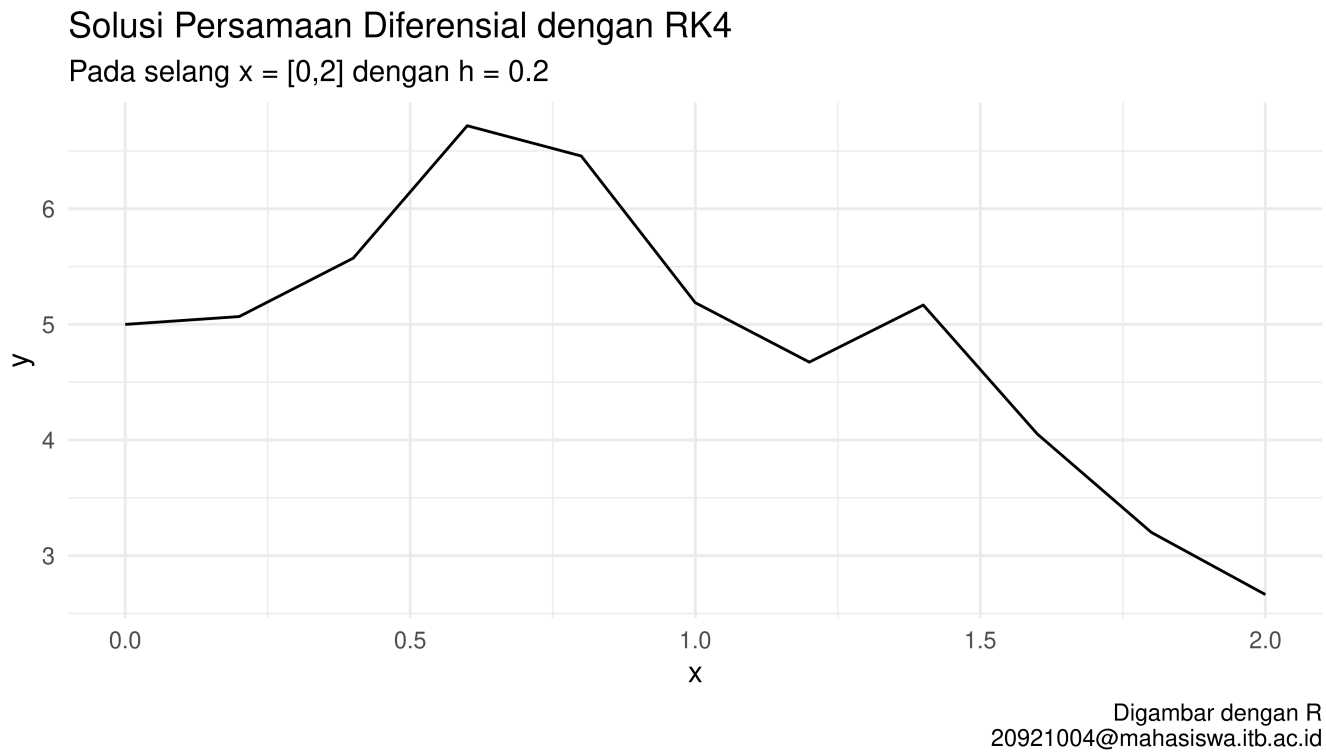

Figure 10: Solusi dy/dx pada [0,2] dengan h = 0.2

3.1.5 Sub Soal Ib iii

Bandingkan kurva solusinya (kurva y vs x) jika digunakan nilai $h = 0.1$ dan $h = 0.01$.

3.1.6 Jawaban Sub Soal Ib iii

Mari kita selesaikan:

```
solusi_h1 = rk_4order(dydx,x0,y0,0.1,xmax)
solusi_h2 = rk_4order(dydx,x0,y0,0.01,xmax)
```

Berikut grafik perbandingannya:

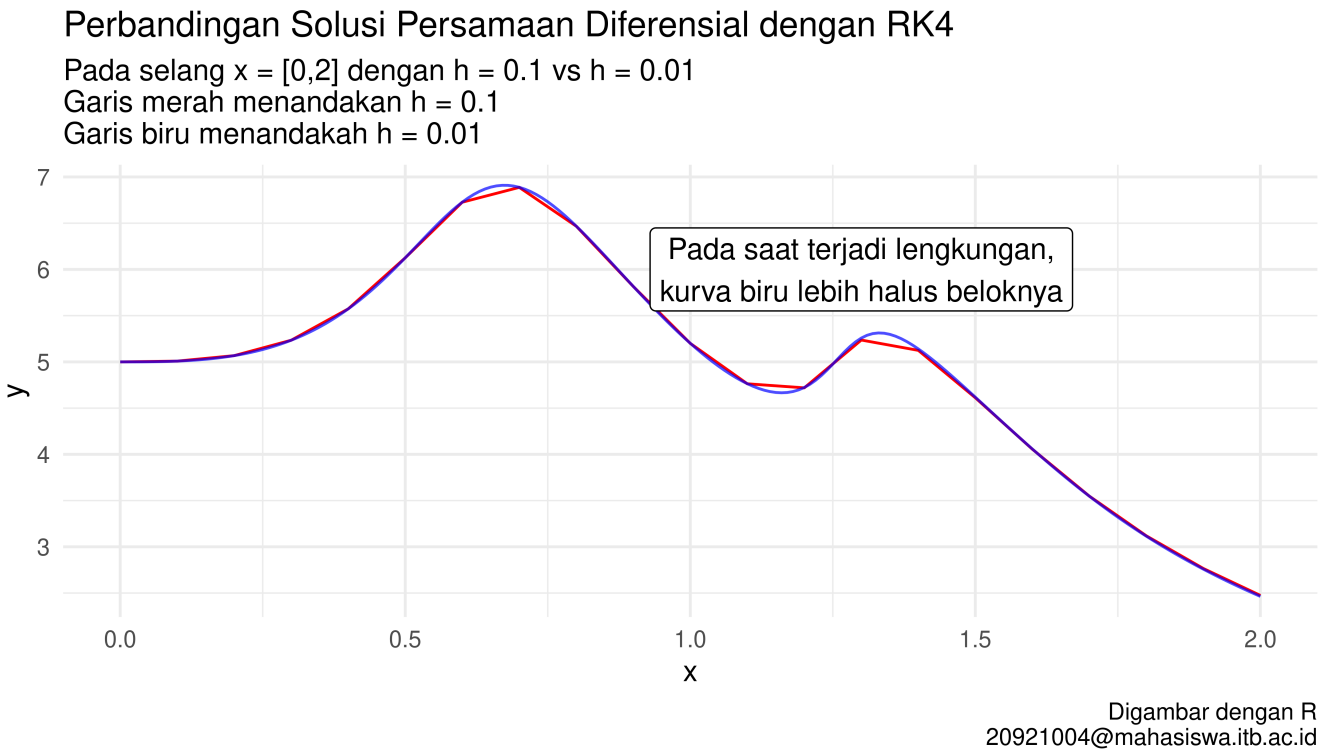

Figure 11: Solusi dy/dx pada [0,2] dengan h = 0.1 vs h = 0.01

Secara visual dan intuitif bisa kita simpulkan bahwa semakin kecil h, maka nilai hampirannya lebih baik karena penambahan x terjadi secara perlahan.

SHORTEST ALGORITHM

4 Soal 2

4.1 Soal Utama

Perhatikan graf berikut ini:

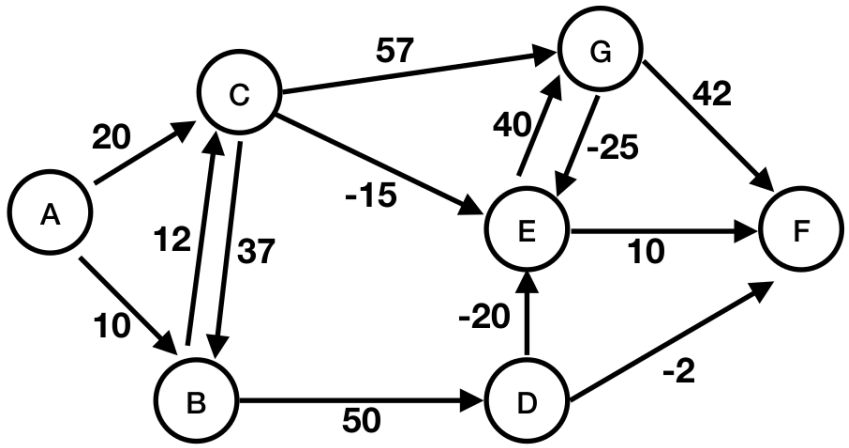

Figure 12: Graf Soal

4.1.1 Sub Soal 2i

Jika **vertex A** merupakan titik sumber, tentukanlah lintasan minimum yang mungkin untuk graf tersebut! Apakah graf memiliki *loop* negatif?

4.1.2 Jawaban Sub Soal 2i

Untuk menentukan *shortest path* dari **A**, kita akan menggunakan algoritma **Bellman-Ford** dengan aturan *relaxation* sebagai berikut:

```
if d[u] + c(u,v) < d[v]
   d[v] = d[u] + c(u,v)
```

Untuk menyelesaikannya, graf di atas saya tuliskan sebagai bentuk tabel berikut:

```
##    from to bobot
## 1     A  B    10
## 2     A  C    20
```

```
## 3     B  C    12
## 4     C  B    27
## 5     B  D    50
## 6     D  E   -20
## 7     C  E   -15
## 8     C  G    57
## 9     E  G    40
## 10    G  E   -25
## 11    G  F    42
## 12    E  F    10
## 13    D  F    -2
```

Dengan bobot *vertex* awal sebagai berikut:

```
d_titik = data.frame(titik = c("A","B","C",'D','E','F','G'),
                     bobot = c(0,rep(Inf,6))
                     )
d_titik
```

```
##   titik bobot
## 1     A     0
## 2     B   Inf
## 3     C   Inf
## 4     D   Inf
## 5     E   Inf
## 6     F   Inf
## 7     G   Inf
```

Jika dijalankan algoritma **Bellman-Ford** didapatkan *update* bobot sebagai berikut:

```
## [[1]]
## [[1]]$`iterasi ke: `
## [1] 1
##
## [[1]]$`hasil bobot vertex`
##   titik bobot
## 1     A     0
## 2     B    10
## 3     C    20
## 4     D   Inf
## 5     E   Inf
## 6     F   Inf
## 7     G   Inf
```

```
##
##
## [[2]]
## [[2]]$`iterasi ke: `
## [1] 2
##
## [[2]]$`hasil bobot vertex`
##   titik bobot
## 1     A     0
## 2     B    10
## 3     C    20
## 4     D    60
## 5     E   Inf
## 6     F   Inf
## 7     G   Inf
##
##
## [[3]]
## [[3]]$`iterasi ke: `
## [1] 3
##
## [[3]]$`hasil bobot vertex`
##   titik bobot
## 1     A     0
## 2     B    10
## 3     C    20
## 4     D    60
## 5     E     5
## 6     F   Inf
## 7     G    77
##
##
## [[4]]
## [[4]]$`iterasi ke: `
## [1] 4
##
## [[4]]$`hasil bobot vertex`
##   titik bobot
## 1     A     0
## 2     B    10
## 3     C    20
## 4     D    60
## 5     E     5
## 6     F    58
## 7     G    77
```

```
##
##
## [[5]]
## [[5]]$`iterasi ke: `
## [1] 5
##
## [[5]]$`hasil bobot vertex`
##   titik bobot
## 1     A     0
## 2     B    10
## 3     C    20
## 4     D    60
## 5     E     5
## 6     F    15
## 7     G    45
##
##
## [[6]]
## [[6]]$`iterasi ke: `
## [1] 6
##
## [[6]]$`hasil bobot vertex`
##   titik bobot
## 1     A     0
## 2     B    10
## 3     C    20
## 4     D    60
## 5     E     5
## 6     F    15
## 7     G    45
```

Setelah iterasi ke kelima, bobot *vertex* **tetap** (tidak berubah). Sehingga bobot tersebut adalah bobot final *shortest path* dari *vertex* A.

Berikut adalah gambar rutenya:

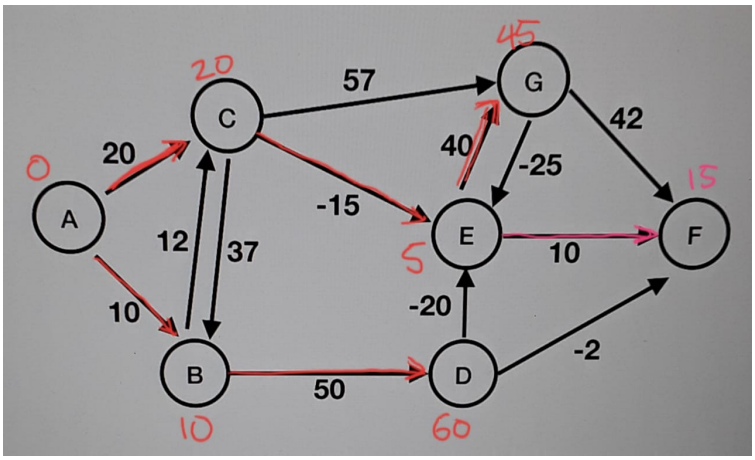

Figure 13: Shortest Path dari A

Terlihat juga bahwa **tidak ada loop negatif** pada graf.

4.1.3 Sub Soal 2ii

Gambarkan lintasan minimumnya jika *vertex* B menjadi titik sumber dan bobot (e, g) menjadi -10! Apakah graf memiliki *loop* negatif?

4.1.4 Jawaban Sub Soal 2ii

Graf dengan bobot *edge* yang ter-*update* saya buat dalam bentuk tabel berikut:

```
##    from to bobot
## 1     A  B    10
## 2     A  C    20
## 3     B  C    12
## 4     C  B    27
## 5     B  D    50
## 6     D  E   -20
## 7     C  E   -15
## 8     C  G    57
## 9     E  G   -10
## 10    G  E   -25
## 11    G  F    42
## 12    E  F    10
## 13    D  F    -2
```

Sedangkan ini adalah tabel dari bobot semua *vertex*:

```
d_titik = data.frame(titik = c("B","C",'D','E','F','G',"A"),
                     bobot = c(0,rep(Inf,6))
                     )
d_titik
```

```
##   titik bobot
## 1     B     0
## 2     C   Inf
## 3     D   Inf
## 4     E   Inf
## 5     F   Inf
## 6     G   Inf
## 7     A   Inf
```

Jika kita jalankan algoritma **Bellman-Ford** didapatkan:

```
## [[1]]
## [[1]]$`iterasi ke: `
## [1] 1
##
## [[1]]$`hasil bobot vertex`
##   titik bobot
## 1     B     0
## 2     C    12
## 3     D    50
## 4     E   -38
## 5     F     7
## 6     G   -13
## 7     A   Inf
##
##
## [[2]]
## [[2]]$`iterasi ke: `
## [1] 2
##
## [[2]]$`hasil bobot vertex`
##   titik bobot
## 1     B     0
## 2     C    12
## 3     D    50
## 4     E   -73
## 5     F   -28
## 6     G   -48
```

```
## 7     A   Inf
##
##
## [[3]]
## [[3]]$`iterasi ke: `
## [1] 3
##
## [[3]]$`hasil bobot vertex`
##   titik bobot
## 1     B     0
## 2     C    12
## 3     D    50
## 4     E  -108
## 5     F   -63
## 6     G   -83
## 7     A   Inf
##
##
## [[4]]
## [[4]]$`iterasi ke: `
## [1] 4
##
## [[4]]$`hasil bobot vertex`
##   titik bobot
## 1     B     0
## 2     C    12
## 3     D    50
## 4     E  -143
## 5     F   -98
## 6     G  -118
## 7     A   Inf
##
##
## [[5]]
## [[5]]$`iterasi ke: `
## [1] 5
##
## [[5]]$`hasil bobot vertex`
##   titik bobot
## 1     B     0
## 2     C    12
## 3     D    50
## 4     E  -178
## 5     F  -133
## 6     G  -153
```

```
## 7     A   Inf
##
##
## [[6]]
## [[6]]$`iterasi ke: `
## [1] 6
##
## [[6]]$`hasil bobot vertex`
##   titik bobot
## 1     B     0
## 2     C    12
## 3     D    50
## 4     E  -213
## 5     F  -168
## 6     G  -188
## 7     A   Inf
```

Algoritma **Bellman-Ford** menghasilkan *infinite looping*, sehingga nilainya selalu menurun terus. Hal ini terjadi karena ada *loop* negatif sehingga tidak ditemukan rute yang konklusif.

Selain itu, *vertex* **A** selalu bernilai ∞ karena tidak ada jalur masuk yang bisa meng-*update* nilainya.

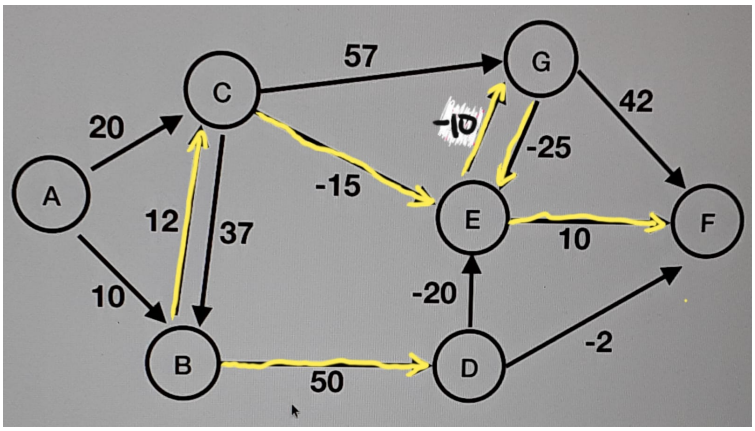

Figure 14: Shortest Path dari B

SELESAI