

SK5001 ANALISIS NUMERIK LANJUT

Catatan Kuliah
Sebuah Usaha untuk Mencatat

Ikang FADHLI
ikanx101.com

19 October 2021

Contents

MUKADIMAH	9
1 SILABUS	9
1.1 Keterangan Umum	9
1.1.1 Silabus Ringkas	9
1.1.2 Silabus Lengkap	9
1.1.3 <i>Outcomes</i>	9
1.1.4 Panduan Penilaian	10
1.2 Satuan Acara Perkuliahan	11
1.3 Buku	12
1.4 Tugas dan Koding	12
CHAPTER I	13
2 MATERI PERKULIAHAN DETAIL	13
2.1 Materi Perkuliahan SK5001 Analisis Numerik	13
3 AKAR PERSAMAAN NON LINEAR	14
3.1 Pendahuluan	14
3.1.1 <i>Intermediate Value Theorem</i> (Teorema Nilai Antara)	14
3.2 Metode <i>Bisection</i>	14
3.2.1 <i>Bisection Theorem</i>	14
3.2.2 Algoritma <i>Bisection</i>	15
3.2.3 Algoritma Formal <i>Bisection</i>	16
3.2.4 Contoh Soal	17
Soal I	17
Soal II	19
Soal III	20
3.2.5 R Function Metode <i>Bisection</i>	21
3.2.6 Tugas Kuliah Algoritma Minggu II	23
Persamaan I	23
Persamaan II	27

3.3	Hal Penting Terkait Metode Bisection	30
3.3.1	Jaminan Keberadaan Akar Persamaan	30
3.3.2	Kriteria STOP Iterasi	31
3.4	Masalah Pada <i>Bisection</i> yang Ditemukan	31
3.4.1	Teorema Penting	32
3.4.2	Contoh Soal	32
3.5	Metode Newton	33
3.5.1	Algoritma Metode Newton	34
3.5.2	Algoritma Formal Metode Newton	35
3.5.3	Kriteria Penghentian Iterasi	36
	Kriteria I	36
	Kriteria II	36
	Kriteria III	36
3.5.4	Contoh Soal	36
	Soal I	36
	Soal II	38
3.5.5	R Function Newton Method	39
3.6	<i>Refreshment</i> Newton	41
3.7	Metode <i>Golden Section Search</i>	41
3.7.1	Definisi Formal	41
3.7.2	Algoritma <i>Golden Section Search</i>	42
3.7.3	Penentuan Nilai r	43
3.7.4	Mencari Akar Persamaan $f(x)$	43
3.7.5	<i>Function</i> di R	43
3.7.6	Contoh Soal	44
	Soal I	44
CHAPTER II		48
4	NUMERICAL INTEGRATION	48
4.1	Hampiran <i>Square</i> dan <i>Trapezoid</i>	48
4.1.1	R Function Hampiran <i>Square</i> dan <i>Trapezoid</i>	48
4.1.2	Contoh Soal	50

4.2	Brute Force	51
4.2.1	<i>Flowchart Brute Force</i>	52
4.2.2	R Function Brute Force	53
4.2.3	Contoh Soal	54
4.3	Modifikasi Monte Carlo	59
4.3.1	<i>Flowchart Modifikasi Monte Carlo</i>	59
4.3.2	R Function Modifikasi Monte Carlo	60
4.3.3	Contoh Soal	60
CHAPTER III		62
5	METODE ITERATIF UNTUK SPL	62
5.1	<i>Refreshment</i> Aljabar Linear	62
5.1.1	Teorema SPL	62
5.1.2	Matriks Singular dan Tak Singular	62
5.1.3	Determinan Matriks	63
5.1.4	Sifat-Sifat Determinan Matriks	63
5.1.5	Nilai Eigen dan Vektor Eigen	64
5.1.6	Norm Vektor	64
5.1.7	Norm Matriks	65
5.1.8	Matriks Diagonal, <i>Upper</i> , dan <i>Lower</i>	66
5.1.9	Matriks Diagonal Dominan Kuat	66
5.1.10	Aljabar di R	67
5.1.11	<i>Key Take Points</i> Materi Aljabar Linear	69
5.2	Visualisasi Aljabar Linear	70
5.2.1	Contoh Matriks A_1	71
5.2.2	Contoh Matriks A_2	74
5.2.3	Contoh Matriks A_3	76
5.2.4	Contoh Matriks dengan Eigen Bilangan Kompleks	76
5.3	Pengantar Iterasi SPL	77
5.3.1	Penulisan dalam Bentuk Matriks	77
5.3.2	Kriteria Penghentian Iterasi	77
5.4	Iterasi Jacobi	78

5.4.1	Pendahuluan	78
5.4.2	Matriks Iterasi B	79
5.4.3	Skema Interasi dalam Bentuk Matriks	79
5.4.4	R Function Jacobi	80
5.5	Metode Gauss Seidel	82
5.5.1	Matriks Iterasi B	82
5.5.2	Skema Interasi dalam Bentuk Matriks	82
5.5.3	R Function Gauss Seidel	82
5.6	Metode SOR Gauss Seidel	87
5.6.1	Matriks Iterasi B	87
5.6.2	Skema Interasi dalam Bentuk Matriks	87
5.6.3	R Function SOR Gauss Seidel	87
CHAPTER IV		89
6 SISTEM PERSAMAAN NON LINEAR		89
6.1	Definisi SPNL	89
6.2	Metode Newton	89
6.2.1	Skema Iterasi Metode Newton	90
6.2.2	R Function Metode Newton SPNL	90
6.2.3	Contoh Soal	91
Soal I		91
Soal II		97
6.3	Metode Broyden	102
6.3.1	Algoritma Metode Broyden	102
6.3.2	Contoh Soal	103
Soal I		103
CHAPTER V		108
7 NUMERICAL OPTIMIZATION		108
7.1	<i>Steepest Descent</i> (Metode Gradien)	108
7.1.1	Definisi	108
7.1.2	Langkah Kerja	108

7.1.3	Algoritma Formal	109
7.1.4	Catatan Lain	109
7.2	Metode <i>Neighborhood Search</i>	110
7.2.1	Definisi	110
7.2.2	Algoritma Formal	110
7.2.3	Catatan Lain	110
7.3	Metode <i>Simulated Annealing</i>	110
7.3.1	Masalah Fisis	110
7.3.2	Algoritma Formal	111
7.3.3	Catatan Penting	111
7.3.4	Contoh Soal	111
	Soal I	111
	Soal II	114
	Soal III	117
7.4	<i>Spiral Optimization Algorithm</i>	119
7.4.1	Ilustrasi Geometris	120
7.4.2	Program <i>Spiral Optimization Algorithm</i>	136
7.4.3	Mengubah Optimisasi Menjadi Pencarian Akar	138
7.4.4	Soal Latihan	139
	Jawaban	139
CHAPTER VI		145
8 CURVE FITTING		145
8.1	Linear <i>Curve Fitting</i> untuk Dua Peubah	145
8.1.1	<i>Function</i> di R Dua Peubah	148
8.2	Linear <i>Curve Fitting</i> untuk Banyak Peubah	150
8.2.1	<i>Function</i> di R Banyak Peubah	153
8.3	Polinomial <i>Curve Fitting</i> Dua Peubah	154
8.3.1	<i>Function</i> di R Polinomial	158
8.3.2	Modifikasi Polinom Banyak Peubah	162
8.4	<i>Cubic Spline Curve Fitting</i>	163
END		164

List of Figures

1	Panduan Penilaian	10
2	Satuan Acara Perkuliahan	11
3	Ilustrasi Bisection	15
4	Algoritma Bisection	16
5	Grafik fungsi $f(x)$	17
6	Grafik fungsi $f(x)$	20
7	Grafik fungsi $f(x)$	24
8	Grafik fungsi $f(x)$	27
9	Ilustrasi Newton's Method	33
10	Algoritma Newton	35
11	Seperempat Lingkaran	50
12	Flowchart Brute Force	52
13	Alur Kerja	54
14	Grafik $f(x)$	55
15	Penentuan Batas Titik Random	56
16	Flowchart Modifikasi Monte Carlo	59
17	Sifat Norm Vektor	64
18	Sifat Norm Matriks	65
19	Norm Matriks dan Jari-jari Spektral	65
20	Menggambar Lingkaran Satuan	70
21	Transformasi Matriks A1	72
22	Letak Vektor Eigen Matriks A1	73
23	Transformasi Matriks A2	74
24	Letak Vektor Eigen Matriks A2	75
25	Transformasi Matriks A2	76
26	Matriks Jacobi Newton SPNL	90
27	Contoh Soal SPNL I	91
28	Grafik Contoh Soal	97
29	Initial Points Soal Contoh	98
30	Solusi Pertama dan Kedua dari Soal	101

31	Algoritma Metode Broyden	102
32	Grafik $f(x)$	112
33	Grafik $f(x)$	114
34	Grafik $f(x)$	115
35	Grafik $f(x,y)$	117
36	Contour Plot Soal 1: f_1	139
37	Contour Plot Soal 1: f_2	140
38	Plot Soal 1: f_1 dan f_2	141
39	Dasar Regresi Linear	146
40	Function Regresi Linear 2 Peubah	149
41	Evaluasi Polinom Base R	156
42	Evaluasi Polinom Operasi Matriks	157
43	Definisi Cubic Spline	163
44	Hasil Spline Base R	164

MUKADIMAH

1 SILABUS

1.1 Keterangan Umum

1.1.1 Silabus Ringkas

Materi kuliah meliputi metode numerik untuk sistem persamaan linear dan tak linear, optimasi numerik, serta persamaan diferensial biasa dan parsial.

1.1.2 Silabus Lengkap

Materi meliputi metode numerik untuk sistem persamaan linear dan tak linear, optimasi numerik, serta persamaan diferensial biasa dan parsial. Akan dibahas metode langsung untuk sistem persamaan linear (SPL), metode iteratif untuk SPL, hampiran nilai eigen, dan sistem persamaan tak linear. Topik optimasi numerik meliputi optimasi untuk satu dan banyak peubah tanpa kendala serta masalah kuadrat terkecil tak linear. Topik terakhir adalah metode numerik untuk persamaan diferensial biasa (PDB) dan persamaan diferensial parsial (PDP): masalah nilai awal dan nilai batas untuk PDB, metode beda hingga untuk PD linear dan tak linear, metode numerik untuk PDP, serta pengantar metode elemen hingga.

1.1.3 *Outcomes*

1. Mahasiswa memiliki wawasan dan pengetahuan mengenai berbagai metode numerik untuk menyelesaikan berbagai permasalahan penting dalam sains
2. Mahasiswa menguasai dasar-dasar mengenai metode numerik untuk menyelesaikan persamaan linear dan tak linear, optimasi numerik, serta persamaan diferensial biasa dan parsial
3. Mahasiswa memiliki gambaran mengenai berbagai teknik numerik untuk menyelesaikan persamaan linear dan tak linear, optimasi numerik, serta persamaan diferensial biasa dan parsial

1.1.4 Panduan Penilaian

Porsi Penilaian Awal Mata Kuliah
Analisa Numerik Lanjut

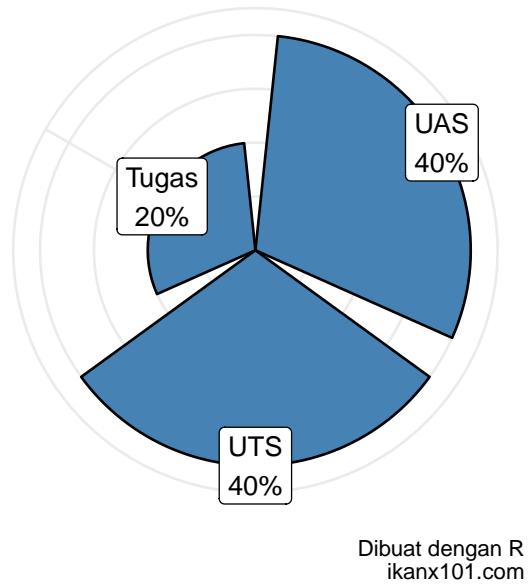


Figure 1: Panduan Penilaian

1.2 Satuan Acara Perkuliahan

Minggu	Topik	Subtopik	Capaian Belajar	Sumber Materi
1	Pendahuluan			
2	Metode langsung untuk menyelesaikan sistem persamaan linear (SPL)	eliminasi Gauss, faktorisasi Lu	Menguasai dan mengimplementasi-kan eliminasi Gauss dan faktorisasi Lu untuk menyelesaikan SPL	
3	Metode langsung untuk menyelesaikan sistem persamaan linear (SPL)	sistem positif definit, faktorisasi LDLt dan Choleski, SPL dengan matriks pita	Menguasai dan mengimplementasi-kan sistem positif definit, faktorisasi LDLt dan Choleski untuk menyelesaikan SPL dengan matriks pita	
4	Metode iteratif untuk menyelesaikan SPL	norm vektor dan matriks, jari-jari spektral matriks, metode iterasi Jacobi, metode iterasi Gauss-Seidel dan Sor, metode conjugate gradient untuk SPL	Memahami dan mengimplementasi-kan metode iteratif untuk menyelesaikan SPL	
5	Metode untuk hampiran nilai eigen	metode power, metode deflasi	Memahami dan mengimplementasi-kan metode power dan deflasi untuk menghampiri nilai eigen	
6	Metode untuk menyelesaikan sistem persamaan tak linear	iterasi titik tetap, metode Newton dan quasi-Newton	Memahami metode iterasi titik tetap, Newton dan quasi-Newton untuk menyelesaikan sistem persamaan tak linear	
7	Metode untuk menyelesaikan sistem persamaan tak linear	metode steepest descent	Memahami metode metode steepest descent untuk menyelesaikan sistem persamaan tak linear	
8	UTS			
9	Optimasi numerik satu peubah tanpa kendala	golden section search, metode Newton	Menguasai golden section search, metode Newton untuk optimasi numerik satu peubah tanpa kendala	
10	Optimasi numerik banyak peubah tanpa kendala	metode Newton, metode quasi-Newton, metode conjugate gradient	Menguasai metode Newton, metode quasi-Newton, metode conjugate gradient untuk optimasi numerik banyak peubah tanpa kendala	
11	Optimasi numerik masalah kuadrat terkecil tak linear	metode Gauss-Newton, metode Levenberg-Marquardt	Menguasai metode metode Gauss-Newton, metode Levenberg-Marquardt optimasi numerik masalah kuadrat terkecil tak linear	
12	Masalah nilai awal untuk PDB, masalah nilai batas untuk PDB	metode Euler, metode deret Taylor berderajat-n, metode Runge-Kutta	Menguasai berbagai metode untuk menyelesaikan masalah nilai awal dan masalah nilai batas untuk PDB	
13	Metode numerik untuk PDP, metode beda hingga untuk PD linear dan tak linear		Menguasai metode numerik untuk PDP dan metode beda hingga untuk PD linear dan tak linear	
14	Metode beda hingga untuk PDP eliptik, parabolik dan hiperbolik		Menguasai metode beda hingga untuk PDP eliptik, parabolik dan hiperbolik	
15	Pengantar metode elemen hingga	Memahami pengantar metode elemen hingga		
16	UAS			

Figure 2: Satuan Acara Perkuliahan

1.3 Buku

Hanya pakai satu buku karena sudah lengkap. Tapi nanti Pak Kuntjoro akan kirim artikel dan jurnal sebagai tambahan lainnya.

1.4 Tugas dan Koding

- Akan banyak koding katanya.
- Tugas dan presentasi bisa saja nanti dilakukan.
- Mungkin ada kuis.
- Ujian jadi 3 kali, materi akan diberikan pas dekat-dekat ujian.
 - PR akan dijadikan bonus nilai.

CHAPTER I

2 MATERI PERKULIAHAN DETAIL

2.1 Materi Perkuliahan SK5001 Analisis Numerik

- Solusi Numerik Persamaan Tak Linear
 - Metode Bisection: metode iterasi titik tetap, metode Newton-Raphson
 - Metode Newton untuk Sistem Persamaan Tak Linear
- Metode Iteratif Untuk Sistem Persamaan Linear
 - Metode Jacobi
 - Gauss-Seidel
 - SOR Gauss-Seidel
- Pencocokan Kurva
 - Metode Kuadrat Terkecil, interpolasi dengan fungsi spline
- Optimisasi Numerik
- Metode Beda Hingga untuk PDP

3 AKAR PERSAMAAN NON LINEAR

3.1 Pendahuluan

Solution of Nonlinear Equations : $f(x) = 0$

Definisi: Akar dari persamaan adalah nilai saat fungsi bernilai nol.

Misalkan $f(x)$ adalah fungsi kontinu. Setiap nilai r yang memenuhi $f(r) = 0$ disebut **akar persamaan**.

3.1.1 *Intermediate Value Theorem* (Teorema Nilai Antara)

Misalkan $f \in C[a, b]$ dan L adalah suatu nilai di antara $f(a)$ dan $f(b)$. Maka **ada** suatu nilai $c \in (a, b)$ yang memenuhi $f(c) = L$.

Berbekal teorema tersebut, secara numerik kita bisa mencari akar persamaannya.

Misalkan fungsi f kontinu di $[a, b]$ dengan $f(a)$ dan $f(b)$ memiliki tanda berlawanan. Berdasarkan teorema nilai antara, maka ada suatu nilai c di $[a, b]$ yang memenuhi $f(c) = 0$. Apa kegunaan dari Teorema Nilai Antara?

3.2 Metode *Bisection*

Untuk mencari solusi $f(x) = 0$ dari fungsi kontinu f di interval $[a, b]$ di mana $f(a)$ dan $f(b)$ memiliki tanda berlawanan.

Metode ini hanya bisa untuk mencari **satu akar** dan mudah di-*plot* jika dimensinya hanya satu. Kalau dimensinya banyak, akan rumit.

Pilih $c = \frac{a+b}{2}$ merupakan titik tengah di $[a, b]$.

- Jika $f(a).f(c) < 0$, maka nol berada di $[a, c]$.
- Jika $f(c).f(b) < 0$, maka nol berada di $[c, b]$.
- Jika $f(c) = 0$, maka c adalah akar yang dicari.

3.2.1 *Bisection Theorem*

Misalkan $f \in C[a, b]$ dan $f(a).f(b) < 0$. Metode *bisection* akan men-generate baris $\{c_n\}_{n=0}^{\infty}$ untuk mendekati akar $r \in [a, b]$ dari fungsi f yang memenuhi:

$$|r - c_n| \leq \frac{b-a}{2^{n+1}}, \text{ untuk } n = 0, 1, 2, 3, \dots$$

Sehingga:

$$\lim_{n \rightarrow \infty} c_n = r$$

3.2.2 Algoritma *Bisection*

- **Step 1:** Find a and b with $a < b$ such that $f(a) \cdot f(b) < 0$.
- **Step 2:** Set $c = \frac{a+b}{2}$ and evaluate $f(c)$. If $f(c) = 0$ then $r = c$ and stop. Otherwise continue to Step 3.
- **Step 3:** If $f(a) \cdot f(c) < 0$ then reset $b = c$. Otherwise reset $a = c$.
- **Step 4:** If $|a - b| < \delta$ then stop. Use $\frac{a+b}{2}$ as the approximation to r . Otherwise return to Step 2.

Ilustrasi:

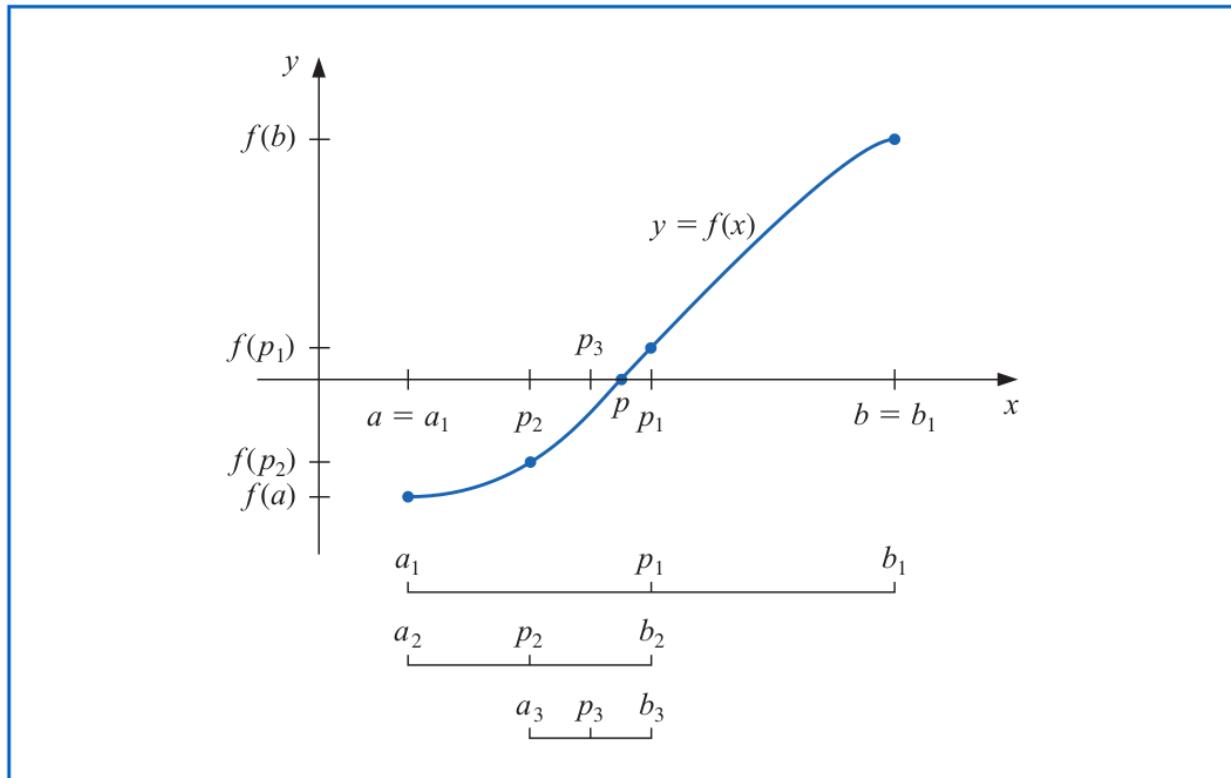


Figure 3: Ilustrasi Bisection

Untuk menemukan akar dari $f(x)$, kita perlu set terlebih dahulu nilai a dan b yang dibutuhkan.

3.2.3 Algoritma Formal *Bisection*

Berikut adalah algoritma formalnya:

To find a solution to $f(x) = 0$ given the continuous function f on the interval $[a, b]$, where $f(a)$ and $f(b)$ have opposite signs:

INPUT endpoints a, b ; tolerance TOL ; maximum number of iterations N_0 .

OUTPUT approximate solution p or message of failure.

Step 1 Set $i = 1$;

$$FA = f(a).$$

Step 2 While $i \leq N_0$ do Steps 3–6.

Step 3 Set $p = a + (b - a)/2$; (*Compute p_i .*)

$$FP = f(p).$$

Step 4 If $FP = 0$ or $(b - a)/2 < TOL$ then

OUTPUT (p); (*Procedure completed successfully.*)

STOP.

Step 5 Set $i = i + 1$.

Step 6 If $FA \cdot FP > 0$ then set $a = p$; (*Compute a_i, b_i .*)

$$FA = FP$$

else set $b = p$. (*FA is unchanged.*)

Step 7 OUTPUT ('Method failed after N_0 iterations, $N_0 =$ ', N_0);

(*The procedure was unsuccessful.*)

STOP.

■

Figure 4: Algoritma Bisection

3.2.4 Contoh Soal

Soal I

Cari nilai x di selang $[0, 2]$ yang memenuhi persamaan $x \cdot \sin(x) = 1$.

Masalah di atas sama dengan mencari nilai $x \in [0, 2]$ sehingga $x \cdot \sin(x) - 1 = 0$.

Sekarang kita akan gunakan metode *bisection*.

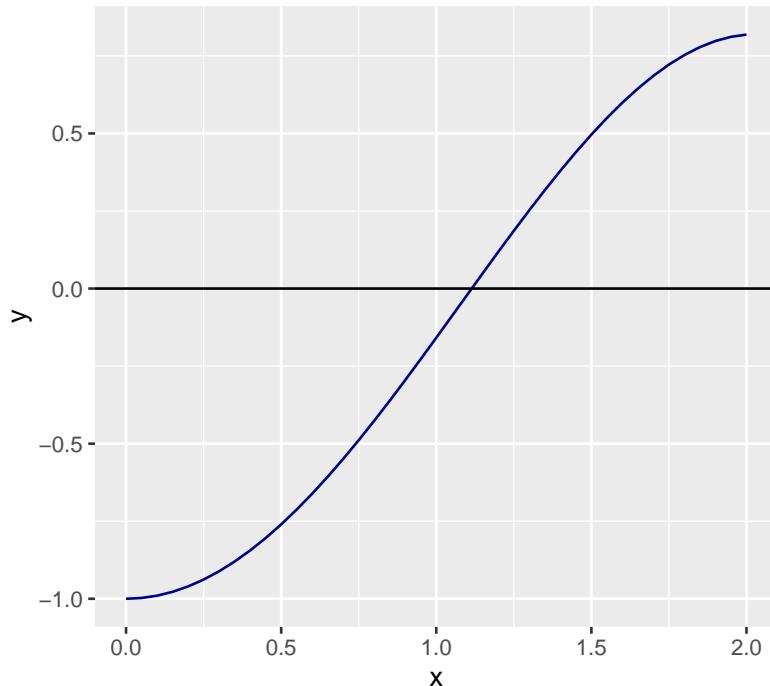


Figure 5: Grafik fungsi $f(x)$

Terlihat ada tepat satu titik $c \in [0, 2]$ di mana nilai $f(x) = 0$.

Sekarang kita lakukan algoritma di **R** nya:

n_iter	a	b	akar persamaan
1	0.000000	2.000000	1.000000
2	1.000000	2.000000	1.500000
3	1.000000	1.500000	1.250000
4	1.000000	1.250000	1.125000
5	1.000000	1.125000	1.062500
6	1.062500	1.125000	1.093750
7	1.093750	1.125000	1.109375
8	1.109375	1.125000	1.117188
9	1.109375	1.117188	1.113281

n_iter	a	b	akar persamaan
10	1.113281	1.117188	1.115234
11	1.113281	1.115234	1.114258
12	1.113281	1.114258	1.113770
13	1.113770	1.114258	1.114014
14	1.114014	1.114258	1.114136
15	1.114136	1.114258	1.114197
16	1.114136	1.114197	1.114166
17	1.114136	1.114166	1.114151
18	1.114151	1.114166	1.114159
19	1.114151	1.114159	1.114155
20	1.114155	1.114159	1.114157
21	1.114157	1.114159	1.114158
22	1.114157	1.114158	1.114157
23	1.114157	1.114157	1.114157
24	1.114157	1.114157	1.114157
25	1.114157	1.114157	1.114157

Kita dapatkan nilai $f(c) = 0$ pada $c = 1.1141571$ dengan toleransi $\delta = 0$ pada iterasi ke 25 kali

Soal II

Tunjukkan bahwa $f(x) = x^3 + 4x^2 - 10$ memiliki akar di selang $[1, 2]$! Gunakan metode *bisection* untuk menemukan akarnya!

Untuk menunjukkannya, cukup gunakan teorema nilai antara.

Perhatikan bahwa $f(1) = -5$ dan $f(2) = 14$. Karena f kontinu dan keduanya berbeda tanda, maka **ada** $c \in [1, 2]$ sehingga $f(c) = 0$.

Sekarang dengan algoritmanya:

n_iter	a	b	akar persamaan
1	1.000000	2.000000	1.500000
2	1.000000	1.500000	1.250000
3	1.250000	1.500000	1.375000
4	1.250000	1.375000	1.312500
5	1.312500	1.375000	1.343750
6	1.343750	1.375000	1.359375
7	1.359375	1.375000	1.367188
8	1.359375	1.367188	1.363281
9	1.363281	1.367188	1.365234
10	1.363281	1.365234	1.364258
11	1.364258	1.365234	1.364746
12	1.364746	1.365234	1.364990
13	1.364990	1.365234	1.365112

Kita dapatkan nilai $f(c) = -0.0019437$ pada $c = 1.3651123$ dengan toleransi $\delta = 0.0001$ pada iterasi ke 13 kali

Soal III

Gunakan metode *bisection* untuk mencari akar persamaan $x - 2^{-x} = 0, x \in [0, 1]$, dengan akurasi 10^{-5} !

Coba perhatikan grafik sebagai berikut:

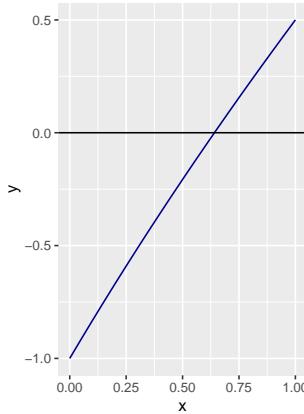


Figure 6: Grafik fungsi $f(x)$

Penyelesaiannya dengan *Bisection* adalah sebagai berikut:

n_iter	a	b	akar persamaan
1	0.0000000	1.0000000	0.5000000
2	0.5000000	1.0000000	0.7500000
3	0.5000000	0.7500000	0.6250000
4	0.6250000	0.7500000	0.6875000
5	0.6250000	0.6875000	0.6562500
6	0.6250000	0.6562500	0.6406250
7	0.6406250	0.6562500	0.6484375
8	0.6406250	0.6484375	0.6445312
9	0.6406250	0.6445312	0.6425781
10	0.6406250	0.6425781	0.6416016
11	0.6406250	0.6416016	0.6411133
12	0.6411133	0.6416016	0.6413574
13	0.6411133	0.6413574	0.6412354
14	0.6411133	0.6412354	0.6411743
15	0.6411743	0.6412354	0.6412048
16	0.6411743	0.6412048	0.6411896

Kita dapatkan nilai $f(c) = 0.0000055$ pada $c = 0.6411896$ dengan toleransi $\delta = 0.00001$ pada iterasi ke 16 kali.

3.2.5 R Function Metode Bisection

Berikut adalah *function* metode *bisection* yang saya buat di R:

```
bagi_dua = function(a,b,f,iter_max,tol_max){
  # fungsi hitung bisection
  # initial condition
  i = 1
  hasil = data.frame(n_iter = NA,
                      a = NA,
                      b = NA,
                      c = NA)

  while(i<= iter_max && (b-a) > tol_max){
    # cari titik tengahnya
    p = a + ((b-a)/2)
    # hitung fungsi di titik tengah
    FP = f(p)
    # hitung fungsi di titik awal
    FA = f(a)
    # hitung fungsi di titik akhir
    FB = f(b)
    # tulis hasil dalam data frame
    hasil[i,] = list(i,a,b,p)

    # tukar nilai a atau b dengan nilai p
    if(FA*FP < 0){b = p} else{a = p}
    # untuk iterasi berikutnya
    i = i + 1

    # mencatat akar persamaan
    akar = p
    # tambahan dulu checking apakah f(a), f(b), atau f(c) ada yang nol?
    if(FP == 0){
      akar = p
      break} else if(FA == 0){
        akar = a
        break} else if(FB == 0){
          akar = b
          break}

  }

  # mencatat iterasi terbesar
  iterasi = i-1 # dikurang satu karena pada i+1
}
```

```
# sebenarnya tidak ada proses jika pada while TRUE

# membuat ouput
hasil = list(
    `iterasi max` = iterasi,
    `akar persamaan` = akar,
    `hasil perhitungan` = hasil
)

# print output
return(hasil)
}
```

3.2.6 Tugas Kuliah Algoritma Minggu II

By using bisection method, find the root of the following functions:

1. $f(x) = x^2 - 3x - 2$
2. $f(x) = x^3 + x^2 - 3x - 2$

Jawab Pada metode *bisection*, pemilihan selang iterasi menjadi penting. Berdasarkan teorema nilai antara:

Misalkan $f \in C[a, b]$ dan L adalah suatu nilai di antara $f(a)$ dan $f(b)$. Maka **ada** suatu nilai $c \in (a, b)$ yang memenuhi $f(c) = L$.

Oleh karena itu, kita perlu menemukan selang awal iterasi $[a, b]$ yang memenuhi $f(a).f(b) < 0$. Iterasi akan dihentikan saat *error* memenuhi nilai yang diinginkan. Saya menggunakan kriteria pemberhentian iterasi sebagai berikut:

$$b_n - a_n < 10^{-5}, n = 1, 2, \dots, I$$

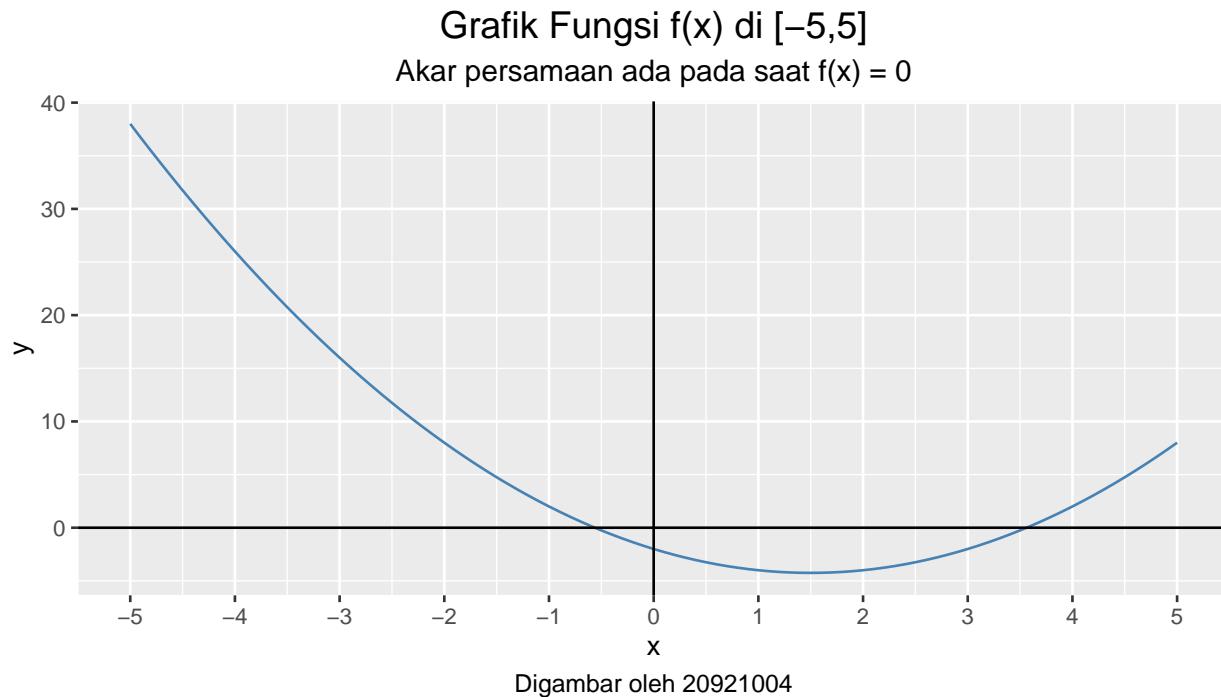
Persamaan I

Pertama-tama, mari kita buat grafik dari $f(x) = x^2 - 3x - 2$ sebagai berikut:

Dari grafik di atas, terlihat bahwa $f(x)$ memiliki dua akar sebagai berikut:

1. Satu akar $f(x)$ berada di selang $[-1, 0]$.
2. Satu akar $f(x)$ lainnya berada di selang $[3, 4]$.

Berdasarkan informasi di atas, kita akan lakukan metode *bisection* di kedua selang tersebut.

Figure 7: Grafik fungsi $f(x)$

Bisection pada selang $[-1, 0]$ Berikut adalah hasil proses *bisection* di selang tersebut:

```
bagi_dua(a = -1,
          b = 0,
          f,
          iter_max = 50,
          tol_max = 10^(-5))
```

```
## $`iterasi max`
## [1] 17
##
## $`akar persamaan`
## [1] -0.5615463
##
## $`hasil perhitungan`
##      n_iter          a          b          c
## 1      1 -1.0000000  0.0000000 -0.5000000
## 2      2 -1.0000000 -0.5000000 -0.7500000
## 3      3 -0.7500000 -0.5000000 -0.6250000
## 4      4 -0.6250000 -0.5000000 -0.5625000
## 5      5 -0.5625000 -0.5000000 -0.5312500
## 6      6 -0.5625000 -0.5312500 -0.5468750
## 7      7 -0.5625000 -0.5468750 -0.5546875
```

```
## 8      8 -0.5625000 -0.5546875 -0.5585938
## 9      9 -0.5625000 -0.5585938 -0.5605469
## 10     10 -0.5625000 -0.5605469 -0.5615234
## 11     11 -0.5625000 -0.5615234 -0.5620117
## 12     12 -0.5620117 -0.5615234 -0.5617676
## 13     13 -0.5617676 -0.5615234 -0.5616455
## 14     14 -0.5616455 -0.5615234 -0.5615845
## 15     15 -0.5615845 -0.5615234 -0.5615540
## 16     16 -0.5615540 -0.5615234 -0.5615387
## 17     17 -0.5615540 -0.5615387 -0.5615463
```

Bisection pada selang [3, 4] Berikut adalah hasil proses *bisection* di selang tersebut:

```
bagi_dua(a = 3,
          b = 4,
          f,
          iter_max = 50,
          tol_max = 10^(-5))

## $`iterasi max`
## [1] 17
##
## $`akar persamaan`
## [1] 3.561546
##
## $`hasil perhitungan`
##   n_iter      a      b      c
## 1 1 3.000000 4.000000 3.500000
## 2 2 3.500000 4.000000 3.750000
## 3 3 3.500000 3.750000 3.625000
## 4 4 3.500000 3.625000 3.562500
## 5 5 3.500000 3.562500 3.531250
## 6 6 3.531250 3.562500 3.546875
## 7 7 3.546875 3.562500 3.554688
## 8 8 3.554688 3.562500 3.558594
## 9 9 3.558594 3.562500 3.560547
## 10 10 3.560547 3.562500 3.561523
## 11 11 3.561523 3.562500 3.562012
## 12 12 3.561523 3.562012 3.561768
## 13 13 3.561523 3.561768 3.561646
## 14 14 3.561523 3.561646 3.561584
## 15 15 3.561523 3.561584 3.561554
## 16 16 3.561523 3.561554 3.561539
## 17 17 3.561539 3.561554 3.561546
```

Kesimpulan $f(x)$ memiliki akar pada $x = -0.5615463$ dan $x = 3.561546$

Persamaan II

Pertama-tama, mari kita buat grafik dari $f(x) = x^3 + x^2 - 3x - 2$ sebagai berikut:

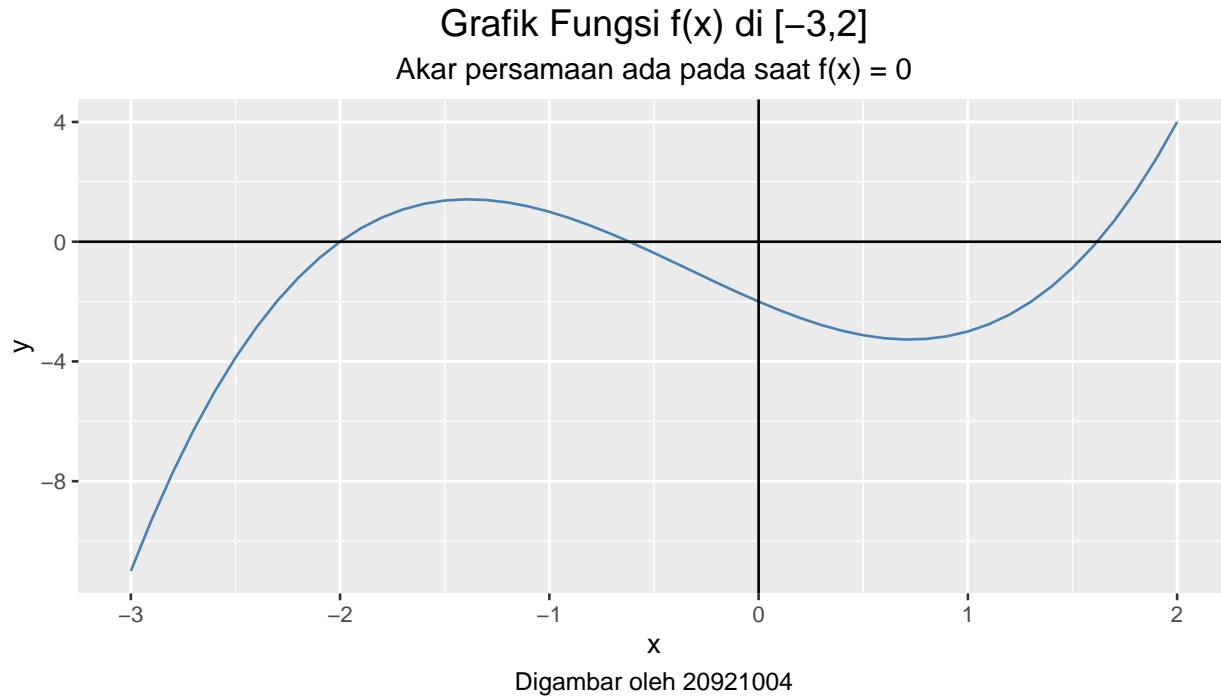


Figure 8: Grafik fungsi $f(x)$

Dari grafik di atas, terlihat ada tiga akar persamaan di selang:

1. $[-3, -1]$
2. $[-1, 0]$
3. $[1, 2]$

Berdasarkan informasi di atas, kita akan lakukan metode *bisection* di ketiga selang tersebut.

Bisection pada selang $[-3, -1]$ Berikut adalah hasil proses *bisection* di selang tersebut:

```
bagi_dua(a = -3,
          b = -1,
          f,
          iter_max = 50,
          tol_max = 10^(-5))
```

```
## $`iterasi max`
## [1] 1
##
## $`akar persamaan`
## [1] -2
##
## $`hasil perhitungan`
##   n_iter   a   b   c
## 1      1 -3 -1 -2
```

Bisection pada selang $[-1, 0]$ Berikut adalah hasil proses *bisection* di selang tersebut:

```
bagi_dua(a = -1,
          b = 0,
          f,
          iter_max = 50,
          tol_max = 10^(-5))

## $`iterasi max`
## [1] 17
##
## $`akar persamaan`
## [1] -0.6180344
##
## $`hasil perhitungan`
##   n_iter      a      b      c
## 1    1 -1.0000000 0.0000000 -0.5000000
## 2    2 -1.0000000 -0.5000000 -0.7500000
## 3    3 -0.7500000 -0.5000000 -0.6250000
## 4    4 -0.6250000 -0.5000000 -0.5625000
## 5    5 -0.6250000 -0.5625000 -0.5937500
## 6    6 -0.6250000 -0.5937500 -0.6093750
## 7    7 -0.6250000 -0.6093750 -0.6171875
## 8    8 -0.6250000 -0.6171875 -0.6210938
## 9    9 -0.6210938 -0.6171875 -0.6191406
## 10  10 -0.6191406 -0.6171875 -0.6181641
## 11  11 -0.6181641 -0.6171875 -0.6176758
## 12  12 -0.6181641 -0.6176758 -0.6179199
## 13  13 -0.6181641 -0.6179199 -0.6180420
## 14  14 -0.6180420 -0.6179199 -0.6179810
## 15  15 -0.6180420 -0.6179810 -0.6180115
## 16  16 -0.6180420 -0.6180115 -0.6180267
## 17  17 -0.6180420 -0.6180267 -0.6180344
```

Bisection pada selang [1, 2] Berikut adalah hasil proses *bisection* di selang tersebut:

```
bagi_dua(a = 1,
         b = 2,
         f,
         iter_max = 50,
         tol_max = 10^(-5))

## $`iterasi max`
## [1] 17
##
## $`akar persamaan`
## [1] 1.618034
##
## $`hasil perhitungan`
##   n_iter      a      b      c
## 1 1 1.000000 2.000000 1.500000
## 2 2 1.500000 2.000000 1.750000
## 3 3 1.500000 1.750000 1.625000
## 4 4 1.500000 1.625000 1.562500
## 5 5 1.562500 1.625000 1.593750
## 6 6 1.593750 1.625000 1.609375
## 7 7 1.609375 1.625000 1.617188
## 8 8 1.617188 1.625000 1.621094
## 9 9 1.617188 1.621094 1.619141
## 10 10 1.617188 1.619141 1.618164
## 11 11 1.617188 1.618164 1.617676
## 12 12 1.617676 1.618164 1.617920
## 13 13 1.617920 1.618164 1.618042
## 14 14 1.617920 1.618042 1.617981
## 15 15 1.617981 1.618042 1.618011
## 16 16 1.618011 1.618042 1.618027
## 17 17 1.618027 1.618042 1.618034
```

Kesimpulan $f(x)$ memiliki akar pada $x = -2$, $x = -0.6180344$, dan $x = 1.618034$.

3.3 Hal Penting Terkait Metode Bisection

3.3.1 Jaminan Keberadaan Akar Persamaan

Perhatikan baik-baik, jaminan bahwa $f(x)$ punya akar di $[a, b]$ adalah:

$f(a)$ dan $f(b)$ berbeda tanda!

Pada algoritma yang dibuat di bagian sebelumnya, perhatikan bahwa sebagai *input* syarat yang diperlukan adalah $a < b$.

Pada bagian ini, saya akan gunakan notasi p sebagai akar dari $f(x)$ dan penulisan deret hasil *bisection* yang digunakan untuk mengakproximasi akar sebenarnya. Menggantikan notasi c pada kuliah sebelumnya.

3.3.2 Kriteria STOP Iterasi

Pada bagian sebelumnya, kita menghentikan iterasi pada saat $\frac{a+b}{2} < \delta$.

Namun, pada bagian berikutnya kita akan menggunakan kriteria *error relatif* yakni:

$$\frac{|p_n - p_{n-1}|}{|p_n|} < \epsilon$$

Di sini mulai ada ϵ sebagai pengganti δ .

Beberapa kriteria lainnya penghentian iterasi antara lain:

$$|p_n - p_{n-1}| < \epsilon$$

dan

$$|f(p_n)| < \epsilon$$

Namun kita *error relatif* adalah yang terbaik karena bisa jadi:

1. $\{p_n\}_{n=0}^{\infty}$ barisnya divergen tapi $p_n - p_{n-1}$ konvergen ke **nol**.
2. $f(p_n)$ dekat ke nol tapi p_n jauh ke p .

3.4 Masalah Pada *Bisection* yang Ditemukan

1. Iterasi akan lebih lambat karena harus *bisect* selang yang ada.
2. Bisa jadi akar persamaan dalam suatu selang lebih dari satu.
 - Akibatnya kita perlu melihat grafik lalu memecah selangnya terlebih dahulu.

3.4.1 Teorema Penting¹

Misalkan $f \in C[a, b]$ dan $f(a) \cdot f(b) < 0$. Metode *bisection* akan menghasilkan deret $\{p_n\}_{n=1}^{\infty}$ yang mendekati p yang merupakan akar dari f dengan:

$$|p_n - p| \leq \frac{b - a}{2^n}, \text{ untuk } n \geq 1$$

Kegunaan dari teorema di atas adalah untuk menentukan berapa banyak iterasi yang dibutuhkan untuk mencapai suatu *error* tertentu.

3.4.2 Contoh Soal

Hitung berapa iterasi yang diperlukan untuk mencari akar persamaan $f(x) = x^3 + 4x^2 - 10 = 0$ dengan akurasi 10^{-3} menggunakan $a_1 = 1$ dan $b_1 = 2$!

Untuk menjawabnya, saya akan gunakan langsung teorema yang ada, yakni:

$$|p_n - p| \leq \frac{b - a}{2^n}$$

Kita menginginkan bahwa iterasi terakhir dari metode *bisection* menghasilkan akurasi sebesar 10^{-3} . Artinya pada $|p_N - p| < 10^{-3}$ sehingga:

$$|p_N - p| \leq 2^{-N}(b - a) = 2^{-N}(2 - 1) < 10^{-3}2^{-N} < 10^{-3}$$

Kita akan gunakan \log_{10} untuk memudahkan perhitungan.

$$\log_{10}2^{-N} < \log_{10}10^{-3} - N\log_{10}2 < -3N > \frac{3}{\log_{10}2} \simeq 9.965784$$

Maka didapatkan setidaknya harus ada 10 kali iterasi agar mencapai akurasi yang diinginkan.

¹Theorem 2.1 hal 48

3.5 Metode Newton

Merupakan metode yang cepat (konvergen ke dalam solusi) dan mudah diaplikasikan. Metode ini mengandalkan turunan atau gradien fungsi. Oleh karena itu **syarat** perlu agar metode ini bisa dipakai adalah:

$f(x)$ kontinu dan diferensiabel di selang $[x, y] \in R$

Ilustrasi:

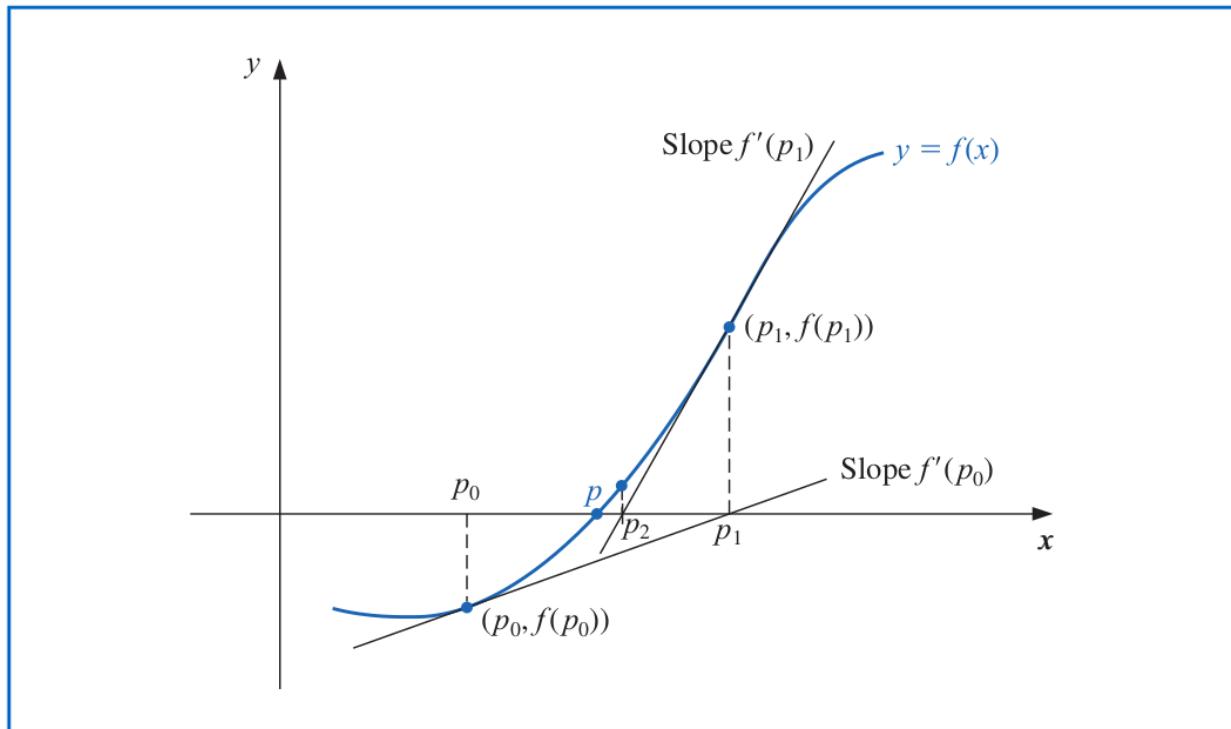


Figure 9: Ilustrasi Newton's Method

Prosesnya adalah menggunakan prinsip Deret Taylor.

Misalkan saya ingin menghitung suatu barisan x_0, x_1, x_2, \dots yang akan konvergen ke suatu solusi x^* , untuk suatu nilai δ yang kecil:

$$f(x_n + \delta) = f(x_n) + \delta f'(x_n) + O(\delta^2)$$

Dengan:

1. Mengabaikan $O(\delta^2)$ (karena si δ sudah kecil sehingga saat dikuadratkan juga akan lebih kecil lagi).
2. $f(x_n) = 0$ karena kita ingin mencari akar persamaan.

Maka kita bisa dapatkan:

$$f(x_n) + \delta f'(x_n) = 0$$

Sehingga menghasilkan:

$$\delta = -\frac{f(x_n)}{f'(x_n)}$$

Ingat kembali bahwa kita ingin x_n sangat dekat sekali dengan akar persamaan x^* .

Sehingga:

$$x_{n+1} - x_n = \delta x_{n+1} = x_n + \delta x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Ini menjadi dasar bagi algoritma pengerjaan metode *Newton*, yakni:

3.5.1 Algoritma Metode Newton

Algoritma berdasarkan skema iteratif berikut:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3.5.2 Algoritma Formal Metode Newton

Berikut adalah algoritma formal dari metode ini:

Newton's

To find a solution to $f(x) = 0$ given an initial approximation p_0 :

INPUT initial approximation p_0 ; tolerance TOL ; maximum number of iterations N_0 .

OUTPUT approximate solution p or message of failure.

Step 1 Set $i = 1$.

Step 2 While $i \leq N_0$ do Steps 3–6.

Step 3 Set $p = p_0 - f(p_0)/f'(p_0)$. (*Compute p_i .*)

Step 4 If $|p - p_0| < TOL$ then

OUTPUT (p); (*The procedure was successful.*)

STOP.

Step 5 Set $i = i + 1$.

Step 6 Set $p_0 = p$. (*Update p_0 .*)

Step 7 OUTPUT ('The method failed after N_0 iterations, $N_0 =$ ', N_0);

(*The procedure was unsuccessful.*)

STOP.

■

Figure 10: Algoritma Newton

3.5.3 Kriteria Penghentian Iterasi

Untuk menghentikan iterasi, kita pilih dulu *error* yang hendak ditoleransi, misalkan ϵ . Lalu saat kita membuat barisan p_1, p_2, \dots, p_N , berisan ini baru akan berhenti saat:

Kriteria I

$$|p_N - p_{N-1}| < \epsilon$$

Kriteria II

$$\frac{|p_N - p_{N-1}|}{p_N} < \epsilon$$

Kriteria III

$$|f(p_n)| < \epsilon$$

3.5.4 Contoh Soal

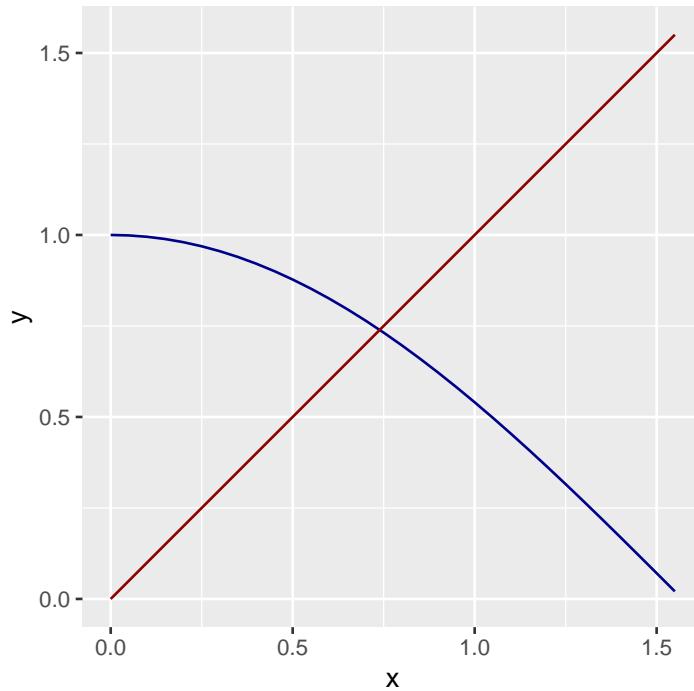
Soal I

Misalkan diketahui fungsi $f(x) = \cos(x) - x = 0$. Cari akar persamaannya!

Mari kita lihat dulu grafiknya:



Tapi coba kita *set back* dulu. Sebenarnya $f(x)$ bisa dipecah menjadi dua fungsi, misal $g(x)$ dan $h(x)$ sehingga akar dari $f(x)$ adalah perpotongan dua grafik tersebut.



Oke sekarang kita mulai iterasi dengan metode Newton.

Langkah pertama adalah mencari $f'(x)$. Saya akan menggunakan library(Ryacas) seperti yang pernah dijelaskan di sini².

```
library(Ryacas)
# persamaan awal
eq = "Cos(x) - x"
# turunan pertama
eq %>% y_fn("D(x)") %>% yac_str()

## [1] "-(Sin(x)+1)"
```

Didapatkan $f'(x) = -(sin(x) + 1)$.

Kita akan pilih $p_0 = \frac{\pi}{4}$. Berikut adalah iterasinya:

n_iter	p
0	0.7853982
1	0.7395361
2	0.7390852
3	0.7390851
4	0.7390851

Soal II

Misalkan $f(x) = x^2 - 6$ dan $p_0 = 1$, cari akar persamaannya dengan metode Newton!

Pertama-tama kita cari dulu $f'(x)$, yakni:

```
## [1] "2*x"
```

Sehingga berikut adalah hasilnya:

n_iter	p
0	1.000000
1	3.500000
2	2.607143
3	2.454256
4	2.449494

²Ryacas. <https://ikanx101.com/blog/math-r/>

n_iter	p
5	2.449490
6	2.449490

3.5.5 R Function Newton Method

Sekarang dari hasil di atas, saya akan buat *function* khusus *Newton Method* di **R**.

```
rm(list=ls())

newton_ikanx = function(p_0,f,df,iter_max,tol_max){
  # fungsi untuk grafik
  # set nilai x
  baris = seq(p_0-2*p_0,p_0+2*p_0,by = .05)
  # nilai y
  y = f(baris)
  # bikin plotnya
  plot =
    data.frame(x = baris,y) %>%
    ggplot(aes(x,y)) +
    geom_line(group = 1,
              color = "darkblue") +
    coord_equal() +
    geom_hline(yintercept = 0)

  # fungsi hitung newton
  # initial condition
  i = 1
  hasil = data.frame(n_iter = 0,
                      p = p_0)

  while(i <= iter_max){
    p = p_0 - (f(p_0) / df(p_0))
    hasil[i+1,] = list(i,p)
    if(abs(p-p_0) < tol_max){break}
    p_0 = p;
    i = i + 1
  }

  # mencatat iterasi terbesar
  iterasi = i # tidak perlu dikurang satu karena saya sudah pake BREAK

  # mencatat akar persamaan
}
```

```

akar = p

# membuat output
hasil = list(
  `plot f(x) di selang [p_0-p_0,p_0+p_0]` = plot,
  `iterasi max` = iterasi,
  `akar persamaan` = akar,
  `hasil perhitungan` = hasil
)

# print output
return(hasil)
}

```

Sekarang kita lihat contoh penggunaan *function* nya:

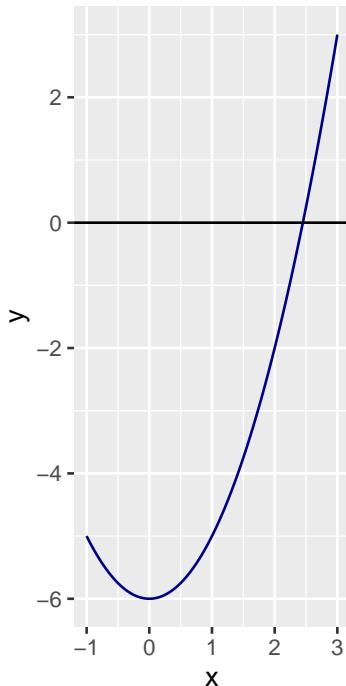
```

p_0 = 1
f = function(x){x^2 -6}
df = function(x){2*x}
iter_max = 50
tol_max = 10^-9

newton_ikanx(p_0,f,df,iter_max,tol_max)

```

```
## $`plot f(x) di selang [p_0-p_0,p_0+p_0]`
```



```
##  
## $`iterasi max`  
## [1] 6  
##  
## $`akar persamaan`  
## [1] 2.44949  
##  
## $`hasil perhitungan`  
##   n_iter      p  
## 1      0 1.000000  
## 2      1 3.500000  
## 3      2 2.607143  
## 4      3 2.454256  
## 5      4 2.449494  
## 6      5 2.449490  
## 7      6 2.449490
```

3.6 Refreshment Newton

Kita masih ingat bahwa konvergensi metode *bisection* itu linear (cenderung lambat) dan konvergensi metode *Newton* lebih cepat. Namun kita perlu tahu bahwa metode *Newton* memiliki syarat perlu berupa $f(x)$ yang diferensiable.

Bagaimana jika $f(x)$ tidak punya turunan?

Maka metode *Newton* tidak bisa digunakan.

Apakah ada metode yang *derivative-free* ?

3.7 Metode *Golden Section Search*

3.7.1 Definisi Formal

Misalkan $f(x)$ disebut *unimodal*³ pada selang $I = [a, b]$, jika terdapat sebuah titik $p \in I$ sehingga f monoton turun murni pada $[a, p]$ dan monoton naik murni pada $[p, b]$.

Dari definisi diatas dimungkinkan suatu cara untuk mengidentifikasi bahwa fungsi f memiliki **minimum global tunggal** di titik p , dengan tidak secara **eksplisit** melibatkan turunan fungsi.

³Fungsi yang hanya memiliki **satu puncak** saja. [https://id.wikipedia.org/wiki/Modus_\(statistika\)](https://id.wikipedia.org/wiki/Modus_(statistika))

Jadi walaupun di sini kita menggunakan istilah **kemonotonan** tapi tidak secara eksplisit disebutkan turunan fungsi.

Jika diperhatikan dengan baik, jadi si p adalah **titik terendah dari grafik fungsinya** sehingga masalah ini menjadi **masalah minimisasi**. Selanjutnya saat membahas **GSS**, kita akan mencari **titik terendah**.

Jadi ini adalah metode modifikasi. Bagaimana proses optimisasi digunakan untuk menghitung akar persamaan.

Jika dalam $[a, b]$ ada banyak titik minimum, kita akan buat sekat-sekat agar selang baru yang ada menjadi *unimodal!* Miriplah dengan si *bisection*. Jadi kita bisa buat agar jarak $[a, b]$ sekecil mungkin agar hanya ada satu **minimum global tunggal**.

3.7.2 Algoritma *Golden Section Search*

Metode **GSS** ini bisa digunakan untuk mengidentifikasi titik p . Pada metode ini selang $[a, b]$ yang memuat p digantikan dengan suatu subselangnya yang masih memuat titik p dengan menambahkan sepasang **titik dalam** c dan d yang memenuhi:

$$c = b - \frac{b-a}{r}d = a + \frac{b-a}{r}$$

r adalah bilangan *real* yang nilainya masih harus ditentukan melalui suatu cara tertentu.

Perhatikan bahwa $a < d < c < b$.

Kita perlu cek nilai $f(a), f(c), f(d), f(b)$.

Perhatikan bahwa karena f *unimodal* maka $f(c)$ dan $f(d)$ masing-masing bernilai lebih kecil dari $\max\{f(a), f(b)\}$. Perhatikan dua kondisi berikut ini:

1. Jika $f(c) \leq f(d)$ maka dari sifat *unimodal*, f monoton naik di selang $[d, b]$ dan dengan demikian maka $p \in [a, d]$.
2. Jika $f(c) > f(d)$ maka dari sifat *unimodal*, f monoton turun di selang $[a, c]$ dan dengan demikian maka $p \in [c, b]$.

Situasi di atas memungkinkan kita untuk **mengurangi lebar selang pencarian**⁴ untuk titik p . Misalkan:

1. $a < c < d < b$ dengan $c = a + r(b-a)$ dan $d = a + (1-r)(b-a)$ maka berikut adalah algoritma perhitungannya:
2. Saya tuliskan $f(a) = fa, f(b) = fb, f(c) = fc, f(d) = fd$.

⁴mengecilkan lebar selang pencarian

```

if f(c) <= f(d)
  b = d
  fb = fd
else
  a = c
  fa = fc
end
c = a + r * (b-a)
fc = f(c)
d = a + (1-r)*(b-a)
fd = f(d)

```

Setelah melakukan langkah di atas maka lebar selang pencarian menjadi berkurang dengan faktor sebesar $(1 - r)$.

Pemilihan nilai r menjadi faktor krusial agar konvergensi menjadi semakin cepat!

3.7.3 Penentuan Nilai r

r yang paling optimal adalah *golden ratio*.

$$r = \frac{1 + \sqrt{5}}{2} \simeq 0.618$$

3.7.4 Mencari Akar Persamaan $f(x)$

Terdapat hubungan antara masalah optimisasi ini dengan akar persamaan. Persamaan $f(x) = 0$ memiliki solusi x jika fungsi objektif F dari masalah optimisasi yang didefinisikan sebagai berikut:

- $F(x) = (f(x))^2$ memiliki nilai minimum global sebesar 0.
- $F(x) = |f(x)|$ memiliki nilai minimum global sebesar 0.
- $F(x) = -\frac{1}{a+|f(x)|}$ memiliki nilai minimum global sebesar -1 .

3.7.5 *Function* di R

Berdasarkan algoritma di atas dan hubungannya dengan **masalah minimisasi**, maka berikut adalah *function* di R-nya:

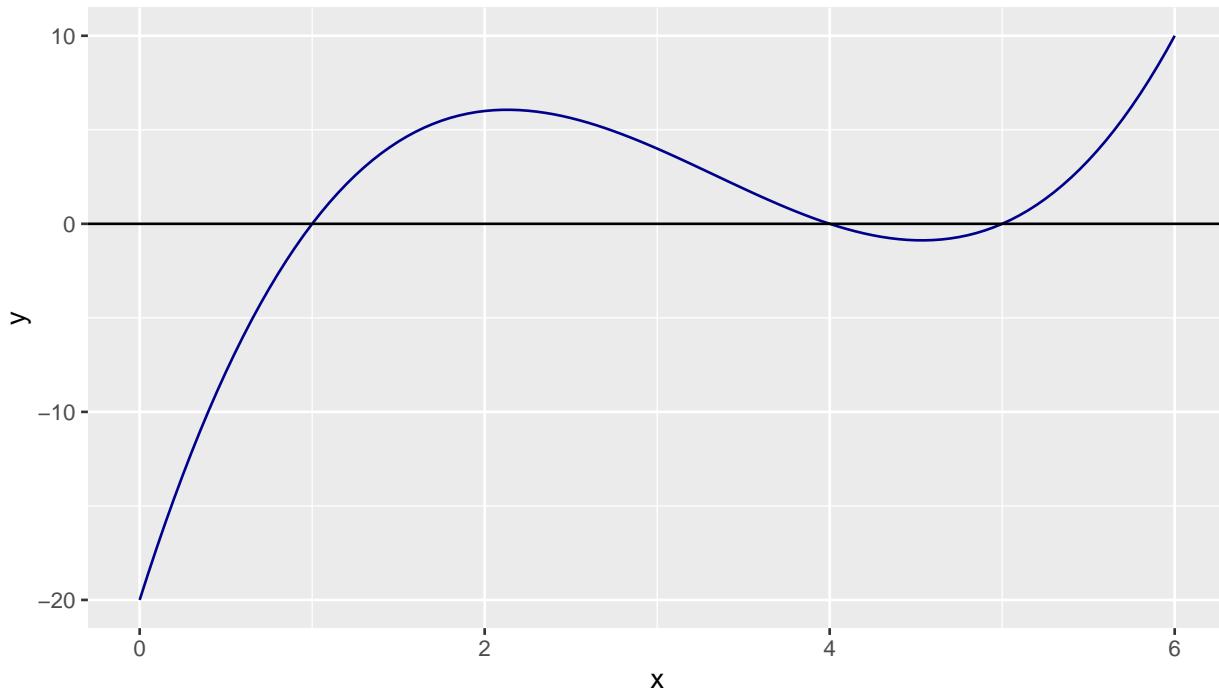
```
golden_ss = function(a,b,f){  
    # definisikan r sebagai golden ratio  
    r = (1 + sqrt(5))/2  
    # toleransi kesalahan yang bisa diterima  
    tol_max = 10^(-10)  
  
    while(abs(b - a) > tol_max){  
        c = b - (b - a) / r  
        d = a + (b - a) / r  
  
        if(f(c) < f(d)){  
            b = d  
        } else{  
            a = c  
        }  
    }  
  
    hasil = ( a + b ) / 2  
    return(hasil)  
}
```

3.7.6 Contoh Soal

Soal I

Untuk fungsi $f(x) = x^3 - 10x^2 + 29x - 20$ di $0 \leq x \leq 6$, tentukan semua akarnya dengan metode *Golden Section Search* !

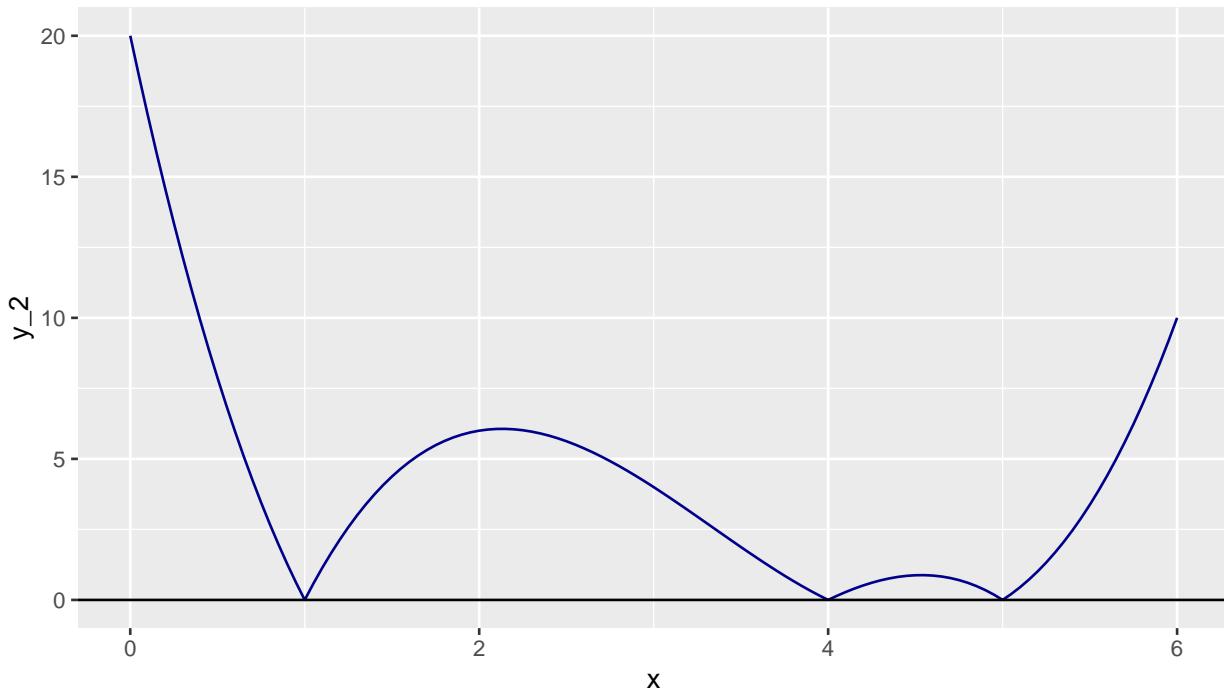
Jawab Mari kita buat dulu grafik fungsinya di selang tersebut:



Kalau kita perhatikan, akar persamaan tersebut ada di selang $[0, 2]$, $[3.5, 4.5]$, $[4.5, 6]$.

Perlu saya ingatkan kembali, tujuan dari **GSS** adalah mencari titik minimum global di selang tertentu.

Oleh karena itu, jika kita ingin mencari akar, maka kita harus ubah terlebih dahulu fungsinya menjadi $f'(x) = |f(x)|$. Berikut adalah grafiknya:



Sekarang kita bisa mencari akarnya di selang $[0, 2]$, yakni:

```
# fungsi dari soal
f_awal = function(x){x^3 - 10*x^2 + 29*x -20}
# fungsi modifikasi
f = function(x){abs(x^3 - 10*x^2 + 29*x -20)}

# initial
a = 0
b = 2

hasil = golden_ss(a,b,f)
```

Akar pada selang tersebut adalah di $x = 1$.

Kita akan coba pada selang $[3.5, 4.5]$ sebagai berikut:

```
golden_ss(3.5,4.5,f)
```

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

Selanjutnya pada selang $[4.5, 6]$ sebagai berikut:

```
golden_ss(4.5,6,f)
```

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

CHAPTER II

4 NUMERICAL INTEGRATION

Selain menggunakan pendekatan analitik, kita bisa menggunakan pendekatan numerik untuk mencari nilai hampiran untuk suatu **integral tentu** satu peubah.

Integral tentu memiliki bentuk seperti ini:

$$\int_a^b f(x)dx = F(b) - F(a)$$

Integral tentu juga bisa dituliskan dalam bentuk **penjumlahan Riemann** berikut:

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

Dengan cara memperbanyak partisi n dan memperkecil selang Δx kita bisa menghampiri nilai luas di bawah kurva.

4.1 Hampiran *Square* dan *Trapezoid*

Kita bisa menggunakan hampiran berupa *square* atau *trapezoid* untuk menghitung luas di bawah kurva.

Ide dasarnya adalah $L = \text{alas.tinggi}$, dimana:

1. *alas* bisa didefinisikan sebagai $\Delta x = x_2 - x_1$.
2. Sedangkan definisi *tinggi* tergantung bidang yang dipilih.
 - Jika kita menggunakan *square* di titik tengah Δx , maka *tinggi* = $f\left(\frac{x_1+x_2}{2}\right)$.
 - Jika kita menggunakan *trapezoid*, maka *tinggi* = $\frac{f(x_1)+f(x_2)}{2}$.

4.1.1 R Function Hampiran *Square* dan *Trapezoid*

Berikut adalah algoritmanyanya:

```
hampiri = function(x0,xn,n,f){
  # save initial dulu
  initial_1 = x0
  initial_2 = xn

  # =====
  # metode square
```

```
h = (xn - x0) / n
integration = f(x0)
for(i in 1:n){
  k = x0 + i*h
  integration = integration + f(k)
}
square = integration * h

# =====
# metode trapezoid
x0 = initial_1
xn = initial_2
h = (xn - x0) / n
integration = f(x0) + f(xn)
for(i in 1:n){
  k = x0 + i*h
  integration = integration + 2*f(k)
}
trapezoid = integration * h/2

# =====
# bikin output
selang = n
luas_persegi = square
luas_trapezoid = trapezoid

output = list(selang,luas_persegi,luas_trapezoid)
}
```

4.1.2 Contoh Soal

Hitung luas area di bawah kurva berikut:

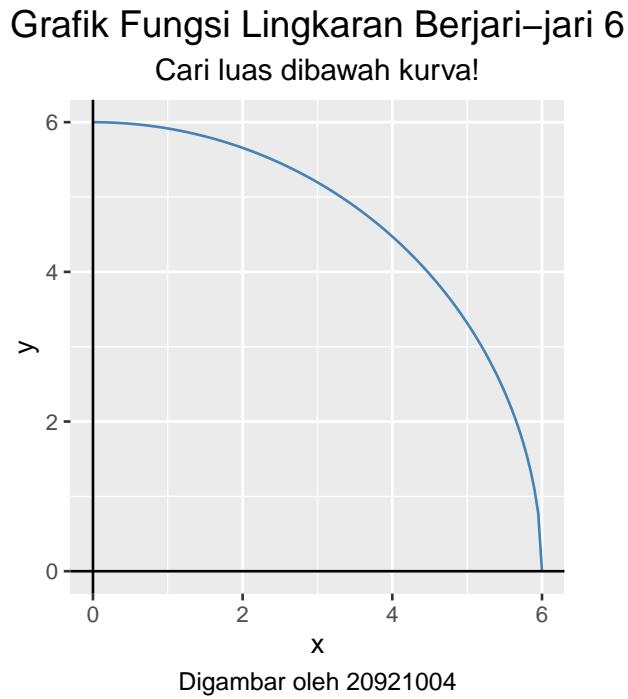


Figure 11: Seperempat Lingkaran

Perhatikan bahwa kita bisa membentuk kurva tersebut dengan fungsi $f(x) = \sqrt{6^2 - x^2}$, untuk $x \in [0, 6]$.

Untuk menghitung luas $f(x)$ di $[0, 6]$ secara *exact*, setidaknya kita bisa melakukan dua hal, yakni:

1. Menghitung $\int_0^6 f(x)dx$.
2. Menghitung dari rumus luas lingkaran: $\frac{1}{4}\pi.r^2 = \frac{1}{4}\pi.6^2 = 28.2743339$.

Untuk menghampirinya secara numerik, kita akan hitung penjumlahan luas dari *square* atau *trapezoid* yang dibangun di bawah kurva.

Sekarang kita akan hitung luas $f(x)$ untuk berbagai macam nilai n kemudian akan kita bandingkan hasilnya dengan hitungan *exact* sebelumnya.

Saya definisikan:

$$\Delta = \text{exact} - \text{hampiran}$$

Table 6: Hasil Perhitungan Hampiran vs Exact Lingkaran
I

n	luas_persegi	luas_trapezoid	delta_persegi_exact	delta_trapezoid_exact
5	30.93344	27.33344	-2.6591056	0.9408944
10	29.74066	27.94066	-1.4663311	0.3336689
25	28.90977	28.18977	-0.6354329	0.0845671
50	28.60442	28.24442	-0.3300827	0.0299173
100	28.44375	28.26375	-0.1694194	0.0105806
200	28.36059	28.27059	-0.0862586	0.0037414
500	28.30939	28.27339	-0.0350534	0.0009466
1000	28.29200	28.27400	-0.0176653	0.0003347
3000	28.28027	28.27427	-0.0059356	0.0000644
5000	28.27790	28.27430	-0.0035701	0.0000299
10000	28.27612	28.27432	-0.0017894	0.0000106
20000	28.27523	28.27433	-0.0008963	0.0000037
50000	28.27469	28.27433	-0.0003591	0.0000009

Silakan baca tulisan saya terkait hampiran nilai π dengan cara ini di sini⁵.

4.2 Brute Force

Salah satu metode lain untuk menghitung integral (luas daerah di bawah kurva) adalah dengan melakukan *brute force* dengan analogi pelemparan *darts*. Luas area di bawah kurva dihitung dengan cara:

$$L = \text{alas} \times \text{tinggi} \times \frac{\text{darts}_{\text{on target}}}{\text{darts}_{\text{All}}}$$

⁵<https://ikanx101.com/blog/pi-trapezoid/>

4.2.1 Flowchart Brute Force

Berikut adalah flowchartnya:

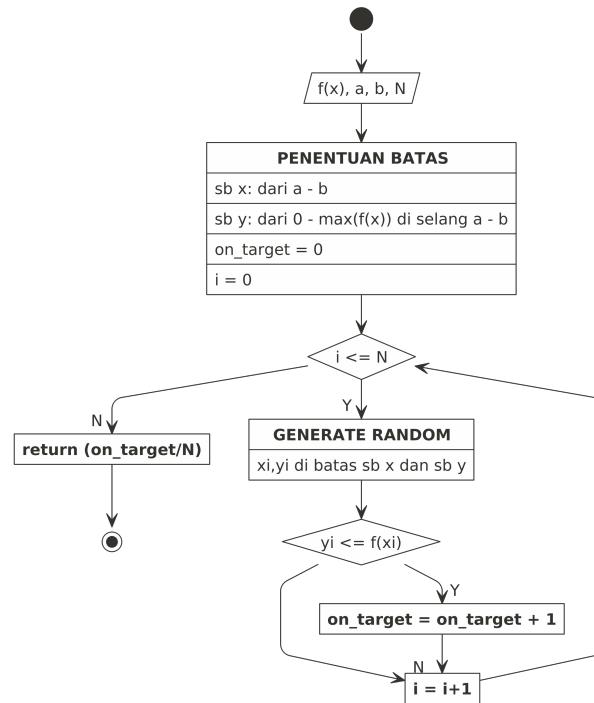


Figure 12: Flowchart Brute Force

4.2.2 R Function Brute Force

Berikut adalah algoritmanya dalam **R**:

```
set.seed(2021)

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

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

  # output
  output = list(
    "Plot Brute Force" = plot_sim,
    "Luas area di bawah kurva" = luas
  )

  return(output)
}
```

4.2.3 Contoh Soal

Buatlah algoritma sederhana dengan metode Monte Carlo untuk mencari solusi dari integral berikut:

$$f(x) = \int_1^5 x^2 dx$$

Bandingkan dengan solusi analitiknya!

Jawab Berikut adalah langkah kerja yang dilakukan untuk menjawab soal ini:

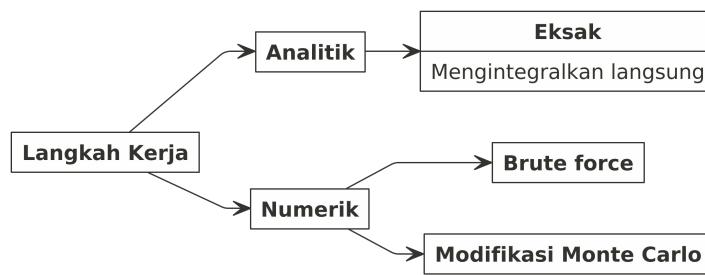


Figure 13: Alur Kerja

Kelak akan kita bandingkan metode numerik dengan hasil eksaknya.

Analitik Perhatikan bahwa pada integral tentu berlaku:

$$\int_a^b f(x)dx = F(b) - F(a)$$

Oleh karena itu, jika kita memiliki $f(x) = x^2$, maka $F(x) = \int f(x)dx = \frac{x^3}{3}$ dari soal:

$$\int_1^5 x^2 dx = \frac{5^3}{3} - \frac{1^3}{3} \approx 41.33333$$

Brute Force Analogi dari metode numerik ini adalah seperti melempar *darts*. Luas area di bawah kurva bisa didefinisikan sebagai:

$$L = \frac{Ndarts_{\text{on target}}}{Ndarts_{\text{All}}}$$

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 grafik $f(x)$ berikut:



Figure 14: Grafik $f(x)$

Untuk sumbu x , batas titik yang akan di-random sudah jelas, yakni: $[1, 5]$.

Lantas bagaimana dengan sumbu y ?

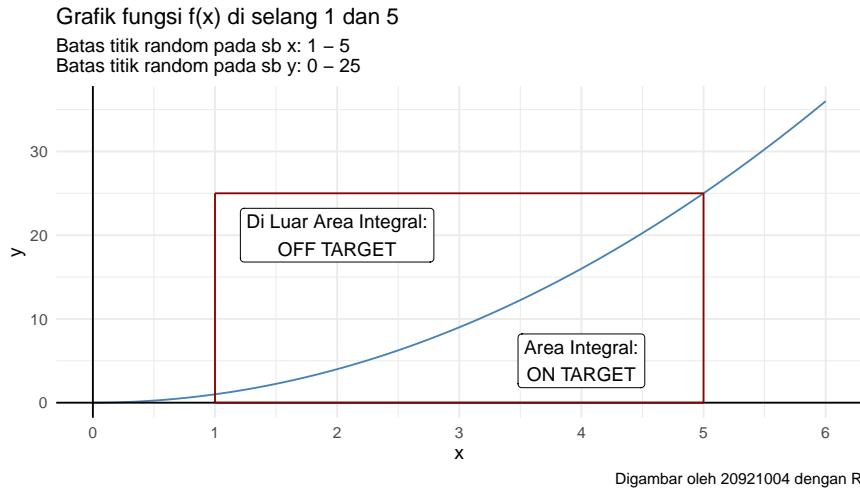


Figure 15: Penentuan Batas Titik Random

Kita akan membuat sejumlah *random* di dalam area kotak merah dari grafik di atas. Kelak luas akan dihitung dari rasio titik di dalam area **on target** dengan **total semua titik yang ada** dikalikan dengan luas dari kotak merah.

$$L = 4 \times 25 \times \frac{Ndarts_{\text{on target}}}{Ndarts_{\text{All}}}$$

Dari *function* di atas, kita akan coba hitung dengan berbagai nilai N sebagai berikut:

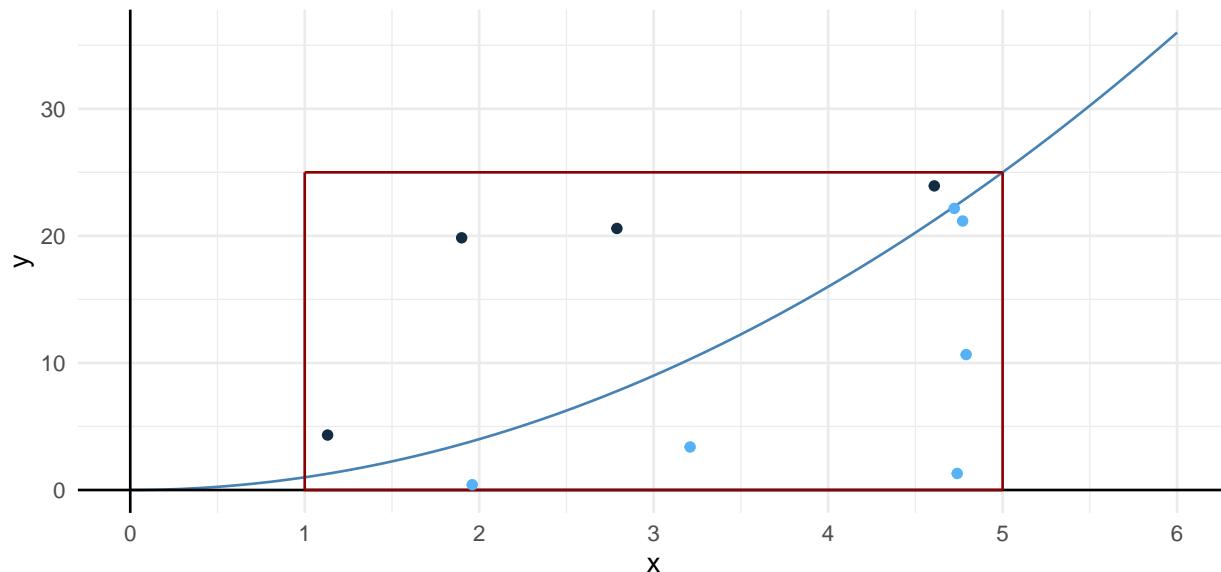
$N = 10$

```
brute_force(f, x1 = 1, x2 = 5, y1 = 0, y2 = 25, N = 10)
```

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

Hasil Simulasi dengan 10 titik

Didapat nilai rasio sebesar 0.6



Digambar oleh 20921004 dengan R

```
##
## $`Luas area di bawah kurva`
## [1] 60
```

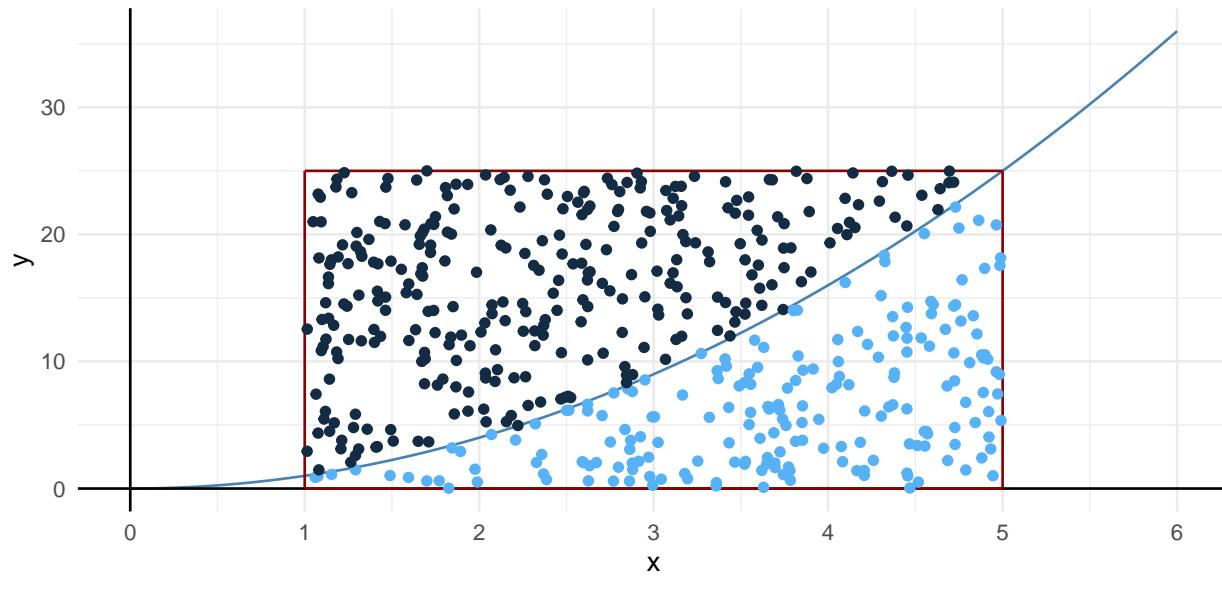
$N = 500$

```
brute_force(f,x1 = 1,x2 = 5,y1 = 0,y2 = 25,N = 500)
```

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

Hasil Simulasi dengan 500 titik

Didapat nilai rasio sebesar 0.396



Digambar oleh 20921004 dengan R

```
##  
## $`Luas area di bawah kurva`  
## [1] 39.6
```

4.3 Modifikasi Monte Carlo

Ide dari algoritma ini adalah men-*generate* titik *random* di selang integral, kemudian dihitung luas *square* yang ada.

$$I = \int_z^b f(x)dx$$

dihitung sebagai:

$$\langle F^N \rangle = \frac{b-a}{N+1} \sum_{i=0}^N f(a + (b-a)\xi_i)$$

dengan

ξ_i adalah random number antara 0 dan 1

4.3.1 Flowchart Modifikasi Monte Carlo

Berikut adalah *flowchart*-nya:

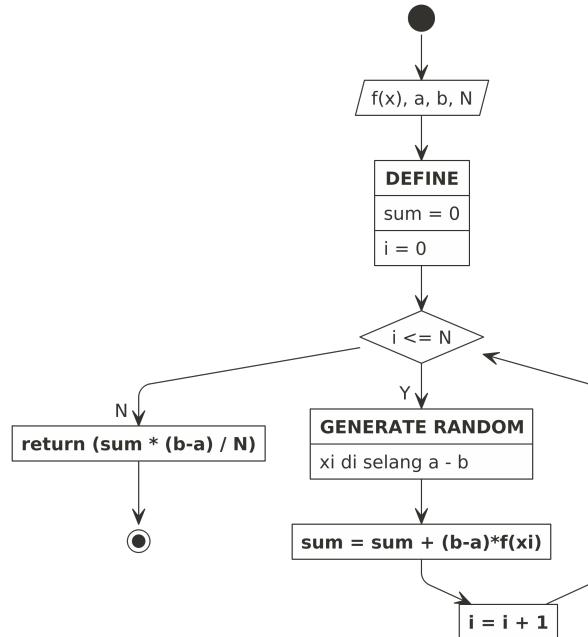


Figure 16: Flowchart Modifikasi Monte Carlo

4.3.2 R *Function* Modifikasi Monte Carlo

Berdasarkan *flowchart* di atas, berikut adalah *function* di **R**-nya:

```
modif_monte = function(f,x1,x2,N){
  # generating random number
  x = runif(N,x1,x2)
  # hitung f(x)
  f_x = f(x)
  # hitung luas
  luas = (x2-x1) * f_x
  # mean luas
  mean_luas = mean(luas)
  # output
  return(mean_luas)
}
```

4.3.3 Contoh Soal

Dari soal pada bagian sebelumnya, saya akan hitung hasilnya sebagai berikut:

Table 7: Hasil Perbandingan Solusi Numerik dan Eksak

N	Selang 1-5	Delta 1-5
104800	41.32739	0.0059408
155300	41.37991	0.0465770
169500	41.39308	0.0597437
203300	41.32101	0.0123214
219200	41.41033	0.0769936
245600	41.28355	0.0497833
253700	41.35787	0.0245387
279400	41.29370	0.0396374
313000	41.33638	0.0030435
313900	41.31451	0.0188270
334200	41.39295	0.0596160
402400	41.37190	0.0385678
426600	41.30155	0.0317806
456600	41.39593	0.0626000
473700	41.31768	0.0156518
479000	41.30772	0.0256102
479800	41.40445	0.0711124
529500	41.32646	0.0068741
608300	41.33363	0.0002936

N	Selang 1-5	Delta 1-5
639700	41.32034	0.0129906
713600	41.32239	0.0109445
723000	41.32154	0.0117930
726000	41.37137	0.0380375
742300	41.37724	0.0439067
757500	41.31624	0.0170941
867500	41.34685	0.0135121
892400	41.27271	0.0606269
898400	41.30221	0.0311214
912000	41.35568	0.0223450
988400	41.33593	0.0026014

CHAPTER III

5 METODE ITERATIF UNTUK SPL

Suatu sistem persamaan linear bisa dituliskan dalam bentuk aljabar berupa matriks dan vektor sebagai berikut:

$$Ax = b$$

SPL tersebut akan memiliki solusi saat A memiliki invers, yakni A^{-1} .

Bagaimana caranya menghitung A^{-1} dari A ?

Pada aljabar linear elementer, kita bisa melakukan **operasi baris elementer (OBE)** sebagai berikut:

$$[A|I] \rightarrow [I|A^{-1}]$$

5.1 Refreshment Aljabar Linear

5.1.1 Teorema SPL

Jika A adalah matriks $n \times n$ *inversible*, maka untuk setiap vektor b $1 \times n$ pada sistem persamaan linear $Ax = b$ memiliki tepat satu solusi. Yaitu:

$$x = A^{-1}b$$

Berikut ini kita coba *review* kembali secara singkat beberapa konsep dalam aljabar yang kelak akan membantu kita memahami bagaimana cara penyelesaian **SPL** secara numerik.

5.1.2 Matriks Singular dan Tak Singular

Dari penjelasan sebelumnya kita telah mengenal apa itu matriks invers. Kita akan mengingat kembali suatu konsep bernama determinan matriks. Di **R** kita bisa menghitung determinan dari suatu matriks dengan perintah `det(matriks)`.

Jika suatu matriks persegi memiliki determinan = 0, maka matriks tersebut tidak memiliki invers. Matriks yang tidak memiliki invers disebut **matriks singular**.

Kebalikannya:

Jika suatu matriks memiliki determinan $\neq 0$, maka matriks memiliki invers. Matriks berinvers disebut **matriks tak singular**.

5.1.3 Determinan Matriks

Untuk matriks berukuran 2×2 , cara menghitung determinan matriksnya adalah sebagai berikut:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$|A| = ad - bc$$

Bagaimana dengan matriks berukuran 3×3 ? Berikut caranya:

Kita perlu memperlebar kolom dari matriks tersebut agar semua elemen terkena.

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$|A| = (aei + bfg + cdh) - (ceg + afh + bdi)$$

5.1.4 Sifat-Sifat Determinan Matriks

Misalkan A adalah matriks yang memiliki determinan, maka:

1. $|A^T| = |A|$
2. $|A \cdot B| = |A| \cdot |B|$
3. $|A^n| = |A|^n$
4. $|A^{-1}| = \frac{1}{|A|}$
5. $|k \times A_{m \times m}| = k^m \times |A|$

5.1.5 Nilai Eigen dan Vektor Eigen

Definisi Jika A adalah matriks $n \times n$, maka vektor tak nol $x \in \mathbb{R}^n$ dinamakan **vektor eigen** dari A jika Ax adalah kelipatan skalar dari x . Yakni:

$$Ax = \lambda x$$

untuk suatu skalar λ tertentu. λ disebut sebagai **nilai eigen** yang bersesuaian dengan **vektor eigen**.

Teorema Jika A adalah matriks berukuran $n \times n$, maka λ adalah nilai eigen dari A jika dan hanya jika ia memenuhi persamaan:

$$\det(\lambda I - A) = 0$$

Persamaan di atas disebut dengan **persamaan karakteristik**.

Vektor dan nilai eigen bisa dihitung di **R** menggunakan *function eigen(matriks)*.

5.1.6 Norm Vektor

Norm vektor merupakan fungsi pemetaan dari vektor-vektor di $x \in \mathbb{R}^n$ ke bilangan *real* $\|x\|$ sehingga memenuhi:

- $\|\mathbf{x}\| > 0 \quad \forall \mathbf{x} \neq \mathbf{0}$, dan $\|\mathbf{x}\| = 0 \leftrightarrow \mathbf{x} = \mathbf{0}$;
 - $\|\alpha \mathbf{x}\| = |\alpha| \|\mathbf{x}\| \quad \forall \alpha \in \mathbb{R}$;
 - $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$;
- contoh-contoh :
- $\|\mathbf{x}\|_1 \equiv \sum_{i=1}^n |x_i|$ norm l_1 ;
 - $\|\mathbf{x}\|_2 \equiv \sqrt{\sum_{i=1}^n |x_i|^2}$ norm l_2 ;
 - $\|\mathbf{x}\|_\infty \equiv \max_{1 \leq i \leq n} |x_i|$ norm l_∞ ;

Figure 17: Sifat Norm Vektor

5.1.7 Norm Matriks

Norm matriks merupakan fungsi pemetaan dari matriks-matriks bujur sangkar di $\mathbb{R}^{n \times n}$ ke \mathbb{R} sehingga memenuhi:

- $\|\mathbf{A}\| \geq 0 \forall \mathbf{A} \neq \mathbf{0}$, dan $\|\mathbf{A}\| = 0 \Leftrightarrow \mathbf{A} = \mathbf{0}$
- $\|\alpha\mathbf{A}\| = |\alpha|\|\mathbf{A}\| \forall \alpha \in \mathbb{R}$
- $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\| \forall \mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$
- $\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\| \forall \mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times n}$

Figure 18: Sifat Norm Matriks

$$\forall z \neq 0 \text{ dan natural matrix norm } \|\cdot\| \text{ diperoleh } \|Az\| \leq \|A\|\|z\|$$

Cara menghitung norm matriks Beberapa teorema dan definisi yang berguna:

Teorema $\|A\|_{\infty} = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|$ dan $\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}|$.

Definisi Jari-jari spektral dari matriks \mathbf{A} , notasi $\rho(\mathbf{A})$, didefinisikan sebagai

$$\rho(\mathbf{A}) = \max \{|\lambda_k| : \lambda_k \text{ suatu nilai eigen matriks } \mathbf{A}\}$$

Perhatikan bahwa jika λ sebarang nilai eigen dari matriks \mathbf{A} yang berpadanan dengan vektor eigen \mathbf{v} dengan $\|\mathbf{v}\|=1$, maka $|\lambda| = |\lambda| \cdot \|\mathbf{v}\| = \|\lambda\mathbf{v}\| = \|\mathbf{Av}\| = \|\mathbf{A}\|\|\mathbf{v}\| = \|\mathbf{A}\|$ sehingga $\rho(\mathbf{A}) \leq \|\mathbf{A}\|$.

Teorema

- (i) $\|\mathbf{A}\|_2 = \sqrt{\rho(\mathbf{A}^T \mathbf{A})}$
- (ii) $\rho(\mathbf{A}) \leq \|\mathbf{A}\| \forall \text{ natural norm } \|\cdot\|$

Figure 19: Norm Matriks dan Jari-jari Spektral

Summaries:

1. Norm ∞ adalah nilai max sum harga mutlak baris.
2. Norm 1 adalah nilai max sum harga mutlak kolom.

Teorema Kekonvergenan Suatu matriks A dikatakan **konvergen** ke $\mathbf{0}$ jika $\lim_{k \rightarrow \infty} (A^k)_{ij} = 0$.

Pernyataan berikut ini ekivalen:

1. A matriks yang **konvergen**.
2. $\lim_{k \rightarrow \infty} \|A^k\| = 0$, untuk suatu *natural norm*.
3. $\lim_{k \rightarrow \infty} \|A^k\| = 0$, untuk setiap *natural norm*.
4. $\rho(A) < 1$.
5. $\lim_{k \rightarrow \infty} A^k x = 0, \forall x$.

Perhatikan baik-baik teorema di atas. Terutama pada poin 4 dimana menjadi **kunci** kekonvergenan dari suatu skema iterasi kelak.

5.1.8 Matriks Diagonal, *Upper*, dan *Lower*

Matriks diagonal adalah matriks yang hanya berisi elemen di diagonalnya saja. Matriks *upper triangle* adalah matriks yang hanya berisi segitiga di atas. Matriks *lower triangle* adalah matriks yang hanya berisi segitiga di bawah.

5.1.9 Matriks Diagonal Dominan Kuat

Matriks **diagonal dominan kuat** adalah matriks yang memiliki elemen harga mutlak diagonal terbesar.

Misalkan:

```
##      [,1] [,2] [,3]
## [1,] "a11" "a12" "a13"
## [2,] "a21" "a22" "a23"
## [3,] "a31" "a32" "a33"
```

Definisi **diagonal dominan kuat** adalah:

$$|a_{ii}| > \sum_{i=1, j \neq i}^n |a_{ij}|$$

Contoh:

```
##      [,1] [,2] [,3]
## [1,]    7    2    0
## [2,]    3    5   -1
## [3,]    0    5   -6
```

A merupakan matriks **diagonal dominan kuat** karena:

$$|7| > |2| + |0|$$

$$|5| > |3| + |-1|$$

$$|-6| > |0| + |-5|$$

Matriks **diagonal dominan kuat** adalah *non singular*.

Bagaimana jika A kita bukan matriks **diagonal dominan kuat**?

Kita harus lakukan *pre-processing* sehingga menjadi matriks **diagonal dominan kuat** dengan cara:

1. Tukar baris, atau
2. Tukar kolom.

5.1.10 Aljabar di R

Dari semua konsep aljabar yang telah dijelaskan pada bagian sebelumnya, mari kita lihat beberapa function di **R**-nya:

Misalkan saya definisikan v suatu vektor sebagai berikut:

```
v = c(2,4,-1,3)
# menghitung norm 1 dari vektor v
norm_vec_1 = function(x)sum(abs(x))
norm_vec_1(v)

## [1] 10

# menghitung norm 2 dari vektor v
norm_vec_2 = function(x)sqrt(sum(abs(x)^2))
norm_vec_2(v)

## [1] 5.477226

# menghitung norm infinity dari vektor v
norm_vec_inf = function(x)max(abs(x))
norm_vec_inf(v)

## [1] 4
```

Misalkan saya definisikan matriks A sebagai berikut:

```

A = matrix(c(2,4,-1,3,1,5,-2,3,-1), ncol = 3)
A

##      [,1] [,2] [,3]
## [1,]    2    3   -2
## [2,]    4    1    3
## [3,]   -1    5   -1

# transpose matriks
t(A)

##      [,1] [,2] [,3]
## [1,]    2    4   -1
## [2,]    3    1    5
## [3,]   -2    3   -1

# inverse matriks
matlib::inv(A)

##                  [,1]          [,2]          [,3]
## [1,]  0.22535211  0.09859155 -0.1549296
## [2,] -0.01408451  0.05633803  0.1971831
## [3,] -0.29577465  0.18309859  0.1408451

# menghitung nilai dan vektor eigen
eigen(A)

## eigen() decomposition
## $values
## [1] -5.607665  5.148415  2.459250
##
## $vectors
##      [,1]      [,2]      [,3]
## [1,]  0.4119543  0.3651285 -0.4204609
## [2,] -0.5715723  0.7504594  0.4578471
## [3,]  0.7096469  0.5509010  0.7833190

# perkalian matriks
A %*% matlib::inv(A) %>% round()

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

```

```
# menghitung norm 1 dari matriks A  
norm(A, "1")
```

```
## [1] 9
```

```
# menghitung norm 2 dari matriks A  
norm(A, "2")
```

```
## [1] 6.161479
```

```
# menghitung norm infinity dari matriks A  
norm(A, "i")
```

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

```
# menghitung rho  
rho = function(matriks){  
  ei = eigen(matriks)  
  ei = max(abs(ei$values))  
  return(ei)  
}  
rho(A)
```

```
## [1] 5.607665
```

```
# norm 2 juga bisa dihitung dengan cara  
rho(t(A) %*% A) %>% sqrt()
```

```
## [1] 6.161479
```

5.1.11 Key Take Points Materi Aljabar Linear

Perhatikan betul teorema kekonvergenan matriks dan bagaimana cara menghitung ρ .

5.2 Visualisasi Aljabar Linear

Materi ini berasal dari *paper* Prof. Kuntjoro berjudul ***Linear Transformation on the Points of the Unit Circle in the Plane and Its Visualization.***

Kita mulai dari lingkaran satuan, misal didefinisikan:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

adalah matriks bilangan *real* yang *non singular*.

Jika kita hendak menggambar lingkaran satuan, maka:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} \cos t \\ \sin t \end{bmatrix}, 0 \leq t \leq 2\pi$$

dengan $A = I$.

Berikut adalah gambarnya:

```
rm(list=ls())
satuan =
  data.frame(t = seq(0,2*pi,by = 0.01)) %>%
    mutate(x = cos(t),
          y = sin(t))

satuan %>%
  ggplot(aes(x,y)) +
  geom_point() +
  coord_equal()
```

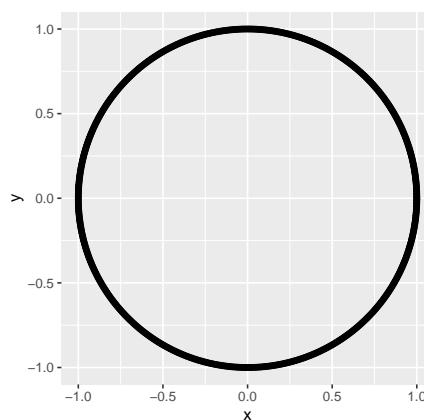


Figure 20: Menggambar Lingkaran Satuan

Kita bisa mentransformasi lingkaran satuan di atas dengan mengubah matriks A . Kita akan coba satu-persatu:

5.2.1 Contoh Matriks A_1

Contoh pertama adalah sebagai berikut:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} \cos t \\ \sin t \end{bmatrix}, 0 \leq t \leq 2\pi$$

```
# definisi matriks
A = matrix(c(2,0,2,2), ncol = 2, byrow = T)

# buat rumah untuk hasil transformasi linear
satuan =
  satuan %>%
    mutate(x_new = NA,
          y_new = NA)

# looping untuk menghitung transformasi linear
for(i in 1:nrow(satuan)){
  temp = A %*% c(satuan$x[i],
                 satuan$y[i])
  satuan$x_new[i] = temp[1]
  satuan$y_new[i] = temp[2]
}

# buat grafik baru
satuan %>%
  ggplot() +
  geom_point(aes(x,y), color = "blue", size = .1) +
  geom_point(aes(x_new,y_new), color = "red", size = .1) +
  coord_equal()
```

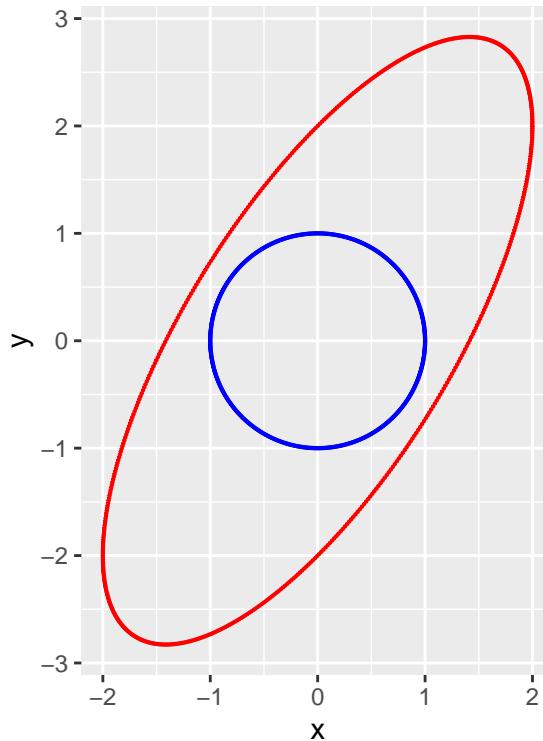


Figure 21: Transformasi Matriks A1

Lantas apa kegunaan visualisasi ini?

Ingatkah apa definisi dari vektor dan nilai eigen?

Dari sini kita bisa mempelajari di mana letak vektor eigen dan berapa nilai eigennya. Kita bisa cek pada t berapa nilai (x', y') merupakan kelipatan dari (x, y) .

Akan saya perlihatkan di t berapa:

```
eigen =
satuan %>%
  mutate(eigen_x = x_new / x,
        eigen_y = y_new / y,
        sama = round(eigen_x,2) == round(eigen_y,2)) %>%
  filter(sama == T)

# buat grafik baru
satuan %>%
  ggplot() +
  geom_point(aes(x,y),color = "blue",size = .1) +
  geom_point(aes(x_new,y_new),color = "red",size = .1) +
  # nilai eigen pertama
```

```
geom_segment(x = 0, xend = eigen$x_new[1] ,
              y = 0, yend = eigen$y_new[1] ,
              color = "brown") +
geom_segment(x = 0, xend = eigen$x[1] ,
              y = 0, yend = eigen$y[1] ,
              color = "darkgreen") +
# nilai eigen kedua
geom_segment(x = 0, xend = eigen$x_new[2] ,
              y = 0, yend = eigen$y_new[2] ,
              color = "brown") +
geom_segment(x = 0, xend = eigen$x[2] ,
              y = 0, yend = eigen$y[2] ,
              color = "darkgreen") +
coord_equal()
```

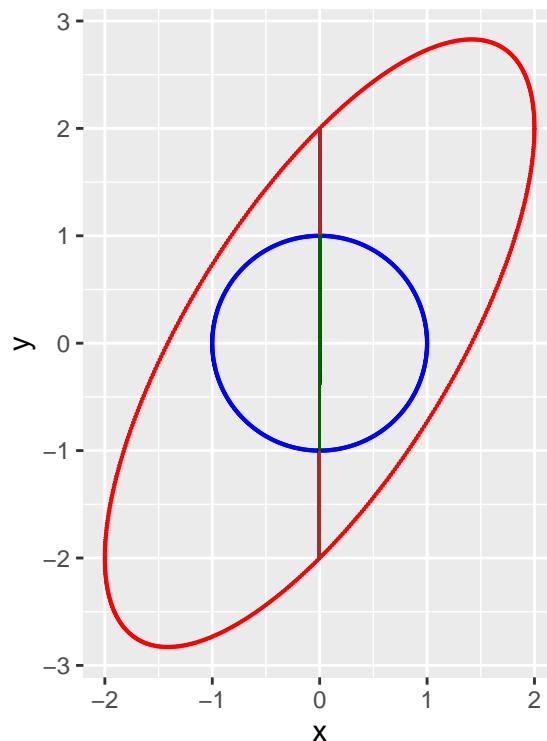


Figure 22: Letak Vektor Eigen Matriks A1

5.2.2 Contoh Matriks A_2

Contoh berikut adalah sebagai berikut:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \cos t \\ \sin t \end{bmatrix}, 0 \leq t \leq 2\pi$$

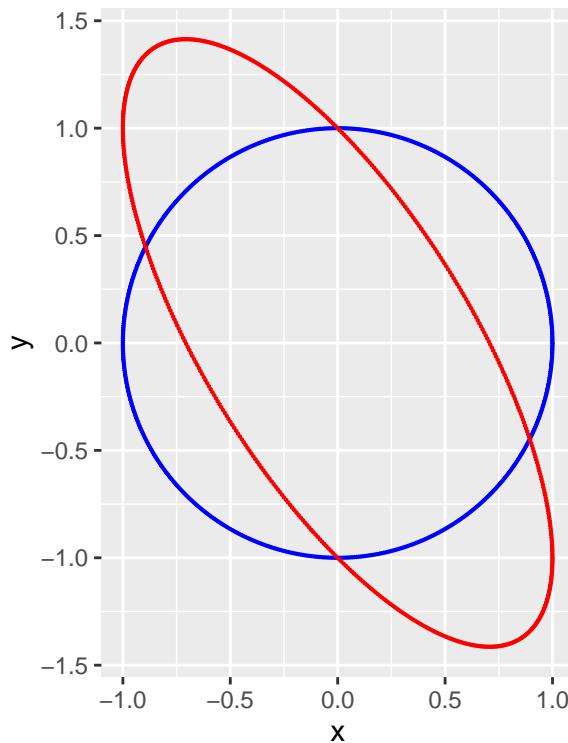


Figure 23: Transformasi Matriks A2

Kita akan lihat, ada di t berapa nilai eigennya:

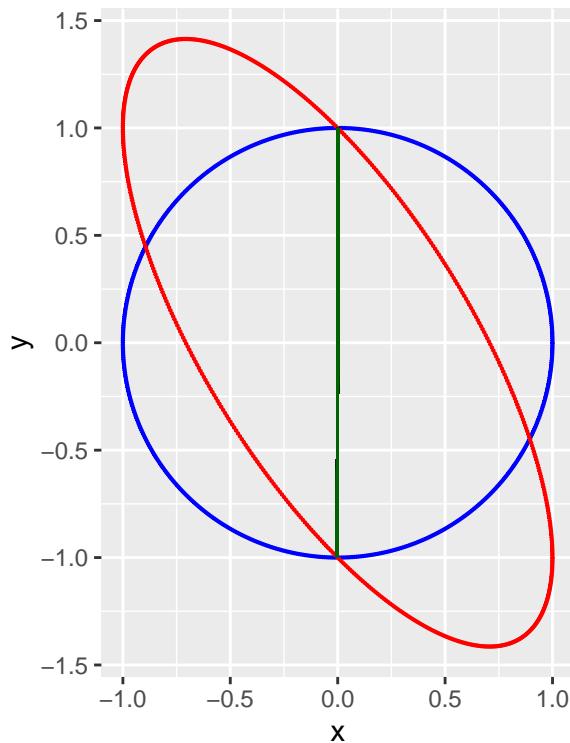


Figure 24: Letak Vektor Eigen Matriks A2

Obviously terlihat ya.

5.2.3 Contoh Matriks A_3

Contoh berikut adalah sebagai berikut:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \cos t \\ \sin t \end{bmatrix}, 0 \leq t \leq 2\pi$$

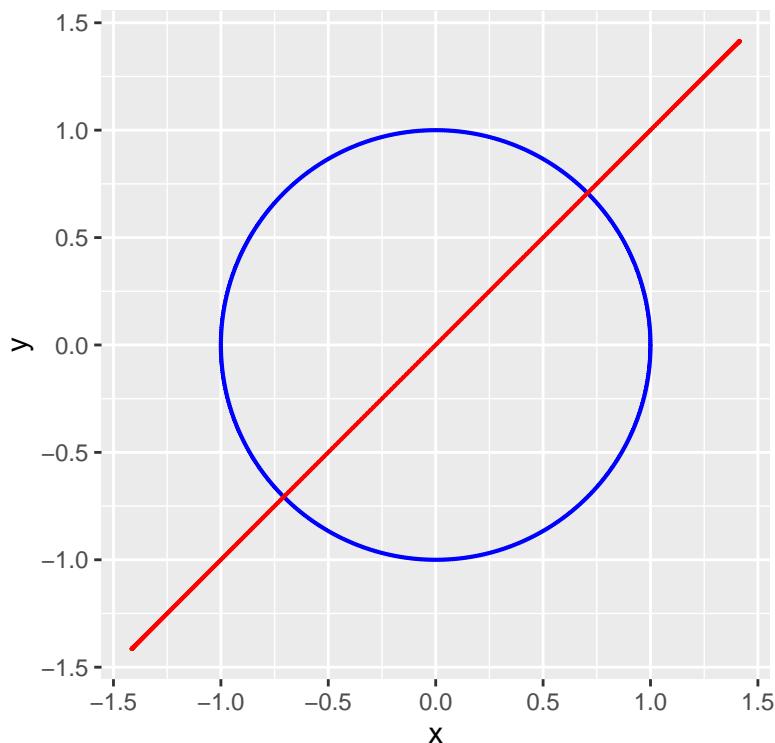


Figure 25: Transformasi Matriks A2

Sudah bisa menebak di mana letak t tempat vektor eigen berada?

5.2.4 Contoh Matriks dengan Eigen Bilangan Kompleks

Bagaimana jika kita tidak bisa menemukan letak t secara grafik? Bisa jadi karena A memiliki nilai eigen berupa bilangan kompleks.

5.3 Pengantar Iterasi SPL

5.3.1 Penulisan dalam Bentuk Matriks

Misalkan di suatu sistem persamaan linear berikut:

$$Ax = b$$

di \mathbb{R}^n dapat dituliskan sebagai:

$$Mx = (M - A)x + b$$

untuk suatu matriks M yang sesuai dan **tidak singular** (punya invers), diperoleh:

$$x = (I - M^{-1}A)x + M^{-1}b$$

yang memberikan skema iterasi:

$$x^{k+1} = (I - M^{-1}A)x^k + M^{-1}b$$

Perhatikan baik-baik skema iterasi ini. Karena skema ini kelak akan menjadi **kunci saat membuat algoritma iterasi**.

Komponen $(I - M^{-1}A)$ disebut sebagai B .

Kekonvergenan iterasi akan diperoleh **jika dan hanya jika** $\rho(B) < 1$. Ini merupakan **syarat perlu**.

Oleh karena kita mengetahui bahwa $\rho(B) \leq \|B\|$, maka **syarat cukup** kekonvergenan iterasi yaitu $\|B\| < 1$.

Pada bagian selanjutnya, kita bisa menuliskan $A = L + D + U$, dengan: L lower, U upper, dan D matriks diagonal.

5.3.2 Kriteria Penghentian Iterasi

Kita bisa menggunakan kriteria penghentian iterasi sebagai berikut:

$$\frac{\|x^{(k)} - x^{(k-1)}\|_\infty}{\|x^{(k)}\|_\infty} < \epsilon$$

Perhatikan bahwa **norm yang dipakai adalah norm vector!**

5.4 Iterasi Jacobi

5.4.1 Pendahuluan

Misalkan saya memiliki SPL seperti ini:

$$4x - y + z = 7$$

$$4x - 8y + z = -21$$

$$-2x + y + 5z = 15$$

Cara pengerjaannya adalah membuat formula solusi untuk masing-masing x, y, z sebagai berikut:

$$x = \frac{7 + y - z}{4}$$

$$y = \frac{21 + 4x + z}{8}$$

$$z = \frac{15 + 2x - y}{5}$$

Dari sini kita akan buat iterasi Jacobi sebagai berikut:

$$x^{k+1} = \frac{7 + y^k - z^k}{4}$$

$$y^{k+1} = \frac{21 + 4x^k + z^k}{8}$$

$$z^{k+1} = \frac{15 + 2x^k - y^k}{5}$$

Dengan *initial value* x, y, z sembarang, kita harap akan menghasilkan iterasi yang konvergen.

$$x_0 = 1, y_0 = 2, z_0 = 2$$

i	x	y	z
0	10.000000	20.000000	20.000000
1	1.750000	10.125000	3.000000

i	x	y	z
2	3.531250	3.875000	1.675000
3	2.300000	4.600000	3.637500
4	1.990625	4.229687	3.000000
5	2.057422	3.995312	2.950312
6	2.011250	4.022500	3.023906
7	1.999648	4.008613	3.000000
8	2.002153	3.999824	2.998137
9	2.000422	4.000844	3.000897
10	1.999987	4.000323	3.000000
11	2.000081	3.999993	2.999930
12	2.000016	4.000032	3.000034
13	1.999999	4.000012	3.000000
14	2.000003	4.000000	2.999997
15	2.000001	4.000001	3.000001
16	2.000000	4.000000	3.000000
17	2.000000	4.000000	3.000000
18	2.000000	4.000000	3.000000
19	2.000000	4.000000	3.000000
20	2.000000	4.000000	3.000000

Agar konvergen menuju solusi, perhatikan bahwa A harus **diagonal dominan kuat**

5.4.2 Matriks Iterasi B

Pada metode Jacobi:

$$B_j = -D^{-1}C = -D^{-1}(L + U)$$

Jadi, untuk mengecek apakah A akan konvergen atau tidak, kita akan mengecek $\rho(B) < 1$ sebagai **syarat perlu**. Sedangkan **syarat cukup** kekonvergenan adalah A harus dominan diagonal kuat.

5.4.3 Skema Interasi dalam Bentuk Matriks

Pilih $M = D$ sehingga $M - A = -L - U$ yang memberikan skema iterasi:

$$Dx^{k+1} = -(L + U)x^k + b$$

$$\rightarrow x^{k+1} = -D^{-1}(L + U)x^k + D^{-1}b$$

5.4.4 R *Function* Jacobi

Berikut adalah *function* iterasi Jacobi:

```
jacobi_ikanx = function(A,b,x_awal,iter_max,tol_max){
  # A adalah matriks
  # b adalah vector
  # x_awal adalah vektor utk initial iteration
  # tol_max toleransi error
  # iter_max iterasi maksimum

  # ambil diagonalnya
  da = diag(A)
  # bikin matriks diagonal
  D = diag(da)
  # bikin matriks C, yakni A yang sudah dihapus diagonalnya
  C = A - D
  # cari D inverse
  Dinv = matlib:::inv(D)
  # cari Jacobian matriks B
  B = - Dinv %*% C
  # hitung b1
  b1 = Dinv %*% b
  # iterasi
  for(i in 1:iter_max){
    x = B %*% x_awal + b1
    if(norm(x-x_awal,"2") < tol_max * norm(x_awal,"2")) {break}
    x_awal = x
  }
  # output
  output = list("iterasi yang dilakukan: " = i,
                "Solusi: " = x_awal)
  return(output)
}
```

Perhatikan baik-baik bahwa A harus dominan diagonal! Sekarang kita coba untuk kasus berikut:

```
A = matrix(c(3,1,1,0,1,5,0,1,1,-1,3,1,0,2,1,4),ncol = 4)
b = c(1,4,-2,1)
iter_max = 100
tol_max = 10^(-5)
x_awal = c(1,1,1,1)

jacobi_ikanx(A,b,x_awal,iter_max,tol_max)
```

```
## $`iterasi yang dilakukan: `  
## [1] 25  
##  
## $`Solusi: `  
## [,1]  
## [1,] 0.5555403  
## [2,] 0.3240906  
## [3,] -0.9907263  
## [4,] 0.4166552
```

5.5 Metode Gauss Seidel

5.5.1 Matriks Iterasi B

Pada metode Gauss Seidel:

$$B_{GS} = -(D + L)^{-1}U$$

Jadi, untuk mengecek apakah A akan konvergen atau tidak, kita akan mengecek $\rho(B) < 1$ sebagai **syarat perlu**. Sedangkan **syarat cukup** kekonvergenan adalah A harus dominan diagonal kuat.

5.5.2 Skema Interasi dalam Bentuk Matriks

Pilih $M = D + L$ sehingga $M - A = -U$ yang memberikan skema iterasi:

$$(D + L)x^{k+1} = -Ux^k + b$$

$$\rightarrow x^{k+1} = -(D + L)^{-1}Ux^k + (D + L)^{-1}b$$

5.5.3 R Function Gauss Seidel

```
jacobi_seigel_ikanx = function(A,b,x_awal,iter_max,tol_max){
  # A adalah matriks
  # b adalah vector
  # x_awal adalah vektor utk initial iteration
  # tol_max toleransi error
  # iter_max iterasi maksimum

  # ambil diagonalnya
  da = diag(A)
  # bikin matriks diagonal
  D = diag(da)
  # bikin matriks C, yakni A yang sudah dihapus diagonalnya
  C = A - D
  # lower diagonal matrix
  U = C
  U[lower.tri(U)] = 0
  # lower diagonal matrix
  L = C
  L[upper.tri(L)] = 0
```

```

# cari D inverse
Dinv = matlib:::inv(D + L)
# cari Jacobian matriks B
B = - Dinv %*% U
# hitung b1
b1 = Dinv %*% b
# bikin vector iterasi
vector_iter = vector("list")
vector_iter[[1]] = x_awal
# iterasi
for(i in 1:iter_max){
  x = B %*% x_awal + b1
  if(norm(x-x_awal,"2") < tol_max * norm(x_awal,"2")) {break}
  x_awal = x
  vector_iter[[i+1]] = x_awal
}
# output
output = list("iterasi yang dilakukan: " = i,
              "Solusi akhir: " = x_awal,
              "Iterasi ke- " = vector_iter)
return(output)
}

```

Berikut adalah contoh penggunaannya:

```

A = matrix(c(3,1,1,0,1,5,0,1,1,-1,3,1,0,2,1,4),ncol = 4)
b = c(1,4,-2,1)
iter_max = 100
tol_max = 10^(-5)
x_awal = c(1,1,1,1)

jacobi_seigel_ikanx(A,b,x_awal,iter_max,tol_max)

```

```

## $`iterasi yang dilakukan: `
## [1] 15
##
## $`Solusi akhir: `
##      [,1]
## [1,]  0.5555466
## [2,]  0.3240809
## [3,] -0.9907355
## [4,]  0.4166636
##
## $`Iterasi ke- `
```

```
## $`Iterasi ke- `[[1]]
## [1] 1 1 1 1
##
## $`Iterasi ke- `[[2]]
## [,1]
## [1,] -0.3333333
## [2,] 0.6666667
## [3,] -0.8888889
## [4,] 0.3055556
##
## $`Iterasi ke- `[[3]]
## [,1]
## [1,] 0.4074074
## [2,] 0.4185185
## [3,] -0.9043210
## [4,] 0.3714506
##
## $`Iterasi ke- `[[4]]
## [,1]
## [1,] 0.4952675
## [2,] 0.3715021
## [3,] -0.9555727
## [4,] 0.3960176
##
## $`Iterasi ke- `[[5]]
## [,1]
## [1,] 0.5280235
## [2,] 0.3448737
## [3,] -0.9746804
## [4,] 0.4074517
##
## $`Iterasi ke- `[[6]]
## [,1]
## [1,] 0.5432689
## [2,] 0.3334295
## [3,] -0.9835735
## [4,] 0.4125360
##
## $`Iterasi ke- `[[7]]
## [,1]
## [1,] 0.5500480
## [2,] 0.3282613
## [3,] -0.9875280
## [4,] 0.4148167
##
```

```
## $`Iterasi ke- `[[8]]
##           [,1]
## [1,]  0.5530889
## [2,]  0.3259500
## [3,] -0.9893018
## [4,]  0.4158380
##
## $`Iterasi ke- `[[9]]
##           [,1]
## [1,]  0.5544506
## [2,]  0.3249143
## [3,] -0.9900962
## [4,]  0.4162955
##
## $`Iterasi ke- `[[10]]
##           [,1]
## [1,]  0.5550606
## [2,]  0.3244505
## [3,] -0.9904520
## [4,]  0.4165004
##
## $`Iterasi ke- `[[11]]
##           [,1]
## [1,]  0.5553338
## [2,]  0.3242427
## [3,] -0.9906114
## [4,]  0.4165922
##
## $`Iterasi ke- `[[12]]
##           [,1]
## [1,]  0.5554562
## [2,]  0.3241496
## [3,] -0.9906828
## [4,]  0.4166333
##
## $`Iterasi ke- `[[13]]
##           [,1]
## [1,]  0.5555111
## [2,]  0.3241079
## [3,] -0.9907148
## [4,]  0.4166517
##
## $`Iterasi ke- `[[14]]
##           [,1]
## [1,]  0.5555356
```

```
## [2,] 0.3240892
## [3,] -0.9907291
## [4,] 0.4166600
##
## $`Iterasi ke- `[[15]]
##           [,1]
## [1,] 0.5555466
## [2,] 0.3240809
## [3,] -0.9907355
## [4,] 0.4166636
```

5.6 Metode SOR Gauss Seidel

5.6.1 Matriks Iterasi B

Dari metode yang ada di atas, kita bisa percepat kembali konvergensi dengan menambah komponen ω sehingga penentuan matriks iterasinya menjadi:

$$B_\omega = (D + \omega L)^{-1}[(1 - \omega)D - \omega U]$$

Perhatikan bahwa saat $\omega = 1$ didapat $B_{GS} = B_\omega$.

Bagaimana menentukan ω ?

$$\omega = \frac{2}{1 + \sqrt{1 - [\rho(B_J)]^2}}$$

5.6.2 Skema Interasi dalam Bentuk Matriks

Perhatikan bahwa:

$$(D + \omega L)x^{k+1} = ((1 - \omega)D - \omega U)x^k + \omega b$$

$$\rightarrow x^{k+1} = (D + \omega L)^{-1}((1 - \omega)D - \omega U)x^k + \omega(D + \omega L)^{-1}b$$

5.6.3 R Function SOR Gauss Seidel

Berikut ini adalah *function*-nya:

```
sor_jacobi_seigel_ikanx = function(A,b,x_awal,iter_max,tol_max,omega){
  # A adalah matriks
  # b adalah vector
  # x_awal adalah vektor utk initial iteration
  # tol_max toleransi error
  # iter_max iterasi maksimum

  # ambil diagonalnya
  da = diag(A)
  # bikin matriks diagonal
  D = diag(da)
  # bikin matriks C, yakni A yang sudah dihapus diagonalnya
  C = A - D
  # lower diagonal matrix
```

```

U = C
U[lower.tri(U)] = 0
# lower diagonal matrix
L = C
L[upper.tri(L)] = 0
# cari D inverse
Dinv = matlib::inv(D + omega * L)
# cari Jacobian matriks B
B = Dinv %*% ((1-omega) * D - omega * U)
# hitung b1
b1 = omega * Dinv %*% b
# iterasi
for(i in 1:iter_max){
  x = B %*% x_awal + b1
  if(norm(x-x_awal,"2") < tol_max * norm(x_awal,"2")) {break}
  x_awal = x
}
# output
output = list("iterasi yang dilakukan: " = i,
              "Solusi: " = x_awal)
return(output)
}

```

Berikut adalah contoh penggunaannya:

```

A = matrix(c(3,1,1,0,1,5,0,1,1,-1,3,1,0,2,1,4),ncol = 4)
b = c(1,4,-2,1)
iter_max = 100
tol_max = 10^(-5)
x_awal = c(1,1,1,1)
omega = 1.1

sor_jacobi_seigel_ikanx(A,b,x_awal,iter_max,tol_max,omega)

## $`iterasi yang dilakukan: `
## [1] 11
##
## $`Solusi: `
##           [,1]
## [1,]  0.5555461
## [2,]  0.3240804
## [3,] -0.9907361
## [4,]  0.4166644

```

Dengan ω tertentu didapatkan iterasi yang lebih sedikit.

CHAPTER IV

6 SISTEM PERSAMAAN NON LINEAR

6.1 Definisi SPNL

Misalkan suatu sistem persamaan non linear memiliki bentuk sebagai berikut:

$$\begin{aligned}f_1(x_1, x_2, \dots, x_n, K) &= 0 \\f_2(x_1, x_2, \dots, x_n, K) &= 0 \\&\dots \\f_n(x_1, x_2, \dots, x_n, K) &= 0\end{aligned}$$

dengan $f_i, i = 1, 2, \dots, n$ merupakan persamaan non linear.

6.2 Metode Newton

Untuk memudahkan perhitungan, mari kita lihat 2 fungsi sederhana berikut ini:

$$\begin{aligned}f_1(x_1, x_2) &= 0 \\f_2(x_1, x_2) &= 0\end{aligned}$$

Misalkan diberikan *initial approximation* berupa $x_1^{(0)}$ dan $x_2^{(0)}$. Kita bisa menghitung aproksimasi untuk iterasi berikutnya berupa:

$$\begin{aligned}x_1^{(1)} &= x_1^{(0)} + Dx_1^{(0)} \\x_2^{(1)} &= x_2^{(0)} + Dx_2^{(0)}\end{aligned}$$

Secara umum, bentuknya adalah sebagai berikut:

$$\begin{aligned}x_1^{(k+1)} &= x_1^{(k)} + Dx_1^{(k)} \\x_2^{(k+1)} &= x_2^{(k)} + Dx_2^{(k)}\end{aligned}$$

Oleh karena kita mengharapkan $x_1^{(k+1)}, x_2^{(k+1)}$ merupakan solusi dari **SPNL**, maka:

$$\begin{aligned}f(x^{(k+1)}) &\approx 0 \\f(x^{(k)} + Dx^{(k)}) &\approx 0\end{aligned}$$

Dengan menggunakan **deret Taylor**, kita bisa *expand* bentuk di atas menjadi:

$$f(x^{(k)} + Dx^{(k)}) = f(x^{(k)}) + J^{(k)}Dx^{(k)} + K$$

Dengan $J^{(k)}$ adalah *Jacobian matrix* yang berisi:

The Jacobian Matrix

Define the matrix $J(\mathbf{x})$ by

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix}.$$

Figure 26: Matriks Jacobi Newton SPNL

6.2.1 Skema Iterasi Metode Newton

Skema iterasinya menjadi berikut:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - J(\mathbf{x}^{(k)})^{-1} F(\mathbf{x}^{(k)})$$

6.2.2 R Function Metode Newton SPNL

Kelak *function* yang ditulis akan memanfaatkan skema iterasi tersebut. Oleh karena **R** bisa melakukan operasi matriks, kita akan memanfaatkan operasi tersebut dalam membuat programnya.

Langkah yang kritis pada metode ini adalah penentuan titik *initial* iterasi. Sebisa mungkin kita akan pilih titik *initial* yang dekat dengan solusi berdasarkan grafik yang ada.

Selain itu, kita perlu pertimbangkan juga persamaan yang ada di matriks Jacobi-nya. Jangan sampai ada pembagian dengan `nol` akibat kita salah memilih titik *initial*.

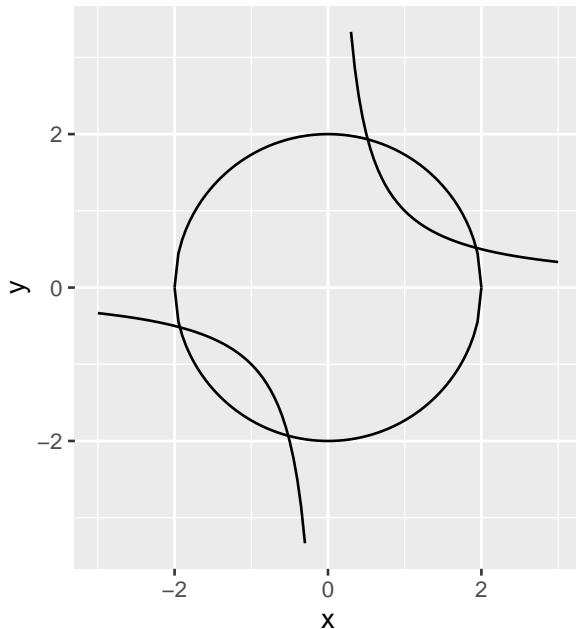
6.2.3 Contoh Soal

Soal I

Cari akar dari SPNL:

$$\begin{aligned}f_1(x_1, x_2) &= x_1^2 + x_2^2 - 4 = 0 \\f_2(x_1, x_2) &= x_1 x_2 - 1 = 0\end{aligned}$$

Jawab Mari kita buat grafiknya terlebih dahulu:



Digambar dengan R
ikanx101.com

Figure 27: Contoh Soal SPNL I

Terlihat ada 4 buah akar dari SPNL tersebut. Kita akan selesaikan satu persatu dengan cara memilih titik *initial* terdekat.

Sebelumnya, kita buat dulu matriks Jacobi nya:

$$J(x) = \begin{bmatrix} 2x_1 & 2x_2 \\ x_2 & x_1 \end{bmatrix}$$

Berdasarkan informasi di atas, kita akan selesaikan secara iteratif dengan **R**:

```
# menghitung norm untuk kriteria penghentian
# menghitung norm infinity dari vektor v
```

```

norm_vec_inf = function(x)max(abs(x))

# initial
x0 = c(0,3)

# bikin fungsi F(x1,x2)
F_x_k = function(x){
  f1 = x[1]^2 + x[2]^2 - 4
  f2 = x[1]*x[2] - 1
  xk = c(f1,f2)
  return(xk)
}

# bikin matriks jacobi
jax = function(x){
  a11 = 2*x[1]
  a12 = 2*x[2]
  a21 = x[2]
  a22 = x[1]
  J = matrix(c(a11,a12,a21,a22),ncol = 2,byrow = T)
  J_inv = matlib:::inv(J)
  return(J_inv)
}

# set toleransi max yang diinginkan
tol_max = 0.000005

# set max iterasi yang diperbolehkan
iter_max = 40

# kita mulai iterasinya
iter = 0
while(norm_vec_inf(F_x_k(x0)) > tol_max && iter <= iter_max){
  xk_new = x0 - jax(x0) %*% F_x_k(x0)
  x0 = xk_new
  iter = iter + 1
}

list("Solusi Final: " = x0,
     "Banyak iterasi: " = iter)

## $`Solusi Final: ` 
##      [,1]
## [1,] 0.517638

```

```
## [2,] 1.931852
##
## $`Banyak iterasi: `
## [1] 4
```

Mari kita plot kembali ke grafiknya:

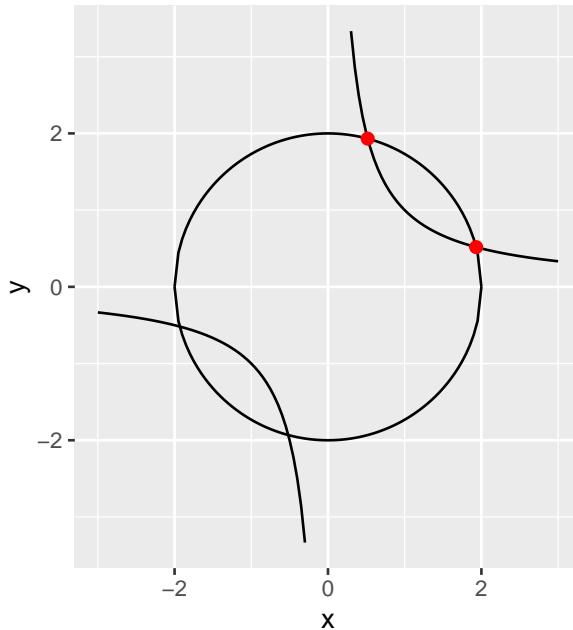


Digambar dengan R
ikanx101.com

Kita lakukan kembali untuk titik berikutnya, misal saya akan dekati dari titik initial $(2, 0)$ berikut:

```
## $`Solusi yang dihasilkan: `
##      [,1]
## [1,] 1.9318527
## [2,] 0.5176371
##
## $`Banyak iterasi: `
## [1] 7
```

Mari kita plot kembali ke grafiknya:

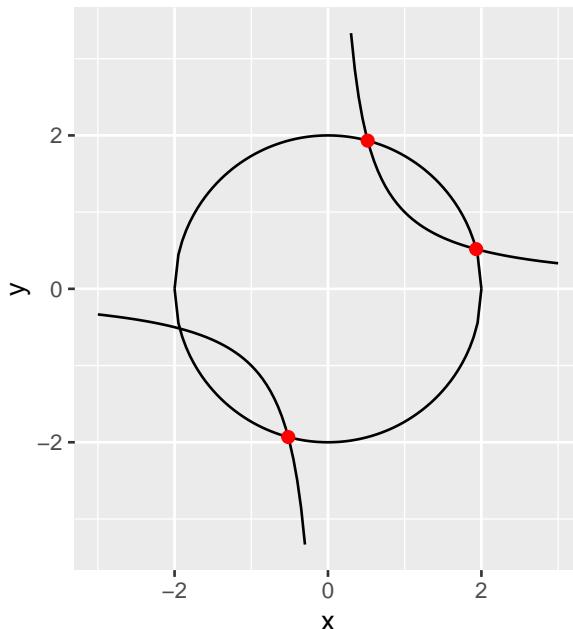


Digambar dengan R
ikanx101.com

Kita bisa ulangi hingga semua titik solusi terpenuhi.

Solusi ketiga:

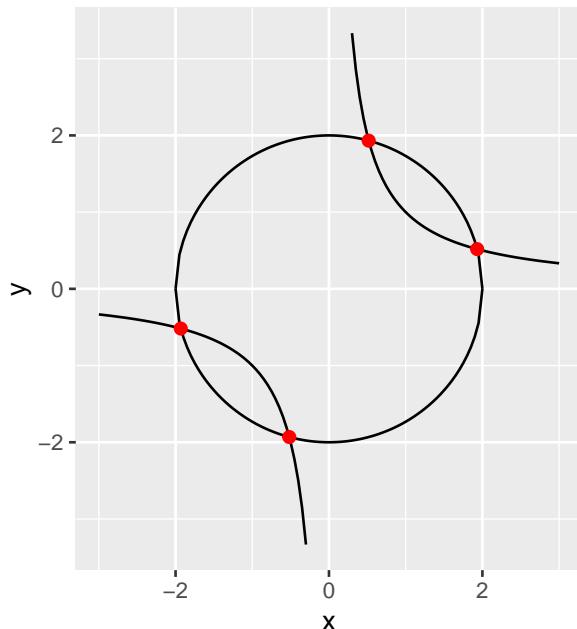
```
## $`Solusi yang dihasilkan: `  
## [1] [,1]  
## [1,] -0.5176371  
## [2,] -1.9318527  
##  
## $`Banyak iterasi: `  
## [1] 10
```



Digambar dengan R
ikanx101.com

Solusi keempat:

```
## $`Solusi yang dihasilkan: `  
##      [,1]  
## [1,] -1.9318527  
## [2,] -0.5176371  
##  
## $`Banyak iterasi: `  
## [1] 13
```



Digambar dengan R
ikanx101.com

Soal II

Cari aproksimasi solusi dari sistem persamaan non linear berikut:

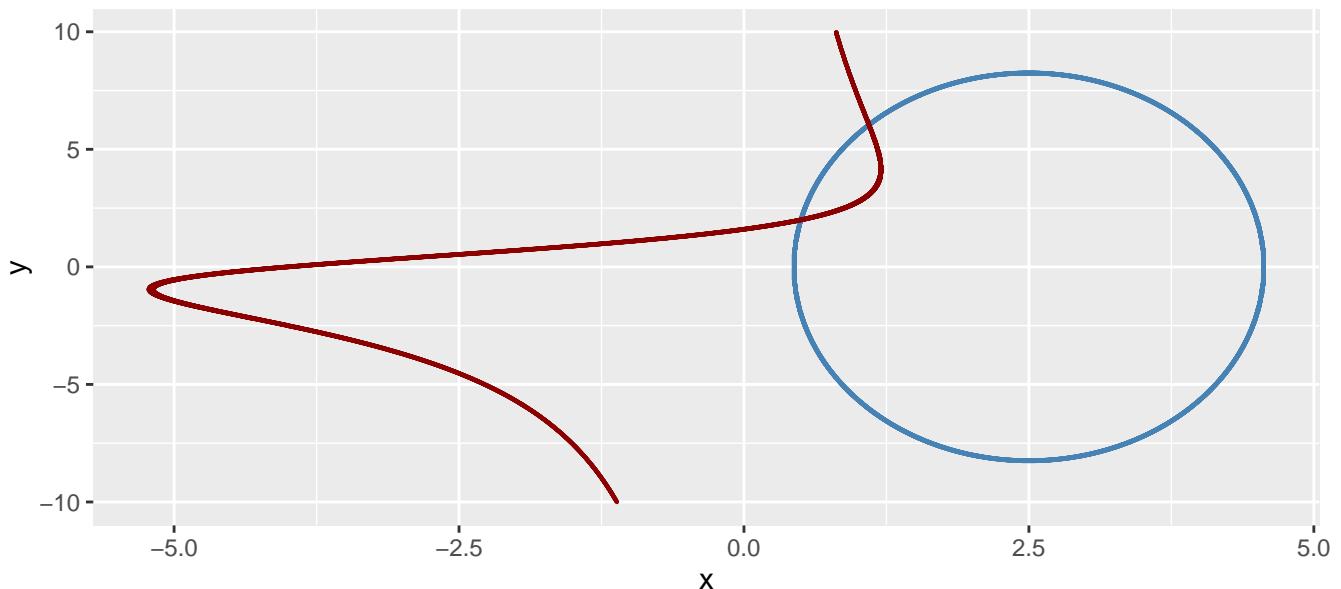
$$4x_1^2 - 20x_1 + \frac{1}{4}x_2^2 + 8 = 0$$

$$\frac{1}{2}x_1x_2^2 + 2x_1 - 5x_2 + 8 = 0$$

Jawab Untuk membantu kita menjawab soal tersebut, pertama-tama kita perlu membuat grafik fungsi dari SPNL tersebut:

Soal Contoh

Grafik $f_1(x_1, x_2)$ dan $f_2(x_1, x_2)$



Dibuat dengan R
ikanx101.com

Figure 28: Grafik Contoh Soal

Jika kita lihat grafik di atas, ada dua titik solusi yang akan kita cari. Oleh karena kita akan gunakan metode Newton, maka diperlukan dua sembarang *initial points*. Diharapkan iterasi dari dua *initial points* tersebut akan konvergen ke dua titik solusi yang dicari.

Kita akan mengambil dua titik *initial* berikut: $(0, 0)$ dan $(2, 10)$ secara sembarang. Perlu diperhatikan bahwa perbedaan *initial points* yang diambil akan mempengaruhi seberapa banyak iterasi yang diperlukan menuju konvergen ke titik solusi.

Berikut adalah grafiknya:

Titik hijau adalah *initial points* yang kita pilih.

Soal Contoh

Grafik $f_1(x_1, x_2)$ dan $f_2(x_1, x_2)$

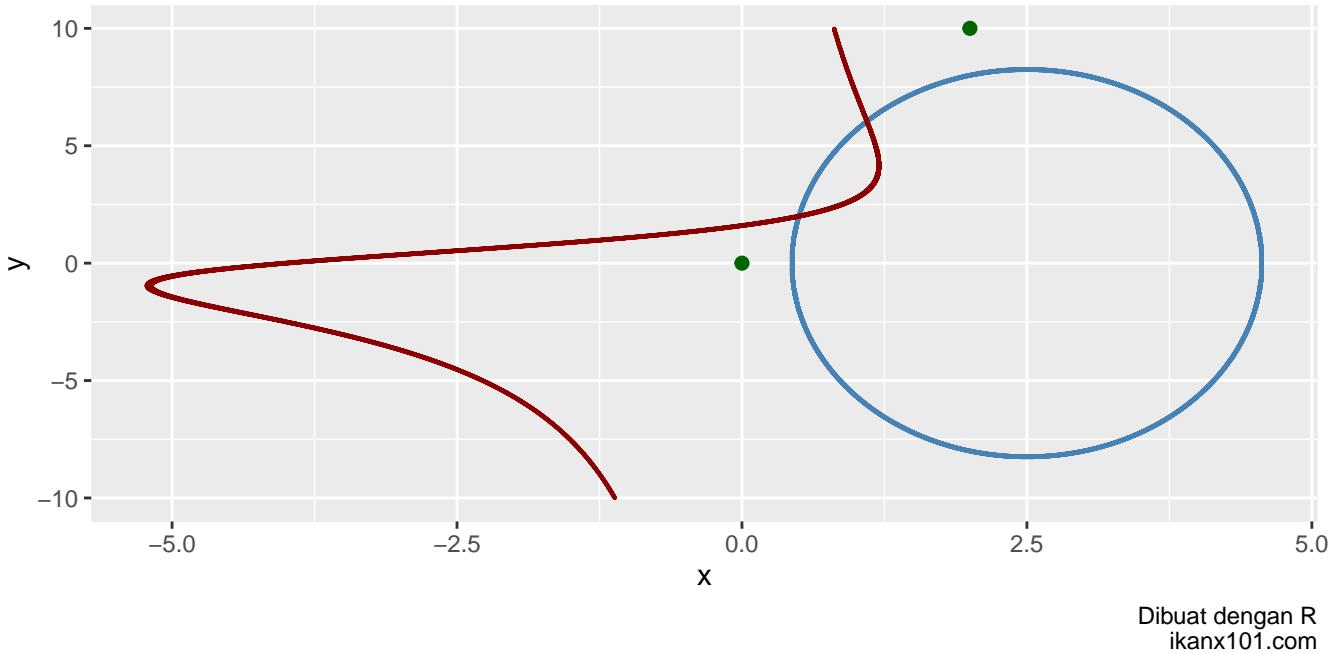


Figure 29: Initial Points Soal Contoh

Untuk menyelesaikan dengan metode Newton, kita perlu mencari persamaan di matriks Jacobi terlebih dahulu. Untuk membantu kita mencari turunan parsial dari kedua fungsi soal, kita akan gunakan `library(Ryacas)` di **R** berikut:

Berikut adalah matriks Jacobi yang sudah kita dapatkan:

$$J(x) = \begin{bmatrix} 8x_1 - 20 & \frac{x_2}{2} \\ \frac{x_2^2}{2} + 2 & x_2 x_1 - 5 \end{bmatrix}$$

Sekarang kita tinggal melakukan iterasi dengan skema berikut:

$$x^{(k+1)} = x^{(k)} - J(x^{(k)})^{-1} F(x^{(k)})$$

Mari kita coba selesaikan titik pertama terlebih dahulu.

```
# initial
x0 = c(0,0)

# bikin fungsi F(x1,x2)
F_x_k = function(x){
  f1 = 4 * x[1]^2 - 20*x[1] + (1/4) * x[2]^2 + 8
  f2 = (1/2) * x[1] * x[2]^2 + 2 * x[1] - 5 * x[2] + 8
}
```

```

xk = c(f1,f2)
return(xk)
}

# bikin matriks jacob
jax = function(x){
  a11 = 8*x[1]-20
  a12 = x[2]/2
  a21 = x[2]^2/2+2
  a22 = x[2]*x[1]-5
  J = matrix(c(a11,a12,a21,a22),ncol = 2,byrow = T)
  J_inv = matlib:::inv(J)
  return(J_inv)
}

# set toleransi max yang diinginkan
tol_max = 0.000001

# set max iterasi yang diperbolehkan
iter_max = 40

# kita mulai iterasinya
iter = 0
while(norm_vec_inf(F_x_k(x0)) > tol_max && iter <= iter_max){
  xk_new = x0 - jax(x0) %*% F_x_k(x0)
  x0 = xk_new
  iter = iter + 1
#   pesan = paste0("Iterasi ke-",iter,
#                 " menghasilkan: x1 = ",x0[1],
#                 " dan x2 = ",x0[2])
#   print(pesan)
}

list("Solusi Final: " = x0,
     "Banyak iterasi: " = iter)

## $`Solusi Final: ` 
##      [,1]
## [1,]  0.5
## [2,]  2.0
##
## $`Banyak iterasi: ` 
## [1] 4

```

Mari kita selesaikan titik berikutnya.

```
## $`Solusi Final: `  
## [1]  
## [1,] 1.096720  
## [2,] 6.040933  
##  
## $`Banyak iterasi: `  
## [1] 6
```

Kita dapatkan ada dua solusi dari SPNL ini, yakni:

1. (0.5,2)
2. (1.096720,6.040933)

Berikut adalah grafiknya:

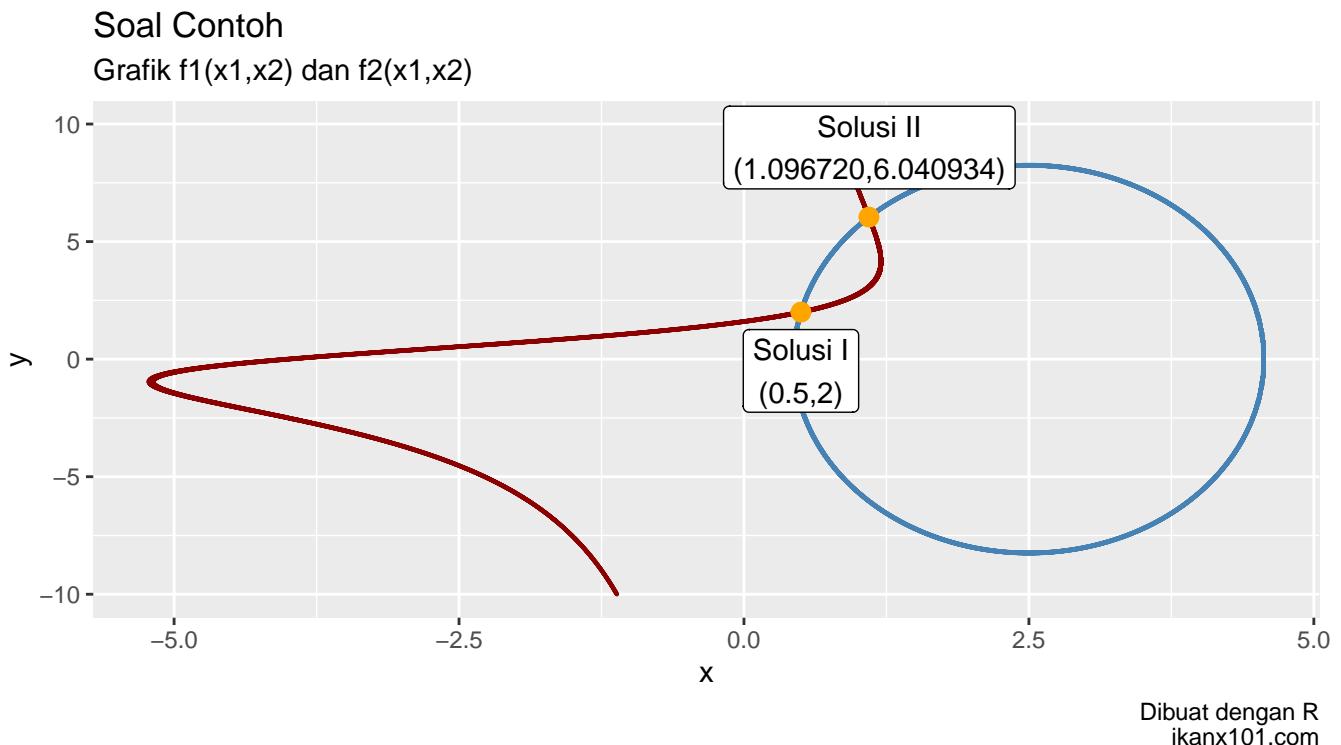


Figure 30: Solusi Pertama dan Kedua dari Soal

6.3 Metode Broyden

Metode Broyden merupakan metode modifikasi dari metode Newton. Perbedaan mendasar dari metode ini adalah kita menggunakan hampiran matriks Jacobi.

6.3.1 Algoritma Metode Broyden

Berikut adalah algoritma 10.2 dari buku:

**ALGORITHM
10.2**

Broyden

To approximate the solution of the nonlinear system $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ given an initial approximation \mathbf{x} :

INPUT number n of equations and unknowns; initial approximation $\mathbf{x} = (x_1, \dots, x_n)^t$; tolerance TOL ; maximum number of iterations N .

OUTPUT approximate solution $\mathbf{x} = (x_1, \dots, x_n)^t$ or a message that the number of iterations was exceeded.

Step 1 Set $A_0 = J(\mathbf{x})$ where $J(\mathbf{x})_{i,j} = \frac{\partial f_i}{\partial x_j}(\mathbf{x})$ for $1 \leq i, j \leq n$;
 $\mathbf{v} = \mathbf{F}(\mathbf{x})$. (Note: $\mathbf{v} = \mathbf{F}(\mathbf{x}^{(0)})$.)

Step 2 Set $A = A_0^{-1}$. (Use Gaussian elimination.)

Step 3 Set $\mathbf{s} = -A\mathbf{v}$; (Note: $\mathbf{s} = \mathbf{s}_1$.)
 $\mathbf{x} = \mathbf{x} + \mathbf{s}$; (Note: $\mathbf{x} = \mathbf{x}^{(1)}$.)
 $k = 2$.

Step 4 While ($k \leq N$) do Steps 5–13.

Step 5 Set $\mathbf{w} = \mathbf{v}$; (Save \mathbf{v}).
 $\mathbf{v} = \mathbf{F}(\mathbf{x})$; (Note: $\mathbf{v} = \mathbf{F}(\mathbf{x}^{(k)})$.)
 $\mathbf{y} = \mathbf{v} - \mathbf{w}$. (Note: $\mathbf{y} = \mathbf{y}_k$.)

Step 6 Set $\mathbf{z} = -A\mathbf{y}$. (Note: $\mathbf{z} = -A_{k-1}^{-1}\mathbf{y}_k$.)

Step 7 Set $p = -\mathbf{s}'\mathbf{z}$. (Note: $p = \mathbf{s}_k^t A_{k-1}^{-1} \mathbf{y}_k$.)

Step 8 Set $\mathbf{u}' = \mathbf{s}'A$.

Step 9 Set $A = A + \frac{1}{p}(\mathbf{s} + \mathbf{z})\mathbf{u}'$. (Note: $A = A_k^{-1}$.)

Step 10 Set $\mathbf{s} = -A\mathbf{v}$. (Note: $\mathbf{s} = -A_k^{-1}\mathbf{F}(\mathbf{x}^{(k)})$.)

Step 11 Set $\mathbf{x} = \mathbf{x} + \mathbf{s}$. (Note: $\mathbf{x} = \mathbf{x}^{(k+1)}$.)

Step 12 If $\|\mathbf{s}\| < TOL$ then OUTPUT (\mathbf{x});
(The procedure was successful.)
STOP.

Step 13 Set $k = k + 1$.

Step 14 OUTPUT ('Maximum number of iterations exceeded');
(The procedure was unsuccessful.)
STOP. ■

Figure 31: Algoritma Metode Broyden

6.3.2 Contoh Soal

Soal I

Aproksimasi solusi dari SPNL berikut ini:

$$x_1^2 + x_2 - 37 = 0$$

$$x_1 - x_2^2 - 5 = 0$$

$$x_1 + x_2 + x_3 - 3 = 0$$

$$-4 \leq x_1 \leq 8$$

$$-2 \leq x_2 \leq 2$$

$$-6 \leq x_3 \leq 0$$

Jawab Berikut adalah matriks Jacobi yang sudah kita dapatkan:

$$J(x) = \begin{bmatrix} 2x_1 & 1 & 0 \\ 1 & -2x_2 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Sekarang kita akan buat program iterasi metode Broyden berdasarkan **algoritma 10.2**.

Kita akan *generate random* titik di domain x_1, x_2, x_3 sebagai berikut:

Sebagai percobaan, saya akan coba dekati dengan *initial point* (5.3,1.8,-1.1). Berikut adalah programnya:

```
# bikin fungsi F(x1,x2,x3)
F_x_k = function(x){
  f1 = x[1]^2 + x[2] - 37
  f2 = x[1] - x[2]^2 - 5
  f3 = x[1] + x[2] + x[3] - 3
  xk = c(f1,f2,f3)
  return(xk)
}

# set toleransi max yang diinginkan
```

```

tol_max = 0.00001

# set max iterasi yang diperbolehkan
iter_max = 70

# kita mulai metode Broyden-nya sesuai dengan algoritma 10.2
# step 1
v = F_x_k(x0)

# step 2
# bikin matriks jacobi
jax = function(x){
  a11 = 2*x[1]
  a12 = 1
  a13 = 0
  a21 = 1
  a22 = -2*x[2]
  a23 = 0
  a31 = 1
  a32 = 1
  a33 = 1
  J = matrix(c(a11,a12,a13,a21,a22,a23,a31,a32,a33),ncol = 3,byrow = T)
  J_inv = matlib:::inv(J)
  return(J_inv)
}
A = jax(x0)

# step 3
s = -A %*% v
x = x0 + s
iter = 2

# step 4
while(norm_vec_inf(F_x_k(x)) > tol_max && iter <= iter_max){
  # step 5
  w = v
  v = F_x_k(x)
  y = v - w
  # step 6
  z = -A %*% y
  # step 7
  p = t(-s) %*% z
  p = as.numeric(p)
  # step 8
  s = p + A %*% y
  x = x + s
  iter = iter + 1
}

```

```

ut = t(s) %*% A
# step 9
A = A + ((s+z)/p) %*% ut
# step 10
s = -A %*% v
# step 11
x = x + s
iter = iter + 1
# output
# pesan = paste(x,collapse = ", ")
# pesan = paste0("Iterasi ke- ",iter,": (",pesan,")")
# print(pesan)
}

if(iter <= iter_max){
  list("Titik initial: " = x0,
       "Solusi Final: " = x,
       "Banyak iterasi: " = iter)
} else if(iter > iter_max){
  list("Titik initial: " = x0,
       "Solusi Final: " = "Tidak Konvergen atau melebihi batas iterasi")
}

## $`Titik initial: `
## [1] 5.3 1.8 -1.1
##
## $`Solusi Final: `
##      [,1]
## [1,] 6.000000
## [2,] 1.000002
## [3,] -4.000002
##
## $`Banyak iterasi: `
## [1] 7

```

Kita bisa mengulang prosedur di atas dengan cara-generate *random number* di domain x_1, x_2, x_3 sebagai berikut:

```

## $`Titik initial: `
## [1] 3.4 0.1 -2.8
##
## $`Solusi Final: `
##      [,1]
## [1,] 5.999996

```

```
## [2,] 0.9999987
## [3,] -3.9999984
##
## $`Banyak iterasi: `
## [1] 10

## $`Titik initial: `
## [1] 8.0 -1.1 -0.4
##
## $`Solusi Final: `
##      [,1]
## [1,] 6.171075
## [2,] -1.082162
## [3,] -2.088913
##
## $`Banyak iterasi: `
## [1] 7

## $`Titik initial: `
## [1] 4.4 2.0 -2.6
##
## $`Solusi Final: `
##      [,1]
## [1,] 6.000000
## [2,] 1.000002
## [3,] -4.000002
##
## $`Banyak iterasi: `
## [1] 7

## $`Titik initial: `
## [1] 7.9 1.0 -0.4
##
## $`Solusi Final: `
##      [,1]
## [1,] 6
## [2,] 1
## [3,] -4
##
## $`Banyak iterasi: `
## [1] 7

## $`Titik initial: `
## [1] 4.7 0.6 -4.9
```

```
##  
## $`Solusi Final: `  
##      [,1]  
## [1,]    6  
## [2,]    1  
## [3,]   -4  
##  
## $`Banyak iterasi: `  
## [1] 8  
  
## $`Titik initial: `  
## [1]  4.1 -0.3 -5.7  
##  
## $`Solusi Final: `  
##      [,1]  
## [1,]  6.000000  
## [2,]  1.000001  
## [3,] -4.000001  
##  
## $`Banyak iterasi: `  
## [1] 10
```

Dari berbagai titik *random* di atas, hanya ada dua titik solusi pada SPNL ini, yakni:

1. (6,1,-4)
2. (6,-1,-2)

CHAPTER V

7 NUMERICAL OPTIMIZATION

Domain optimisasi akan selalu ada di berbagai bidang kehidupan. Kali ini kita akan mencoba mencari solusi optimal secara numeric. Ada beberapa metode *metaheuristic* yang akan kita bahas di bagian-bagian selanjutnya.

7.1 *Steepest Descent* (Metode Gradien)

Salah satu metode yang populer dipakai dalam *numerical optimization* adalah metode gradien. Bentuk gradien yang ditulis di sini adalah gradien dari fungsi multi peubah (agar berlaku umum). Jikalau masalah yang dihadapi hanya fungsi satu peubah, tidak menjadi masalah.

7.1.1 Definisi

Misalkan kita memiliki masalah optimisasi:

Minimasi $f(x)$ dimana $x = (x_1, x_2, x_3, \dots, x_n)^T \in \mathbb{R}$.

Gradien dari $f(x)$ didefinisikan sebagai:

$$\text{grad } f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \dots, \frac{\partial f(x)}{\partial x_n} \right)^T$$

Vektor gradien $f(x)$ dari suatu titik, **secara lokal** (tidak berlaku global di domain fungsi hanya spesifik dalam rentang titik tersebut) menandai **arah laju kenaikan tercepat** dari $f(x)$. Kebalikannya vektor gradien $-f(x)$ menandai **arah laju penurunan tercepat**.

Catatan Perlu diperhatikan bahwa masalah **minimisasi** bisa diganti ke **maksimisasi** dengan cara mengubah $f(x)$ menjadi $-f(x)$.

7.1.2 Langkah Kerja

Berangkat dari titik P_0 , kita akan cari garis yang melalui P_0 , dinotasikan sebagai S_0 .

$$S_0 = \frac{-G_0}{||G_0||}$$

Dimana $G_0 = \text{grad } f(P_0)$.

Setelah bergerak dari *initial*, kita akan dapatkan P_1 . Titik minimum lokal akan berada pada titik x yang berada pada garis $X = P_0 + tS_0$.

Selanjutnya kita akan hitung G_1 dengan cara sama dengan sebelumnya:

$$S_1 = \frac{-G_1}{\|G_1\|}$$

Sehingga kita akan mendapatkan nilai P_2 di mana titik minimum lokal akan berada pada titik x yang berada pada garis $X = P_1 + tS_1$. Perhatikan bahwa ini adalah bentuk dari persamaan garis linear⁶.

Hasil iterasi terus menerus akan mendapatkan barisan:

$$\{P_0, P_1, P_2, \dots\}$$

dengan sifat:

$$f(P_0) > f(P_1) > \dots > f(P_k) > \dots$$

Kenapa berlaku demikian?

Karena kita sudah tentukan bahwa masalah minimisasi, sehingga nilai iterasi selanjutnya pasti akan lebih rendah dari nilai iterasi selanjutnya.

Iterasi akan berhenti saat gradien bernilai **nol**.

7.1.3 Algoritma Formal

Misal diketahui nilai P_k .

- **STEP 1** Hitung vektor gradien $G_k = \text{grad } f(P_k)$.
- **STEP 2** Cari vektor satunya $S_k = \frac{-G_k}{\|G_k\|}$.
- **STEP 3** Lakukan *single parameter minimization* untuk $\varphi(t) = f(P_k + tS_k)$ di interval $[0, b]$ dimana b adalah suatu bilangan **besar**. Ini akan menghasilkan suatu nilai $t = h_{\min}$ di mana titik minimum lokal di $\varphi(t)$ terjadi. Hubungan $\varphi(h_{\min}) = f(P_k + h_{\min}S_k)$ menunjukkan ini adalah minimum fungsi $f(x)$ di garis $X = P_k + h_{\min}S_k$
- **STEP 4** Hitung $P_{k+1} = P_k + h_{\min}S_k$.
- **STEP 5** Lakukan iterasi hingga $\|P_{k+1} - P_k\|$ sangat kecil.

7.1.4 Catatan Lain

∇ menandakan turunan parsial. Dalam kasus ini, gradien dari fungsi multi peubah.

Hasil iterasi *gradient descent* pasti menghasilkan nilai yang lebih kecil dibandingkan iterasi sebelumnya.

⁶Contoh persamaan garis linear $y = ax + b$

7.2 Metode *Neighborhood Search*

Metode ini mirip dengan metode simplex yang dijelaskan secara geometrik.

7.2.1 Definisi

Jika i adalah suatu *feasible solution*, kita definisikan $N(i)$ adalah himpunan solusi yang dekat jaraknya dengan i (tetanggaan).

7.2.2 Algoritma Formal

- **STEP 1** Pilih *initial feasible solution* i .
- **STEP 2** Pilih titik $j \in N(i)$ sehingga terpenuhi $f(j) \leq f(k), \forall k \in N(i)$.
- **STEP 3** Jika:
 - $f(j) \geq f(i)$ **STOP**.
 - $f(j) < f(i)$ definisikan $i = j$ lalu ulangi **STEP 2**.

7.2.3 Catatan Lain

Metode ini termasuk ke dalam metode yang *derivative-free*. Namun bisa jadi kita terjebak di titik minimum lokal padahal kita sedang mencari **minimum global**.

7.3 Metode *Simulated Annealing*

Metode gradien memiliki dua masalah karena:

1. Gradien harus dihitung berulang kali setiap algoritma dijalankan.
2. Bisa jadi kita terjebak di minimum lokal saat mendapatkan $\nabla f(x) = 0$.

Metode *neighborhood search* juga bisa terjebak dalam titik minimum lokal. Maka kita bisa menmodifikasi metode *neighborhood search* pada **STEP 3** sebagai berikut:

Boleh menerima j yang nilainya **tidak harus paling rendah** untuk dijadikan $N(j)$.

Metode ini bersifat *derivation-free*.

7.3.1 Masalah Fisis

Metode ini terinspirasi oleh proses fisis yang terjadi saat **peleburan metal**. Bagaimana metal dilebur, lalu melihat proses penurunan temperatur hingga metal mengeras kembali (molekul metal menjadi setimbang).

7.3.2 Algoritma Formal

Misalkan:

- T adalah *temperature* ($T > 0$). *State* ini adalah analogi dari *feasible solution*.
- E adalah *epoch* menandai iterasi.
- r adalah *cooling rate* ($0 < r < 1$). *Ratio* penurunan temperatur antar *epoch*.

Berikut adalah *pseudo-code*-nya:

```
# STEP 1
Generate state i (initial state)
T = T0

# STEP 2
Repeat
    Repeat
        k = 0
        generate state j (suatu neighbor state)
        delta = f(j) - f(i)
        if delta < 0
            i = j
        else if rand() < exp(-delta / T)
            i = j
        k = k + 1
    Until k = E
    T = r * T
Until N_max iteration
```

7.3.3 Catatan Penting

Metode ini adalah **killer method** yang bisa kita gunakan untuk menyelesaikan masalah optimisasi kontinu atau menyelesaikan masalah SPNL pada bagian sebelumnya. Caranya adalah dengan memodifikasi $F(X)$ menjadi masalah minimisasi.

7.3.4 Contoh Soal

Soal I

Cari minimum fungsi dari $f(x) = x^4 - 8x^3 + 20x^2 - 16.5x + 3$ di $0 < x < 4$

Figure 32: Grafik $f(x)$

Jawab Kita buat grafiknya terlebih dahulu:

Terlihat bahwa titik minimum global terletak di antara $[3, 4]$. Bagaimana penyelesaian dengan metode **AS**?

Pertama-tama saya akan buat *initial point* di $x = i = 1$. Prinsipnya adalah melakukan iterasi di sebanyak *epoch* untuk mencari *neighborhood* di sekitar i .

Berikut adalah *function* dalam **R**-nya:

```
# soal
f = function(x)x^4-8*x^3+20*x^2-16.5*x+3

# buat function untuk generate j
gen_j = function(x,selang){runif(1,0,1) - 0.5 * 2*selang + x}

# definisi kondisi awal
T = 10 # state
iter_max = 200 # batas max iterasi
epoch = 5
r = 0.7 # ratio penurunan suhu
lebar_selang = .2
```

```
# initial
i = 1.7
y = f(i) # nilai f dari tebakan awal

# indeks iterasi
m = 0 # untuk keperluan iterasi full

# template
min_global = c(i)

# kita mulai ya
while(m < iter_max){
  k = 0 # untuk keperluan epoch
  while(k < epoch){
    j = gen_j(i,lebar_selang)
    delta = f(j) - f(i)
    if(delta < 0){i = j}
    else if(runif(1,0,1) < exp((-delta)/T)) {i = j}
    y = ifelse(f(i) < y,
               f(i),
               y)
    k = k + 1
  }
  T = r * T
  m = m + 1
  min_global = c(min_global,i)
}

min_global %>% tail(1)
```

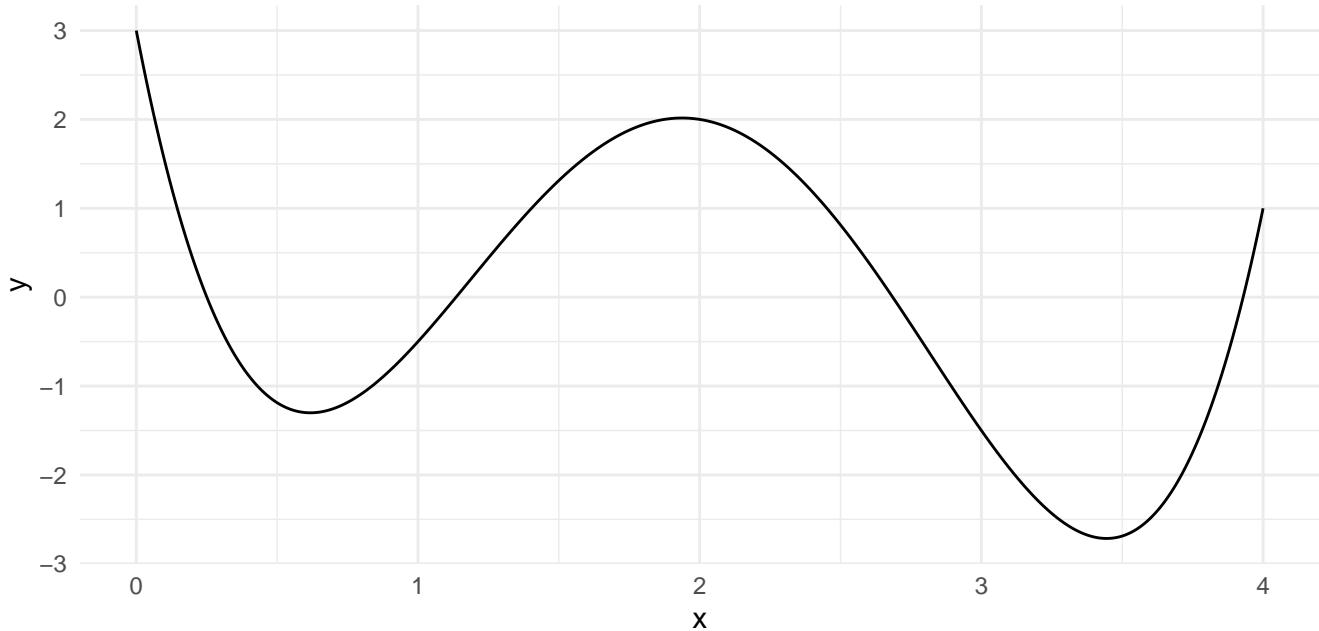
```
## [1] 3.44679
```

Soal II

Cari akar dari fungsi $f(x) = x^4 - 8x^3 + 20x^2 - 16.5x + 3$ di $0 < x < 4$

Jawab Kita buat grafiknya terlebih dahulu:

Grafik $f(x) = x^4 - 8x^3 + 20x^2 - 16.5x + 3$

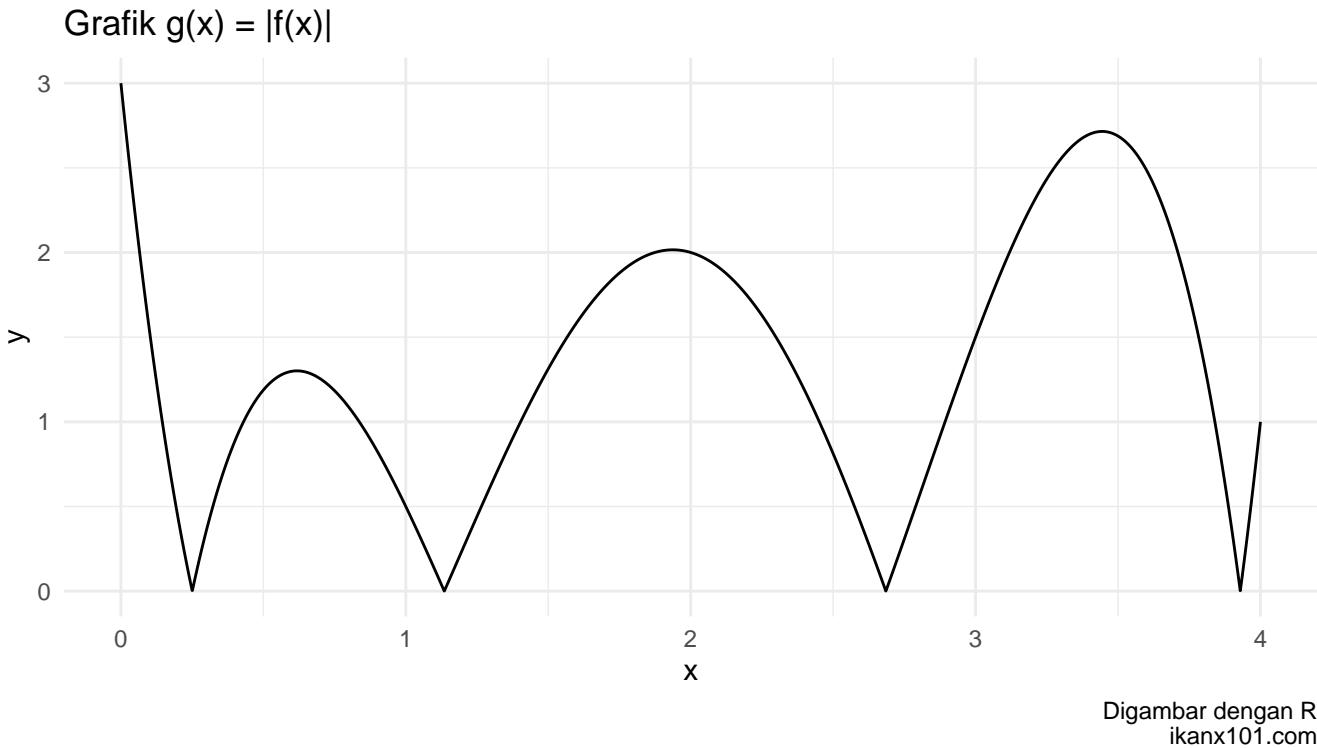


Digambar dengan R
ikanx101.com

Figure 33: Grafik $f(x)$

Terlihat ada 4 buah akar. Coba lihat lagi bagian 3.7.4. Kita bisa mengganti masalah optimisasi (minimisasi) menjadi masalah pencarian akar dengan memodifikasi $f(x)$.

Misal saya akan buat $g(x) = -\frac{1}{|f(x)|}$.

Figure 34: Grafik $f(x)$

Berikut adalah *function* dalam **R**-nya:

```
# buat function untuk generate j
gen_j = function(x,selang){runif(1,0,1) - 0.5 * 2*selang + x}

# template all
hasil = c()

for(ikanx in 1:100){

  # definisi kondisi awal
  T = 10 # state
  iter_max = 100 # batas max iterasi
  epoch = 10
  r = 0.8 # ratio penurunan suhu
  lebar_selang = .5

  # initial
  i = 0
  y = g(i) # nilai f dari tebakan awal

  # indeks iterasi
```

```

m = 0 # untuk keperluan iterasi full

# template
min_global = c(i)

# kita mulai ya
while(m < iter_max){
  k = 0 # untuk keperluan epoch
  while(k < epoch){
    j = gen_j(i,lebar_selang)
    delta = g(j) - g(i)
    if(delta < 0){i = j}
    if(runif(1,0,1) < exp((-delta)/T)) {i = j}
    y = ifelse(g(i) < y,
               g(i),
               y)
    k = k + 1
  }
  T = r * T
  m = m + 1
  min_global = c(min_global,i)
}

temp = min_global %>% tail(1) %>% as.numeric
hasil = c(hasil,temp)
}

unique(sort(round(hasil,1)))

```

[1] 0.2 0.3 1.1 2.7 3.9

Perhatikan bahwa saya melakukan looping hingga 100 kali agar hasilnya konvergen ke beberapa solusi saja.

Soal III

Cari minimum fungsi dari $f(x, y) = 0.5(x^4 - 16x^2 + 5x) + 0.5(y^4 - 16y^2 + 5y)$ di $-4 < x, y < 4$

Jawab Kita buat grafiknya terlebih dahulu:

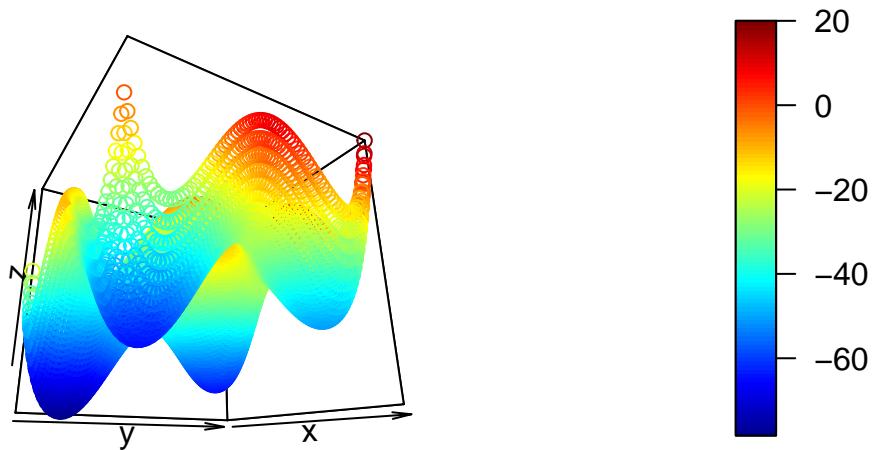


Figure 35: Grafik $f(x,y)$

Dengan *initial point* di $(1, 2)$ kita akan buat **R function**-nya:

```
# soal
f = function(v){0.5*(v[1]^4-16*v[1]^2+5*v[1]) + 0.5*(v[2]^4-16*v[2]^2+5*v[2])}

# buat function untuk generate j
gen_j = function(x,selang){runif(1,x-selang,x+selang)}

# definisi kondisi awal
T = 100 # state
iter_max = 500 # batas max iterasi
epoch = 100
r = 0.9 # ratio percepatan iterasi
lebar_selang = 0.9
```

```

# initial
i = c(0,0)
y = f(i) # nilai f dari tebakan awal

# indeks iterasi
m = 0 # untuk keperluan iterasi full

# kita mulai ya
while(m < iter_max){
  k = 0 # untuk keperluan epoch
  while(k < epoch){
    j = c(gen_j(i[1],lebar_selang),
          gen_j(i[2],lebar_selang))
    delta = f(j) - f(i)
    if(delta < 0){i = j}
    else if(runif(1,0,1) < exp((-delta)/T)) {i = j}
    y = ifelse(f(i) < y,
               f(i),
               y)
    k = k + 1
  }
  T = r * T
  m = m + 1
}

list("Titik minimum global" = i,
     "nilai f(x,y)" = f(i))

## $`Titik minimum global`
## [1] -2.906216 -2.905413
##
## $`nilai f(x,y)`
## [1] -78.33215

```

Sebagai catatan, karena ini adalah metode *meta heuristic* maka **tidak ada jaminan bahwa solusi yang didapatkan adalah yang paling optimal**.

Kita bisa menyiasatinya dengan cara melakukan percobaan berkali-kali agar hasil yang kita dapatkan konvergen ke suatu solusi tertentu.

7.4 Spiral Optimization Algorithm

Spiral Optimization Algorithm adalah salah satu metode *meta heuristic* yang digunakan untuk mencari minimum global dari suatu sistem persamaan.

Algoritmanya mudah dipahami dan intuitif tanpa harus memiliki latar keilmuan tertentu. Proses kerjanya adalah dengan melakukan *random number generating* pada suatu selang dan melakukan rotasi sekaligus kontraksi dengan titik paling minimum pada setiap iterasi sebagai pusatnya.

Berikut adalah algoritmanya:

```

INPUT
  m >= 2 # jumlah titik
  theta # sudut rotasi (0 <= theta <= 2pi)
  r      # kontraksi
  k_max # iterasi maksimum
PROCESS
  1 generate m buah titik secara acak
    x_i
  2 initial condition
    k = 0 # untuk keperluan iterasi
  3 cari x_* yang memenuhi
    min(f(x_*))

  4 lakukan rotasi dan kontraksi semua x_i
    x_* sebagai pusat rotasi
    k = k + 1
  5 ulangi proses 3 dan 4
  6 hentikan proses saat k = k_max
    output x_*

```

Berdasarkan algoritma di atas, salah satu proses yang penting adalah melakukan **rotasi** dan **kontraksi** terhadap semua titik yang telah di-*generate*.

Agar memudahkan penjeasan, saya akan memberikan ilustrasi geometri beserta operasi matriks aljabar terkait kedua hal tersebut.

Berikut adalah langkah-langkah yang ditempuh:

1. **Pertama** saya akan membuat program yang bisa merotasi suatu titik berdasarkan suatu θ tertentu.
2. **Kedua** saya akan memodifikasi program tersebut untuk melakukan rotasi sekaligus kontraksi dengan rasio r tertentu.
3. **Ketiga** saya akan memodifikasi program tersebut untuk melakukan rotasi sekaligus kontraksi dengan **titik pusat rotasi tertentu**.

7.4.1 Ilustrasi Geometris

7.4.1.1 Operasi Matriks Rotasi Misalkan saya memiliki titik $x \in \mathbb{R}^2$. Untuk melakukan rotasi sebesar θ , saya bisa menggunakan suatu matriks $A_{2 \times 2}$ berisi fungsi-fungsi trigonometri sebagai berikut:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

Berdasarkan operasi matriks di atas, saya membuat **program** di **R** dengan beberapa modifikasi. Sebagai contoh, saya akan membuat program yang bertujuan untuk melakukan rotasi suatu titik $x \in \mathbb{R}$ sebanyak n kali:

```
# mendefinisikan program
rotasi_kan = function(x0,rot){
  # menghitung theta
  theta = 2*pi/rot
  # definisi matriks rotasi
  A = matrix(c(cos(theta),-sin(theta),
              sin(theta),cos(theta)),
             ncol = 2,byrow = T)

  # membuat template
  temp = vector("list")
  temp[[1]] = x0
  # proses rotasi
  for(i in 2:rot){
    xk = A %*% x0
    temp[[i]] = xk
    x0 = xk
  }

  # membuat template data frame
  final = data.frame(x = rep(NA,rot),
                      y = rep(NA,rot))

  # gabung data dari list
  for(i in 1:rot){
    tempura = temp[[i]]
    final$x[i] = tempura[1]
    final$y[i] = tempura[2]
  }
  # membuat plot
  plot =
    ggplot() +
```

```
geom_point(aes(x,y),data = final) +
  geom_point(aes(x[1],y[1]),
             data = final,
             color = "red") +
  coord_equal() +
  labs(title = "titik merah adalah titik initial")

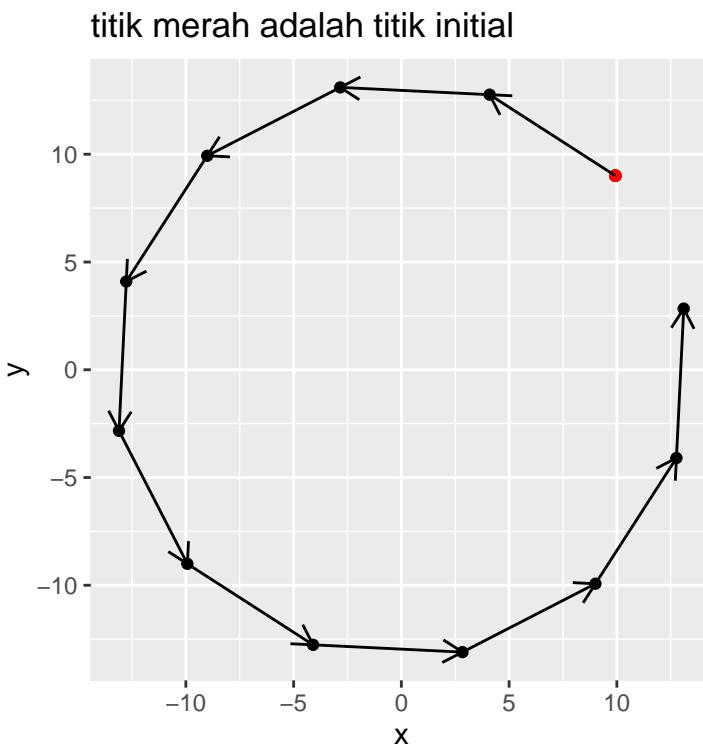
# enrich dengan garis panah
panah = data.frame(
  x_start = final$x[1:(rot-1)],
  x_end = final$x[2:rot],
  y_start = final$y[1:(rot-1)],
  y_end = final$y[2:rot]
)
# menambahkan garis panah ke plot
plot =
  plot +
  geom_segment(aes(x = x_start,
                    xend = x_end,
                    y = y_start,
                    yend = y_end),
               data = panah,
               arrow = arrow(length = unit(.3,"cm")))
}

# menyiapkan output
list("Grafik rotasi" = plot,
     "Titik-titik rotasi" = final)
}
```

Berikut adalah uji coba dengan titik sembarang berikut ini:

```
# uji coba
rot = 12 # berapa banyak rotasi
x0 = rand_titik(0,10) # generate random titik
rotasi_kan(x0,rot)

## $`Grafik rotasi`
```



```
##
## $`Titik-titik rotasi`  

##          x      y  

## 1  9.929247  9.006721  

## 2  4.095620 12.764673  

## 3 -2.835426 13.102341  

## 4 -9.006721  9.929247  

## 5 -12.764673  4.095620  

## 6 -13.102341 -2.835426  

## 7 -9.929247 -9.006721  

## 8 -4.095620 -12.764673  

## 9  2.835426 -13.102341  

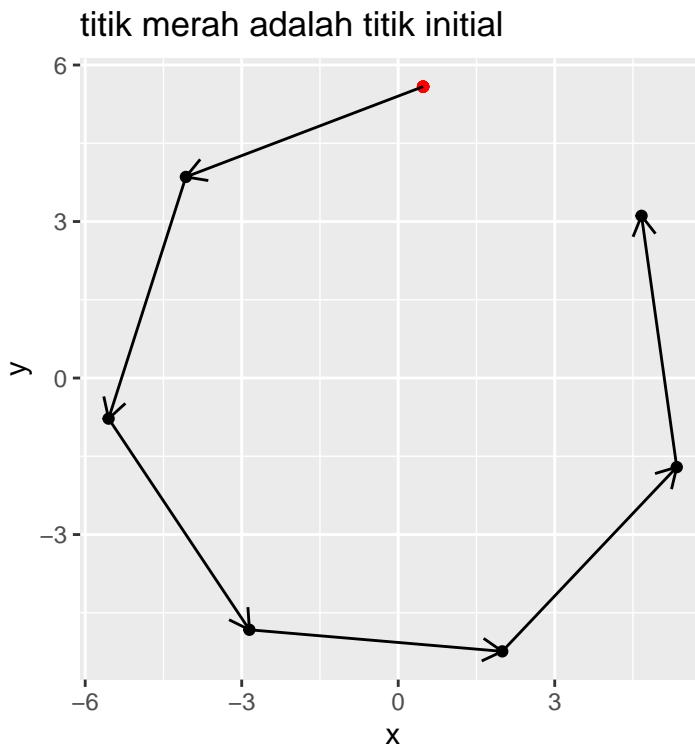
## 10 9.006721 -9.929247
```

```
## 11 12.764673 -4.095620
## 12 13.102341  2.835426
```

Uji coba kembali dengan titik sembarang lainnya berikut ini:

```
# uji coba
rot = 7 # berapa banyak rotasi
x0 = rand_titik(0,10) # generate random titik
rotasi_kan(x0,rot)

## $`Grafik rotasi`
```



```
##
## $`Titik-titik rotasi`
```

	x	y
## 1	0.4765614	5.5855158
## 2	-4.0698009	3.8551028
## 3	-5.5515202	-0.7782812
## 4	-2.8528315	-4.8256036
## 5	1.9940975	-5.2391481
## 6	5.3394304	-1.7075072
## 7	4.6640633	3.1099215

7.4.1.2 Operasi Matriks Rotasi dan Kontraksi Jika pada sebelumnya saya hanya melakukan **rotasi**, kali ini saya akan memodifikasi operasi matriks agar melakukan rotasi dan konstraksi secara bersamaan. Untuk melakukan hal tersebut, saya akan definisikan $r, 0 < r < 1$ dan melakukan operasi matriks sebagai berikut:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} r \\ r \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix}$$

Oleh karena itu saya akan modifikasi program **R** sebelumnya menjadi sebagai berikut:

```
# mendefinisikan program
rotasi_konstraksi_kan = function(x0,rot,r){
  # menghitung theta
  theta = 2*pi/rot
  # definisi matriks rotasi
  A = matrix(c(cos(theta),-sin(theta),
              sin(theta),cos(theta)),
             ncol = 2,byrow = T)

  # membuat template
  temp = vector("list")
  temp[[1]] = x0
  # proses rotasi dan konstraksi
  for(i in 2:rot){
    xk = A %*% x0
    xk = r * xk
    temp[[i]] = xk
    x0 = xk
  }

  # membuat template data frame
  final = data.frame(x = rep(NA,rot),
                      y = rep(NA,rot))

  # gabung data dari list
  for(i in 1:rot){
    tempura = temp[[i]]
    final$x[i] = tempura[1]
    final$y[i] = tempura[2]
  }
  # membuat plot
  plot =
  ggplot() +
  geom_point(aes(x,y),data = final) +
  geom_point(aes(x[1],y[1]),
```

```
        data = final,
        color = "red") +
coord_equal() +
labs(title = "titik merah adalah titik initial")

# enrich dengan garis panah
panah = data.frame(
  x_start = final$x[1:(rot-1)],
  x_end = final$x[2:rot],
  y_start = final$y[1:(rot-1)],
  y_end = final$y[2:rot]
)
# menambahkan garis panah ke plot
plot =
  plot +
  geom_segment(aes(x = x_start,
                    xend = x_end,
                    y = y_start,
                    yend = y_end),
                data = panah,
                arrow = arrow(length = unit(.3,"cm")))
}

# menyiapkan output
list("Grafik rotasi" = plot,
     "Titik-titik rotasi" = final)
}
```

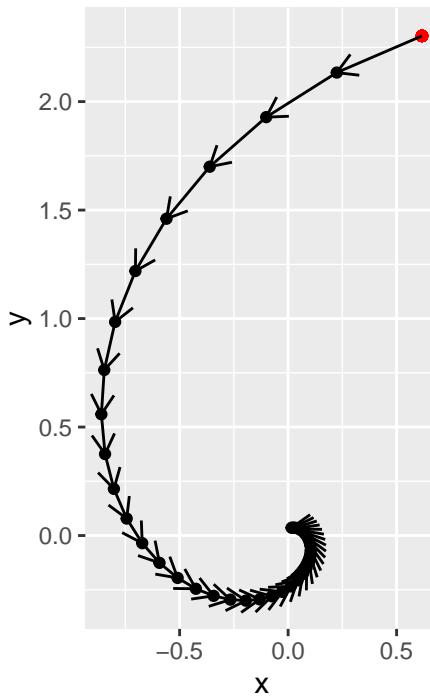
Berikutnya saya akan tunjukkan ilustrasi dari program ini.

Saya akan uji coba untuk sembarang titik berikut ini:

```
# uji coba
rot = 40 # berapa banyak rotasi
x0 = rand_titik(0,4) # generate random titik
r = .9
rotasi_konstraksi_kan(x0,rot,r)

## $`Grafik rotasi`
```

titik merah adalah titik initial



```
##
## $`Titik-titik rotasi`#
##           x          y
## 1  0.61708673 2.302703753
## 2  0.22434043 2.133798553
## 3 -0.10099909 1.928360274
## 4 -0.36127587 1.699937298
## 5 -0.56048107 1.460243027
## 6 -0.70381166 1.219127810
## 7 -0.79727416 0.984616131
## 8 -0.84733766 0.762995444
## 9 -0.86063788 0.558944001
```

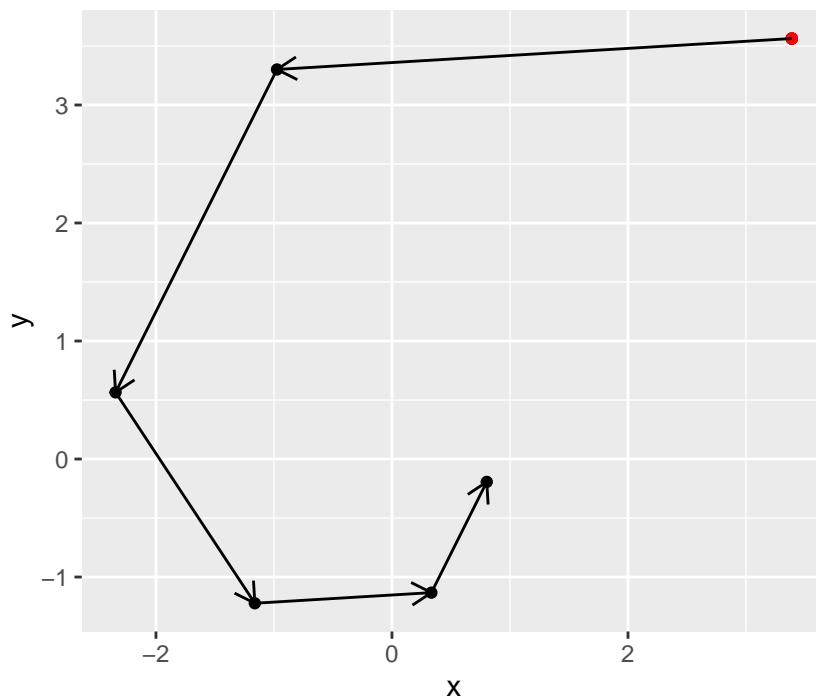
```
## 10 -0.84373210 0.375686142
## 11 -0.80290315 0.215164839
## 12 -0.74400955 0.078222670
## 13 -0.67237765 -0.035216206
## 14 -0.59273149 -0.125969107
## 15 -0.50915526 -0.195427666
## 16 -0.42508358 -0.245403952
## 17 -0.34331442 -0.277992310
## 18 -0.26604006 -0.295448374
## 19 -0.19489172 -0.300085874
## 20 -0.13099366 -0.294191191
## 21 -0.07502334 -0.279955019
## 22 -0.02727456 -0.259420090
## 23 0.01227913 -0.234443591
## 24 0.04392271 -0.206672690
## 25 0.06814141 -0.177531462
## 26 0.08556707 -0.148217481
## 27 0.09692993 -0.119706335
## 28 0.10301648 -0.092762434
## 29 0.10463347 -0.067954542
## 30 0.10257813 -0.045674664
## 31 0.09761428 -0.026159021
## 32 0.09045419 -0.009510050
## 33 0.08174543 0.004281468
## 34 0.07206231 0.015314903
## 35 0.06190139 0.023759442
## 36 0.05168024 0.029835391
## 37 0.04173902 0.033797375
## 38 0.03234426 0.035919625
## 39 0.02369428 0.036483437
## 40 0.01592577 0.035766781
```

Saya akan uji coba kembali untuk sembarang titik lainnya berikut ini:

```
# uji coba
rot = 6 # berapa banyak rotasi
x0 = rand_titik(0,4) # generate random titik
r = .7
rotasi_konstraksi_kan(x0,rot,r)

## $`Grafik rotasi`
```

titik merah adalah titik initial



```
##
## $`Titik-titik rotasi`#
##      x      y
## 1  3.3880156  3.5630947
## 2 -0.9742059  3.3009585
## 3 -2.3420718  0.5647545
## 4 -1.1620894 -1.2221415
## 5  0.3341526 -1.1322288
## 6  0.8033306 -0.1937108
```

Catatan penting:

Terlihat bahwa semakin banyak rotasi dan konstraksi yang dilakukan akan membuat titik *initial* menuju pusat (0, 0).

7.4.1.3 Operasi Matriks Rotasi dan Kontraksi dengan Titik x^* Sebagai Pusatnya
 Salah satu prinsip utama dari *spiral optimization algorithm* adalah menjadikan titik x^* sebagai pusat rotasi di setiap iterasinya. Operasi matriksnya adalah sebagai berikut:

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} + \begin{bmatrix} r \\ r \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \left(\begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} - \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} \right)$$

Oleh karena itu kita akan modifikasi program bagian sebelumnya menjadi seperti ini:

```
# mendefinisikan program
rotasi_konstraksi_pusat_kan = function(x0,rot,r,x_bin){
  # pusat rotasi
  pusat = x_bin
  # menghitung theta
  theta = 2*pi/rot
  # definisi matriks rotasi
  A = matrix(c(cos(theta),-sin(theta),
              sin(theta),cos(theta)),
             ncol = 2,byrow = T)

  # membuat template
  temp = vector("list")
  temp[[1]] = x0
  # proses rotasi dan konstraksi
  for(i in 2:rot){
    xk = A %*% (x0-pusat) # diputar dengan x_bin sebagai pusat
    xk = pusat + (r * xk)
    temp[[i]] = xk
    x0 = xk
  }

  # membuat template data frame
  final = data.frame(x = rep(NA,rot),
                      y = rep(NA,rot))

  # gabung data dari list
  for(i in 1:rot){
    tempura = temp[[i]]
    final$x[i] = tempura[1]
    final$y[i] = tempura[2]
  }
  # membuat plot
  plot =
    ggplot() +
    geom_point(aes(x,y),data = final) +
```

```
geom_point(aes(x[1],y[1]),
           data = final,
           color = "red") +
geom_point(aes(x = pusat[1],
               y = pusat[2]),
           color = "blue") +
labs(title = "titik merah adalah titik initial\ntitik biru adalah pusat rotasi")

# enrich dengan garis panah
panah = data.frame(
  x_start = final$x[1:(rot-1)],
  x_end = final$x[2:rot],
  y_start = final$y[1:(rot-1)],
  y_end = final$y[2:rot]
)
# menambahkan garis panah ke plot
plot =
  plot +
  geom_segment(aes(x = x_start,
                    xend = x_end,
                    y = y_start,
                    yend = y_end),
               data = panah,
               arrow = arrow(length = unit(.3,"cm")))
# menyiapkan output
list("Grafik rotasi" = plot,
     "Titik-titik rotasi" = final)
}
```

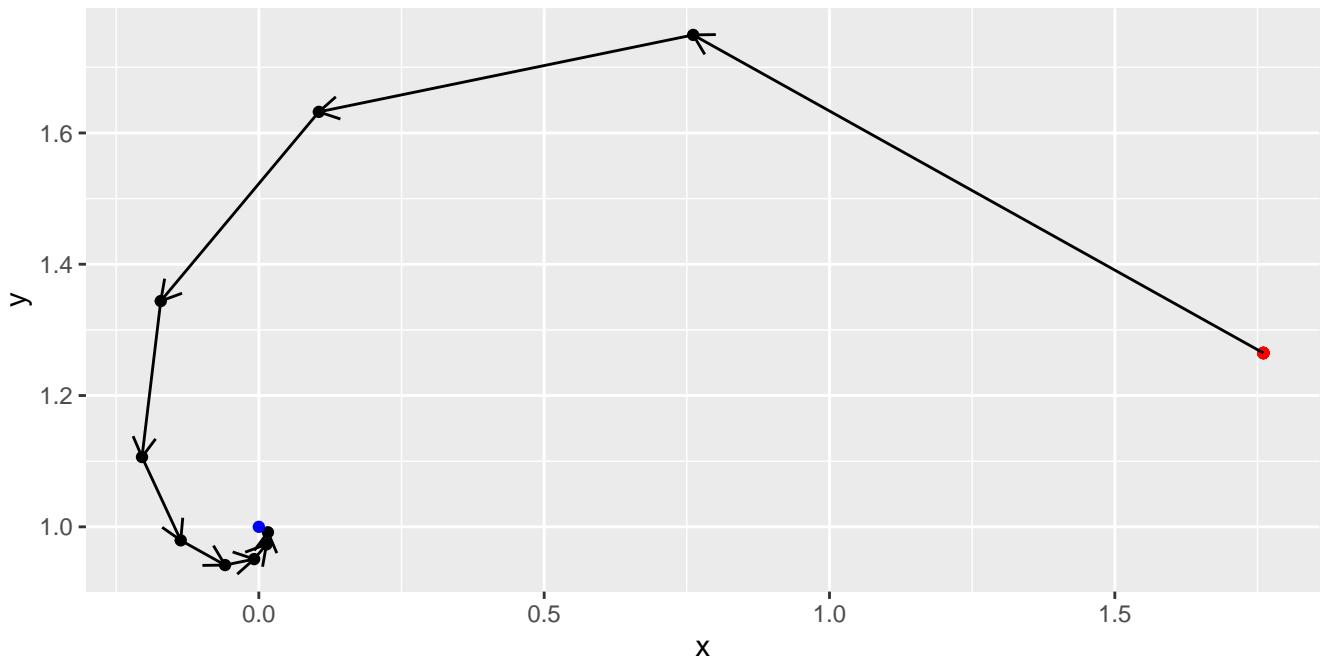
Berikutnya saya akan tunjukkan ilustrasi dari program ini.

Saya akan coba dengan sembarang titik berikut:

```
# uji coba
rot = 10 # berapa banyak rotasi
x0 = rand_titik(0,4) # generate random titik
x_bintang = c(0,1) # contoh pusat rotasi
r = .6
rotasi_konstraksi_pusat_kan(x0,rot,r,x_bintang)
```

```
## $`Grafik rotasi`
```

titik merah adalah titik initial
titik biru adalah pusat rotasi



```
##
## $`Titik-titik rotasi`#
##           x      y
## 1   1.760299423 1.2647985
## 2   0.761080487 1.7493427
## 3   0.105164666 1.6321497
## 4  -0.171892973 1.3439405
## 5  -0.204736484 1.1063305
## 6  -0.136880883 0.9794093
## 7  -0.059181619 0.9417311
## 8  -0.008177604 0.9508440
```

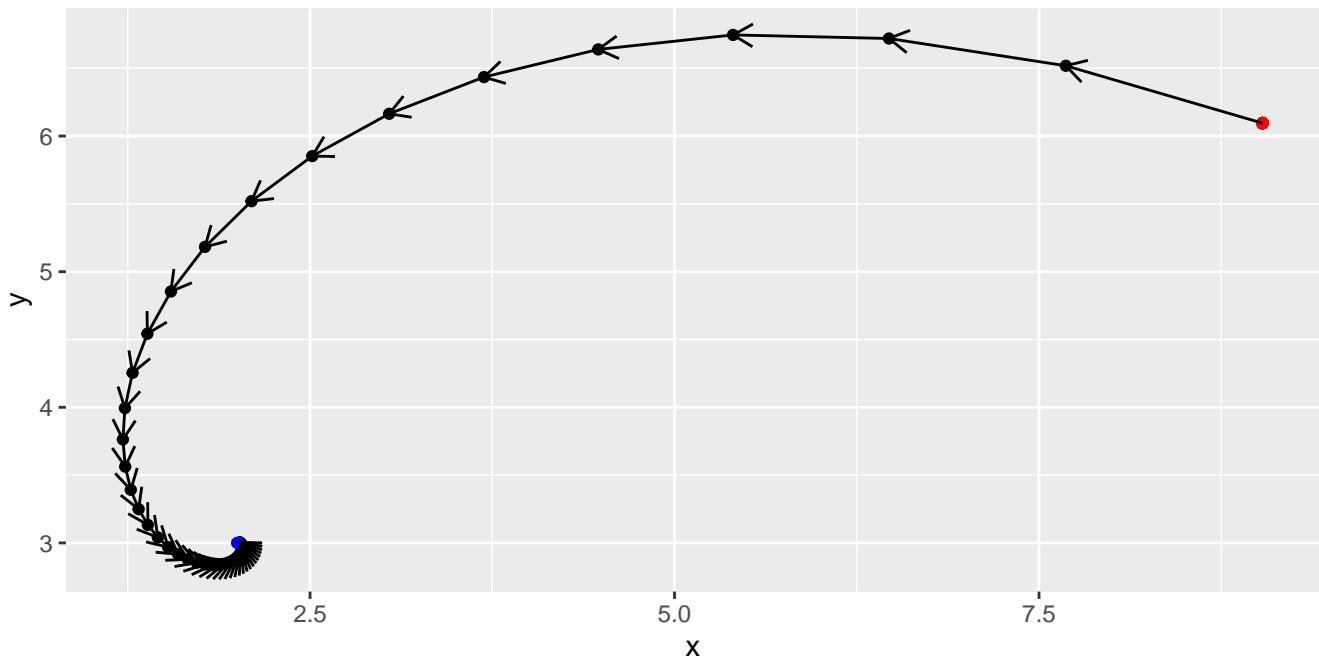
```
## 9  0.013366398 0.9732552
## 10 0.015920309 0.9917317
```

Saya akan coba kembali dengan sembarang titik lainnya:

```
# uji coba
rot = 45 # berapa banyak rotasi
x0 = rand_titik(0,10) # generate random titik
x_bintang = c(2,3) # contoh pusat rotasi
r = .87
rotasi_konstraksi_pusat_kan(x0,rot,r,x_bintang)
```

```
## $`Grafik rotasi`
```

titik merah adalah titik initial
titik biru adalah pusat rotasi



```
##
## $`Titik-titik rotasi`
```

	x	y
1	9.032531	6.095544
2	7.683949	6.518417
3	6.470899	6.719449
4	5.401475	6.745768
5	4.476944	6.638956
6	3.693362	6.434992
7	3.042977	6.164393
8	2.515413	5.852514

```
## 9  2.098661 5.519942
## 10 1.779884 5.182960
## 11 1.546049 4.854040
## 12 1.384417 4.542353
## 13 1.282906 4.254253
## 14 1.230334 3.993754
## 15 1.216583 3.762961
## 16 1.232680 3.562460
## 17 1.270826 3.391670
## 18 1.324369 3.249148
## 19 1.387754 3.132843
## 20 1.456445 3.040318
## 21 1.526828 2.968921
## 22 1.596109 2.915933
## 23 1.662214 2.878670
## 24 1.723677 2.854571
## 25 1.779547 2.841250
## 26 1.829294 2.836539
## 27 1.872723 2.838504
## 28 1.909901 2.845455
## 29 1.941089 2.855945
## 30 1.966688 2.868759
## 31 1.987192 2.882898
## 32 2.003144 2.897562
## 33 2.015112 2.912127
## 34 2.023659 2.926124
## 35 2.029328 2.939218
## 36 2.032627 2.951185
## 37 2.034019 2.961895
## 38 2.033923 2.971290
## 39 2.032702 2.979373
## 40 2.030671 2.986189
## 41 2.028096 2.991815
## 42 2.025197 2.996350
## 43 2.022150 2.999906
## 44 2.019094 3.002601
## 45 2.016135 3.004553
```

7.4.2 Program *Spiral Optimization Algorithm*

Berbekal program yang telah dituliskan di bagian sebelumnya, kita akan sempurnakan program untuk melakukan *spiral optimization* sebagai berikut:

```
soa_mrf = function(N,      # banyak titik
                    x1_d,  # batas bawah x1
                    x1_u,  # batas atas x1
                    x2_d,  # batas bawah x2
                    x2_u,  # batas atas x2
                    rot,    # berapa banyak rotasi
                    k_max, # iterasi maks
                    r){    # berapa rate konstraksi

# N pasang titik random di selang [a,b] di R2
x1 = runif(N,x1_d,x1_u)
x2 = runif(N,x2_d,x2_u)
# hitung theta
theta = 2*pi / rot
# definisi matriks rotasi
A = matrix(c(cos(theta),-sin(theta),
             sin(theta),cos(theta)),
            ncol = 2,byrow = T)
# bikin data frame
temp = data.frame(x1,x2) %>% mutate(f = f(x1,x2))
# proses iterasi
for(i in 1:k_max){
  # mencari titik x* dengan min(f)
  f_min =
    temp %>%
    filter(f == min(f))
  pusat = c(f_min$x1,f_min$x2)
  for(j in 1:N){
    # kita akan ambil titiknya satu persatu
    x0 = c(temp$x1[j],temp$x2[j])

    # proses rotasi dan konstraksi terhadap pusat x*
    xk = A %*% (x0-pusat) # diputar dengan x_bin sebagai pusat
    xk = pusat + (r * xk)

    # proses mengembalikan nilai ke temp
    temp$x1[j] = xk[1]
    temp$x2[j] = xk[2]
  }
}
```

```

# hitung kembali nilai f(x1,x2)
temp = temp %>% mutate(f = f(x1,x2))
}
# proses output hasil
output = temp %>% filter(f == min(f))
return(output)
}

```

7.4.2.1 Contoh Penggunaan Program Kita akan coba performa program tersebut untuk menyelesaikan fungsi berikut:

$$f(x_1, x_2) = \frac{x_1^4 - 16x_1^2 + 5x_1}{2} + \frac{x_2^4 - 16x_2^2 + 5x_2}{2}$$

$$-4 \leq x_1, x_2 \leq 4$$

Dengan $r = 0.8, N = 50, rot = 20, k_{max} = 60$.

```

# definisi
N = 50
a = -4 # x1 dan x2 punya batas bawah yang sama
b = 4 # x1 dan x2 punya batas atas yang sama
k_max = 70
r = .75
rot = 30
f = function(x1,x2){
  ((x1^4 - 16 * x1^2 + 5 * x1)/2) + ((x2^4 - 16 * x2^2 + 5*x x2)/2)
}
# solving
soa_mrf(N,a,b,a,b,rot,k_max,r)

```

```

##           x1           x2           f
## 1 -2.922185 -2.936256 -78.30756

```

Catatan

Pada algoritma ini, penentuan θ, r, x menjadi penentu hasil perhitungan.

7.4.3 Mengubah Optimisasi Menjadi Pencarian Akar

Spiral optimization algorithm adalah suatu metode untuk mencari solusi minimum global. Jika kita hendak memakainya untuk mencari suatu akar persamaan (atau sistem persamaan), kita bisa melakukan modifikasi pada fungsi-fungsi yang terlibat (membuat fungsi *merit*).

Misalkan suatu sistem persamaan non linear:

$$g_1(x_1, x_2, \dots, x_n) = 0$$

$$g_2(x_1, x_2, \dots, x_n) = 0$$

$$g_n(x_1, x_2, \dots, x_n) = 0$$

dengan $(x_1, x_2, \dots, x_n)^T \in D$

$$D = a_1, b_1 \times a_2, b_2 \times \dots \times a_n, b_n \subset \mathbb{R}^n$$

7.4.3.1 Pencarian Akar Sistem di atas memiliki solusi $x = (x_1, x_2, \dots, x_n)^T$ jika $F(x)$ yang kita definisikan sebagai:

$$F(x) = \frac{1}{1 + \sum_{i=1}^n |g_i(x)|}$$

memiliki nilai maksimum sama dengan 1. **Akibatnya algoritma yang sebelumnya adalah mencari $\min F(x)$ diubah menjadi $\max F(x)$.** Kenapa demikian?

Karena jika $F(x) = 1$ artinya $\sum_{i=1}^n |g_i(x)| = 0$ yang merupakan akar dari $g_i, i = 1, 2, \dots, n$.

7.4.4 Soal Latihan

Tentukanlah akar-akar sistem persamaan berikut dengan **SOA**. Buatlah terlebih dahulu *contour plot*-nya:

$$f_1(x_1, x_2) = \cos(2x_1) - \cos(2x_2) - 0.4 = 0$$

$$f_2(x_1, x_2) = 2(x_2 - x_1) + \sin(x_2) - \sin(x_1) - 1.2 = 0$$

dengan $-10 \leq x_1, x_2 \leq 10$

Jawaban

7.4.4.1 Contour Plot Pertama-tama, saya akan buat *contour plot* dari $f_1(x_1, x_2)$ sebagai berikut:

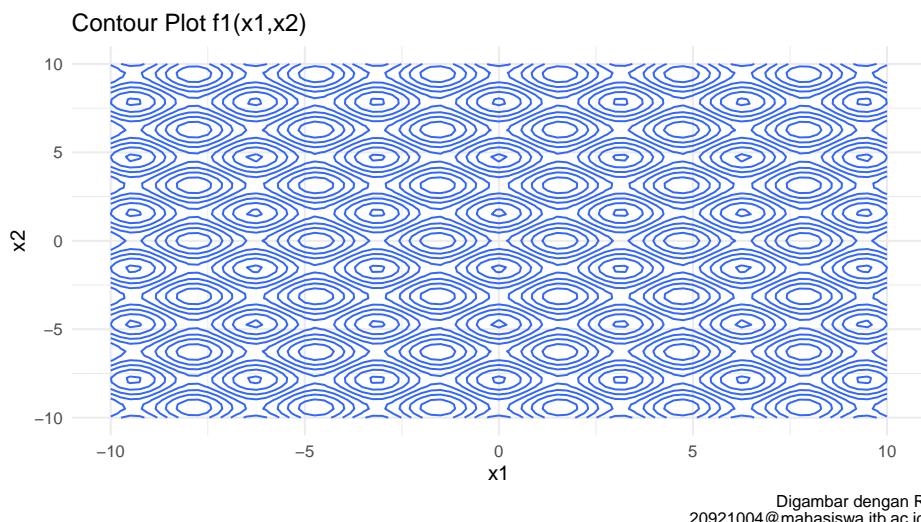


Figure 36: Contour Plot Soal 1: f1

Selanjutnya, saya akan buat *contour plot* dari $f_2(x_1, x_2)$ sebagai berikut:

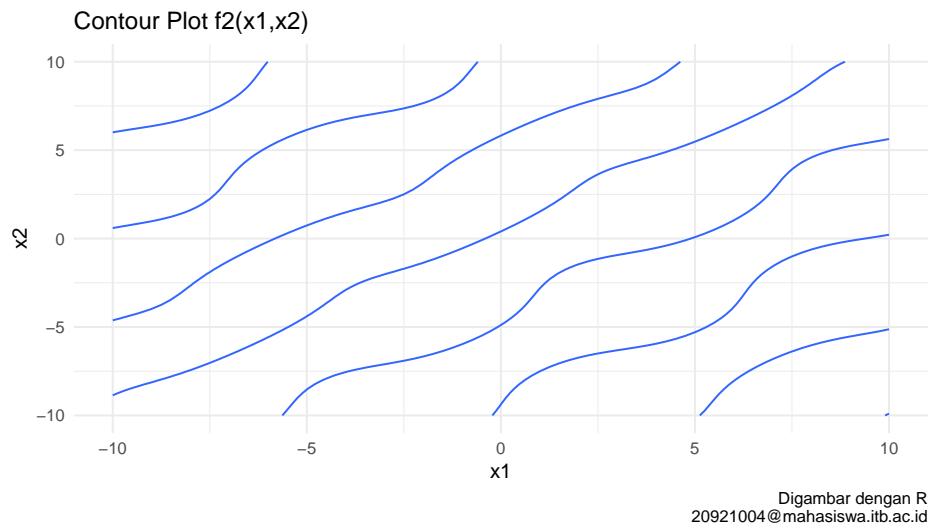


Figure 37: Contour Plot Soal 1: f_2

7.4.4.2 Grafik Sistem Persamaan Kita akan mencari akar-akar sistem persamaan saat $f_1 = 0$ dan $f_2 = 0$ dengan bantuan grafik sebagai berikut:

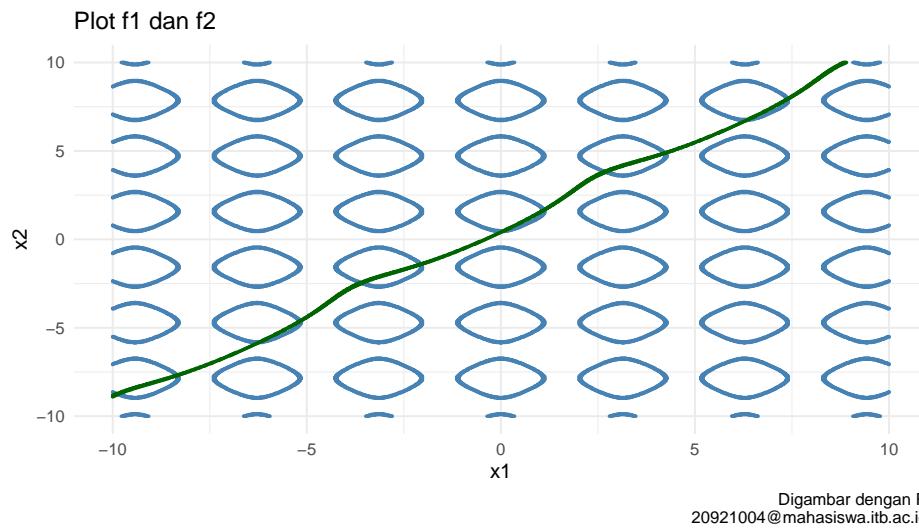


Figure 38: Plot Soal 1: f1 dan f2

Terlihat bahwa ada beberapa titik solusi (persinggungan antara $f_1(x_1, x_2)$ dengan $f_2(x_1, x_2)$).

7.4.4.3 Mencari Akar Sistem Persamaan Untuk mencari akarnya kita perlu membentuk $F(x)$ sebagaimana yang telah dijelaskan pada bagian sebelumnya.

```
# fungsi f1 dan f2 dari soal
f1 = function(x1,x2){cos(2*x1) - cos(2*x2) - 0.4}
f2 = function(x1,x2){2*(x2 - x1) + sin(x2) - sin(x1) - 1.2}
# membuat F(x)
# saya notasikan sebagai f kecil
f = function(x1,x2){
  sum = abs(f1(x1,x2)) + abs(f2(x1,x2))
  bawah = 1 + sum
  hasil = 1/bawah
  return(hasil)
}
```

Oleh karena solusi dari grafik ada banyak, maka kita akan *run* program yang telah dibuat sebelumnya berulang kali:

```
# solving
N = 50
a = -10 # x1 dan x2 punya batas yang sama
b = 10 # x1 dan x2 punya batas yang sama
rot = 20
k_max = 60
r = .65
# run I
soa_mrf_2(N,a,b,a,b,rot,k_max,r)
```

```
##           x1           x2           f
## 1 1.099944 1.646985 0.9999967
```

```
# run II
soa_mrf_2(N,a,b,a,b,rot,k_max,r)
```

```
##           x1           x2           f
## 1 -9.834834 -8.804864 0.8602116
```

```
# run III
soa_mrf_2(N,a,b,a,b,rot,k_max,r)
```

```
##           x1           x2           f
## 1 -3.622671 -2.489113 0.9142902
```

Berikutnya saya coba *run* sebanyak **100 kali**, berikut adalah rekap semua akar yang saya dapatkan:

```
##          x1      x2      f
## 1 -10.096 -8.646 0.64
## 2 -9.939 -8.785 0.84
## 3 -9.883 -8.750 0.99
## 4 -8.430 -7.763 0.83
## 5 -8.386 -7.792 0.80
## 6 -8.350 -7.734 0.90
## 7 -8.348 -7.695 1.00
## 8 -8.342 -7.755 0.86
## 9 -6.380 -5.950 0.78
## 10 -6.241 -5.836 0.97
## 11 -6.223 -5.818 0.99
## 12 -6.220 -5.815 1.00
## 13 -6.219 -5.815 0.99
## 14 -6.219 -5.813 1.00
## 15 -5.301 -4.642 0.67
## 16 -5.183 -4.636 1.00
## 17 -5.181 -4.633 0.99
## 18 -3.605 -2.476 0.96
## 19 -3.603 -2.472 0.97
## 20 -3.601 -2.475 0.96
## 21 -3.597 -2.463 1.00
## 22 -3.596 -2.474 0.97
## 23 -3.593 -2.478 0.96
## 24 -3.589 -2.459 0.98
## 25 -3.581 -2.461 0.96
## 26 -2.066 -1.411 1.00
## 27 -2.066 -1.404 0.98
## 28 -2.065 -1.412 1.00
## 29 -2.037 -1.392 0.94
## 30  0.060  0.466 1.00
## 31  0.063  0.468 1.00
## 32  0.063  0.469 1.00
## 33  0.064  0.469 1.00
## 34  0.064  0.472 0.99
## 35  0.066  0.472 0.99
## 36  0.130  0.544 0.89
## 37  0.159  0.571 0.88
## 38  1.089  1.648 0.96
## 39  1.100  1.647 1.00
## 40  1.110  1.630 0.94
## 41  1.124  1.654 0.92
```

```
## 42  2.686  3.820 1.00
## 43  4.187  4.912 0.86
## 44  4.218  4.872 1.00
## 45  4.270  4.916 0.88
## 46  4.287  4.927 0.86
## 47  6.359  6.770 0.96
## 48  6.439  6.849 0.89
## 49  6.564  6.982 0.78
## 50  7.393  7.958 0.95
## 51  7.395  7.990 0.89
## 52  7.426  7.942 0.87
## 53  7.429  7.931 0.85
## 54  9.064 10.109 0.80
```

CHAPTER VI

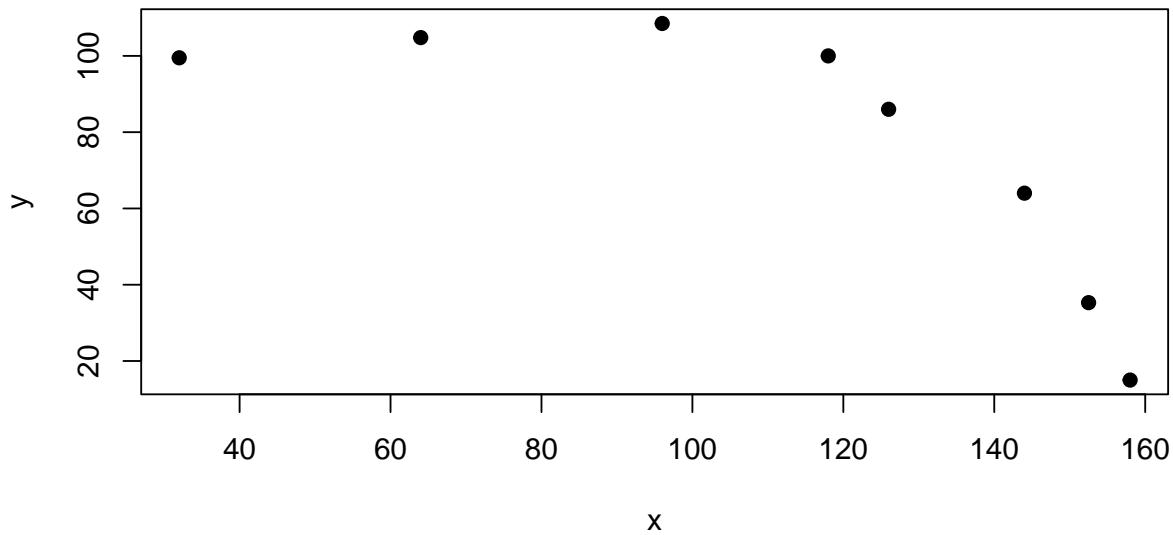
8 *CURVE FITTING*

Curve fitting alias pencocokan kurva adalah suatu metode yang lazim digunakan untuk mendekati suatu data *real* dengan fungsi dengan *error* terkecil. Ada banyak metode untuk melakukannya, seperti pendekatan linear, polinomial, dan *spline cubic*.

8.1 Linear *Curve Fitting* untuk Dua Peubah

Biasa kita kenal dengan regresi linear. Menggunakan `base` dari **R** kita bisa dengan mudah menghitungnya. Sebagai contoh:

```
# ini data contoh
x = c(32,64,96,118,126,144,152.5,158)
y = c(99.5,104.8,108.5,100,86,64,35.3,15)
# Plotnya
plot(x,y,pch=19)
```



Untuk membuat model linear, kita bisa gunakan `base R` via `lm()`. Berikut adalah caranya:

```
# linear regression
model1 = lm(y~x)
summary(model1)
```

```

## 
## Call:
## lm(formula = y ~ x)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
## -33.78 -18.69   3.40  19.27  27.35 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 143.0438   24.8489   5.757  0.0012 **  
## x           -0.5966    0.2090  -2.854  0.0290 *   
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 24.7 on 6 degrees of freedom 
## Multiple R-squared:  0.5759, Adjusted R-squared:  0.5052 
## F-statistic: 8.148 on 1 and 6 DF,  p-value: 0.02901

```

Lantas bagaimana caranya jika kita hendak membuat algoritmanya sendiri?

Misalkan suatu vektor X dan Y hendak dibuat persamaan linear sebagai berikut $f(x) = ax+b$.

$$A = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix}, \quad X = \begin{bmatrix} b \\ a \end{bmatrix}, \quad B = \begin{bmatrix} \sum y_i \\ \sum (x_i y_i) \end{bmatrix}$$

Figure 39: Dasar Regresi Linear

Dari mana operasi matriks tersebut berasal?

Perhatikan bahwa kita menginginkan y dapat ditulis sebagai fungsi dari x .

$$y = f(x) = ax + b$$

Dengan memodifikasi 2 hal berikut ini:

1. Mengalikan $f(x)$ dengan x mengakibatkan $yx = ax^2 + bx$ lalu menjumlahkan semua data yang ada $\sum_i y_i x_i = a \sum_i x_i^2 + b \sum_i x_i$.
2. Menjumlahkan $f(x)$ menjadi $\sum_i y = a \sum_i x_i + nb$, dengan n adalah banyaknya data (panjang vektor x atau y).

Akibatnya kita bisa menuliskannya dalam bentuk operasi matriks berikut ini:

$$\begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$$

Karena kita telah memiliki data X dan Y , artinya untuk mencari $(b, a)^T$ kita cukup melakukan operasi matriks sederhana:

$$A^{-1}b = x$$

Cukup cari invers matriks dan kalikan dengan vektor yang ada.

Pada contoh soal ini, kita bisa menghitung:

```
n = length(x)
sum_xi = sum(x)
sum_xi_xi = sum(x^2)
sum_xi_yi = sum(x*y)
sum_yi = sum(y)

A = matrix(c(n,sum_xi,sum_xi,sum_xi_xi),nrow = 2,byrow = T)
A

##          [,1]      [,2]
## [1,]    8.0    890.5
## [2,]  890.5 113092.2

A_inv = solve(A)
A_inv

##          [,1]      [,2]
## [1,]  1.012031562 -0.00796884054
## [2,] -0.007968841  0.00007158981

b = c(sum_yi,sum_xi_yi)

A_inv %*% b

##          [,1]
## [1,] 143.0437906
## [2,] -0.5965753
```

Terlihat bahwa hasilnya sama dengan *output base R*.

8.1.1 Function di R Dua Peubah

Oke, berbekal algoritma di atas, kita akan buat *function*-nya di **R** dengan *input* berupa dua *vectors* sebagai berikut:

```
tolong_dibantu_linear_fit = function(x,y){
  # hitung elemen-elemen yang dibutuhkan:
  n = length(x)
  sum_xi = sum(x)
  sum_xi_xi = sum(x^2)
  sum_xi_yi = sum(x*y)
  sum_yi = sum(y)
  # bikin persamaan reg linearanya
  A = matrix(c(n,sum_xi,sum_xi,sum_xi_xi),nrow = 2,byrow = T)
  A_inv = solve(A)
  b = c(sum_yi,sum_xi_yi)
  koef = A_inv %*% b
  koef_ = round(koef,3)
  a = koef_[2]
  b = koef_[1]
  # bikin persamaan
  temuan_1 = ifelse(b <= 0,
                     paste0("y = ",a,"x ",b),
                     paste0("y = ",a,"x + ",b))

  # hitung fungsi
  f_y = koef[2]*x + koef[1]
  error = mean(abs(y - f_y))

  # bikin grafik perbandingan
  plot =
    data.frame(x,y,f_y) %>%
    ggplot() +
    geom_point(aes(x,y),
               size = 1,
               color = "darkred") +
    geom_line(aes(x,f_y),
              color = "steelblue",
              alpha = .5) +
    theme_minimal() +
    labs(title = "Perbandingan Antara Titik Data dengan Hasil Curve Fitting",
         subtitle = "Linear Curve Fitting untuk Dua Peubah",
         caption = "Digambar dengan R\nnikanx101.com")

  # tulis output
```

```

output = list("Persamaan regresi linear" = temuan_1,
             "Mean Absolut Error" = error,
             "Grafik" = plot)
return(output)
}

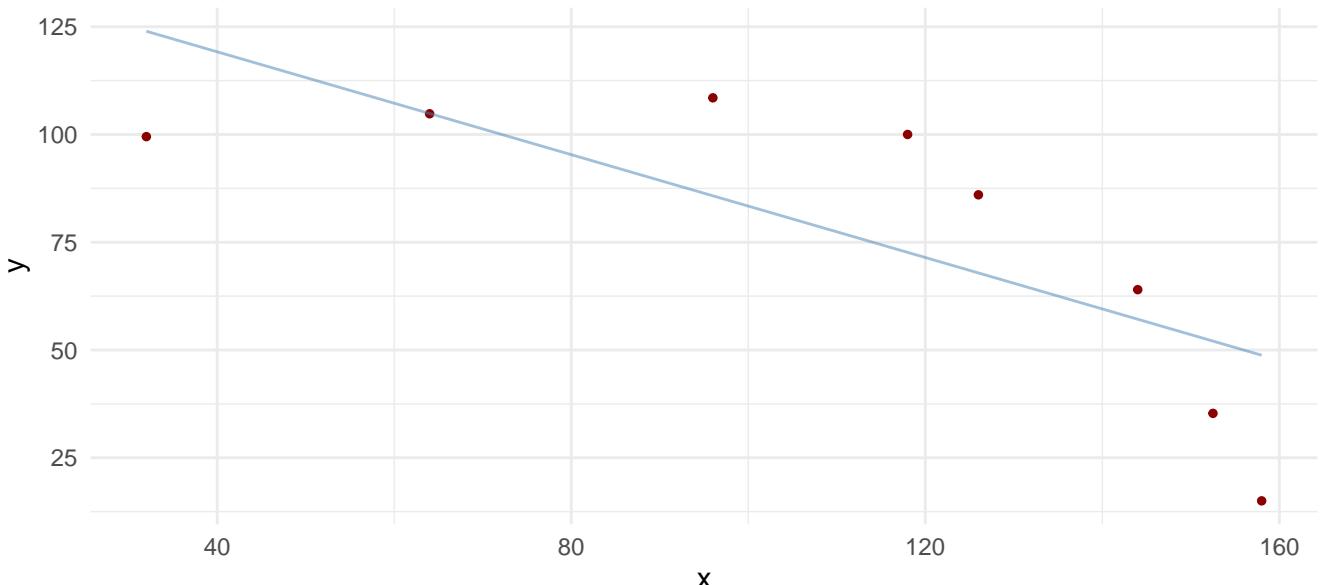
```

Berikut adalah penggunaannya:

```
# contoh penggunaan
tolong_dibantu_linear_fit(x,y)
```

```
## $`Persamaan regresi linear`
## [1] "y = -0.597x + 143.044"
##
## $`Mean Absolut Error`
## [1] 18.76682
##
## $Grafik
```

Perbandingan Antara Titik Data dengan Hasil Curve Fitting
Linear Curve Fitting untuk Dua Peubah



Digambar dengan R
ikanx101.com

Figure 40: Function Regresi Linear 2 Peubah

8.2 Linear *Curve Fitting* untuk Banyak Peubah

Lantas bagaimana caranya melakukan *linear curve fitting* jika masalah yang kita hadapi memiliki banyak peubah? Untuk menjelaskannya saya akan berikan sebuah contoh sederhana: Misalkan kita hendak membuat model sebagai berikut:

$$y = a_0 + a_1x_1 + a_2x_2 + a_3x_3$$

dari data:

```
data = mtcars[c(1,3,4,5)]
data = data %>% rename(y = mpg, x1 = disp, x2 = hp, x3 = drat)
row.names(data) = NULL
data

##      y     x1    x2    x3
## 1 21.0 160.0 110 3.90
## 2 21.0 160.0 110 3.90
## 3 22.8 108.0  93 3.85
## 4 21.4 258.0 110 3.08
## 5 18.7 360.0 175 3.15
## 6 18.1 225.0 105 2.76
## 7 14.3 360.0 245 3.21
## 8 24.4 146.7   62 3.69
## 9 22.8 140.8   95 3.92
## 10 19.2 167.6 123 3.92
## 11 17.8 167.6 123 3.92
## 12 16.4 275.8 180 3.07
## 13 17.3 275.8 180 3.07
## 14 15.2 275.8 180 3.07
## 15 10.4 472.0 205 2.93
## 16 10.4 460.0 215 3.00
## 17 14.7 440.0 230 3.23
## 18 32.4   78.7   66 4.08
## 19 30.4   75.7   52 4.93
## 20 33.9   71.1   65 4.22
## 21 21.5 120.1   97 3.70
## 22 15.5 318.0 150 2.76
## 23 15.2 304.0 150 3.15
## 24 13.3 350.0 245 3.73
## 25 19.2 400.0 175 3.08
## 26 27.3   79.0   66 4.08
## 27 26.0 120.3   91 4.43
## 28 30.4   95.1 113 3.77
```

```
## 29 15.8 351.0 264 4.22
## 30 19.7 145.0 175 3.62
## 31 15.0 301.0 335 3.54
## 32 21.4 121.0 109 4.11
```

Jika kita selesaikan dengan *function base* di **R**, kita bisa melakukannya dengan `lm()` yang sama dengan kasus linear dua peubah.

```
model = lm(y ~ ., data = data)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ ., data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.1225 -1.8454 -0.4456  1.1342  6.4958
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 19.344293   6.370882   3.036  0.00513 **
## x1          -0.019232   0.009371  -2.052  0.04960 *
## x2          -0.031229   0.013345  -2.340  0.02663 *
## x3           2.714975   1.487366   1.825  0.07863 .
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.008 on 28 degrees of freedom
## Multiple R-squared:  0.775, Adjusted R-squared:  0.7509
## F-statistic: 32.15 on 3 and 28 DF,  p-value: 0.00000000328
```

Bagaimana caranya jika kita hendak mengerjakannya sendiri?

Misalkan kita memiliki n buah data dan m buah peubah.

$$X_{m \times n} \beta_{m \times 1} = Y_{m \times 1}$$

Maksudnya bagaimana?

Perhatikan:

$$X = \begin{bmatrix} 1 & x_1 & x_2 & x_3 \\ 1 & x_1 & x_2 & x_3 \\ 1 & x_1 & x_2 & x_3 \\ \vdots & & & \\ 1 & x_1 & x_2 & x_3 \end{bmatrix}$$

$$\beta = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Sudah *clear* kan?

Banyaknya baris X adalah sebanyak n buah data.

Untuk mencari β kita tinggal mencari inverse X^{-1} lalu dikalikan dengan Y .

$$\beta = X^{-1}Y$$

Tapi karena X bukanlah *square matrix* kita perlu me-*rework* menjadi berikut ini:

$$X^T X \beta = X^T Y$$

Perlu diperhatikan bahwa bentuk di atas disebut dengan **Normal Equations**.

Akibatnya:

$$\beta = (X^T X)^{-1} X^T Y$$

Maka untuk menyelesaikannya di **R**:

```
# hitung semua elemen
X =
  data %>%
  mutate(x0 = 1) %>%
  relocate(x0, .before = x1) %>%
  select(contains("x")) %>%
  as.matrix()
X_T = t(X)
Y = data %>% select("y") %>% as.matrix()

# solve
beta = solve(X_T %*% X) %*% X_T %*% Y
beta

##          y
## x0  19.34429256
## x1 -0.01923223
## x2 -0.03122932
## x3  2.71497521
```

Bagaimana hasilnya? Sama kan dengan hasil dari **base R**?

8.2.1 Function di R Banyak Peubah

Dari proses di atas, kita akan buat *function* nya. *Input*-nya akan saya definisikan sebagai *data frame* saja dengan nama-nama variabel adalah Y, X_i dengan $i \in \{0, 1, \dots, m\}$.

```
tolong_dibantu_linear_fit_banyak = function(dataframe){
  # bikin variabel x0
  dataframe =
    dataframe %>%
    mutate(x0 = 1) %>%
    relocate(x0,.before = x1)

  # hitung semua elemen
  X =
    dataframe %>%
    mutate(x0 = 1) %>%
    relocate(x0,.before = x1) %>%
    select(contains("x")) %>%
    as.matrix()
  X_T = t(X)
  Y = dataframe %>% select("y") %>% as.matrix()

  # solve
  beta = solve(X_T %*% X) %*% X_T %*% Y
  beta = beta %>% as.numeric()
  m = ncol(X)
  a = paste0("a",0:(m-1))
  power_x = paste0("x^",0:(m-1))
  temp = data.frame(a,
                     koef = beta %>% round(3),
                     x = power_x)

  # output
  output = list("Persamaan linearnya: " = temp)
  return(output)
}
```

Sekarang kita akan coba untuk contoh data sebelumnya:

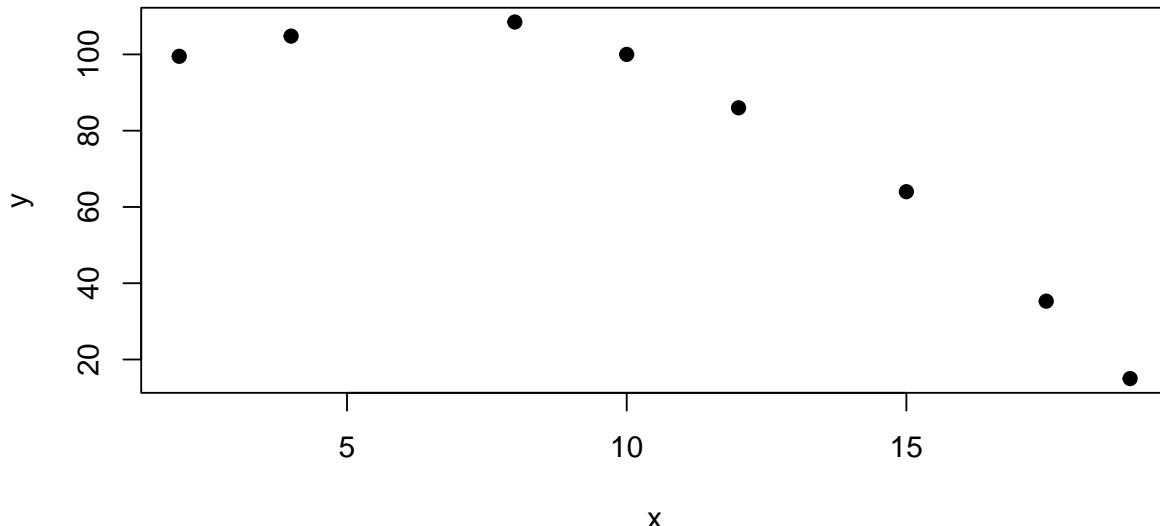
```
tolong_dibantu_linear_fit_banyak(data)
```

```
## $`Persamaan linearnya: ` 
##   a   koef   x
## 1 a0 19.344 x^0
## 2 a1 -0.019 x^1
## 3 a2 -0.031 x^2
## 4 a3  2.715 x^3
```

8.3 Polinomial *Curve Fitting* Dua Peubah

Kita kembali ke contoh awal sebagai berikut:

```
# ini data contoh
x = c(2,4,8,10,12,15,17.5,19)
y = c(99.5,104.8,108.5,100,86,64,35.3,15)
# Plotnya
plot(x,y,pch=19)
```



Selain *linear curve fitting*, kita bisa melakukan pendekatan polinom. Misalnya tuliskan sebagai berikut:

$$y = f(x) = c_0 + c_1x + c_2x^2 + \dots + c_nx^n$$

Proses pengerjaannya mirip dengan kasus ***curve fitting*** banyak peubah. Perbedaannya adalah kita mau pakai berapa derajat polinomialnya? Karena itu akan mendefinisikan berapa banyak kolom matriks yang hendak dibuat.

Misalkan saya hendak membuat polinom order 5, dengan **base R** kita cukup melakukan:

```
banyak_order = 5

model = lm(y ~ poly(x,banyak_order))

summary(model)

## 
## Call:
## lm(formula = y ~ poly(x, banyak_order))
## 
## Residuals:
##      1       2       3       4       5       6       7       8 
## 0.10780 -0.31078  0.63020  0.00232 -1.20145  1.47720 -1.08172  0.37643 
## 
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)                76.6375   0.5832 131.399 0.0000579 ***
## poly(x, banyak_order)1 -81.6586   1.6497 -49.500 0.000408 ***  
## poly(x, banyak_order)2 -44.1626   1.6497 -26.771 0.001392 **  
## poly(x, banyak_order)3  1.0331    1.6497   0.626  0.595087    
## poly(x, banyak_order)4  0.1664    1.6497   0.101  0.928842    
## poly(x, banyak_order)5 -2.6788   1.6497  -1.624  0.245889    
## ---                        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 1.65 on 2 degrees of freedom
## Multiple R-squared:  0.9994, Adjusted R-squared:  0.9978 
## F-statistic: 634 on 5 and 2 DF,  p-value: 0.001576
```

Bagaimana jika kita hendak menyelesaikannya sendiri? Kita akan buat operasi matriksnya terlebih dahulu. Misalkan saya definisikan:

$$A = \begin{bmatrix} 1 & x & x^2 \\ 1 & x & x^2 \\ 1 & x & x^2 \\ \vdots & & \\ 1 & x & x^2 \end{bmatrix}$$

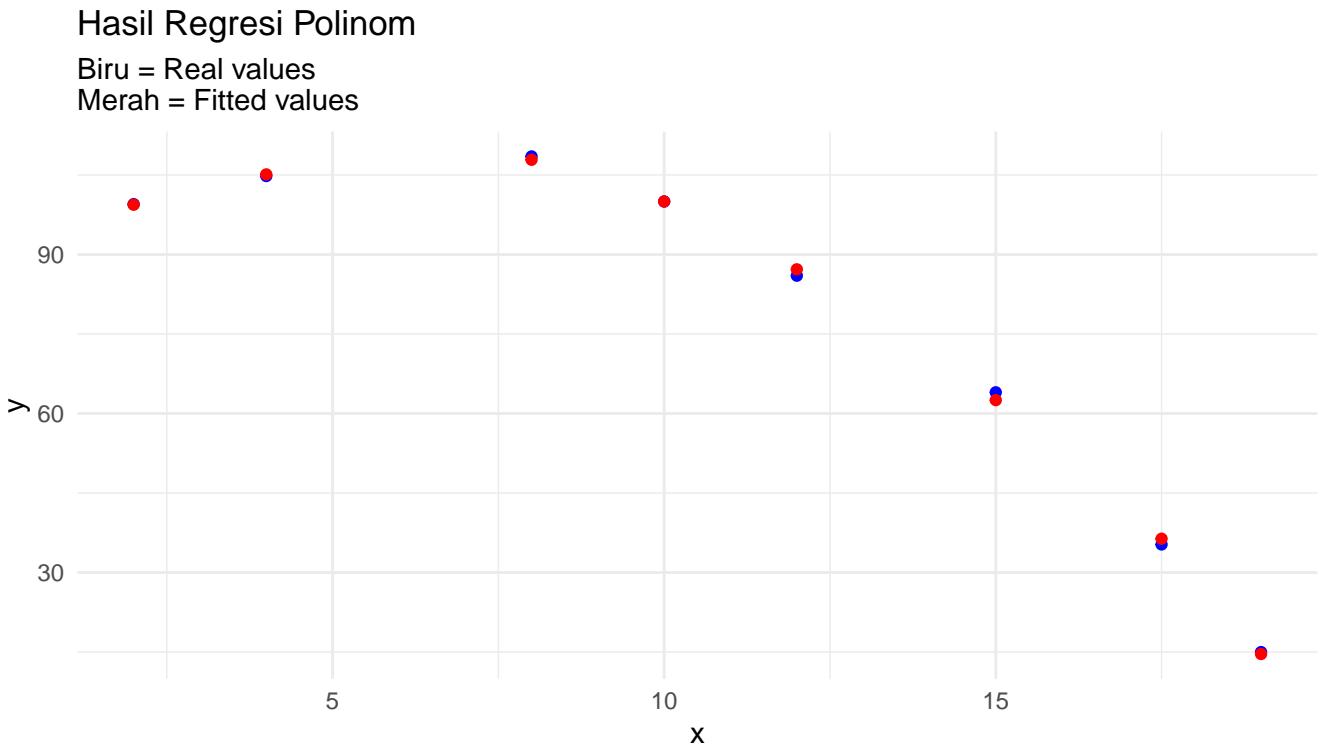


Figure 41: Evaluasi Polinom Base R

Dengan banyaknya baris adalah banyaknya data yang kita miliki.

Lalu saya definisikan C sebagai vektor berikut:

$$C = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

Tapi karena A belum tentu berupa *square matrix* kita perlu me-rework menjadi berikut ini:

$$C = (A^T A)^{-1} A^T Y$$

Berikut penyelesaiannya di R:

```
# definisi matriks
A = matrix(ncol = banyak_order+1,
           nrow = length(x))

# untuk isi c0 hingga cn
for(i in 1:(banyak_order+1)){
  A[,i] = x^(i-1)
}
```

```
# hitung transpose A
A_T = t(A)

# solve
C = solve(A_T %*% A) %*% A_T %*% y
C

## [,1]
## [1,] 101.8720591947
## [2,] -5.7323303826
## [3,] 3.0164272626
## [4,] -0.4284301182
## [5,] 0.0225171681
## [6,] -0.0004293536
```

Terlihat beda dengan hasil dari **base R** tapi kita coba cek perbandingan grafisnya:

Hasil Regresi Polinom

Biru = Real values

Merah = Fitted values

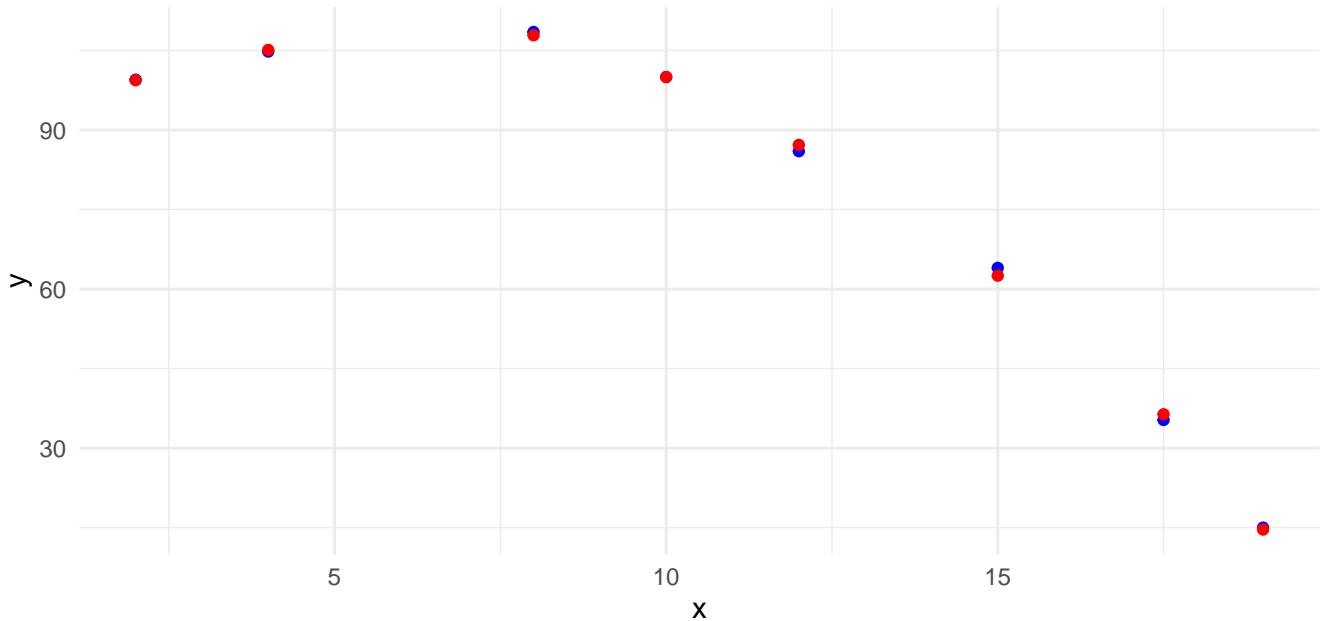


Figure 42: Evaluasi Polinom Operasi Matriks

Hasilnya juga sangat bagus *yah*.

8.3.1 Function di R Polinomial

Oke, sekarang saatnya kita membuat *function* di **R**. Kita akan buat *input*-nya berupa dua buah *vectors* sama dengan regresi linear sebelumnya:

```
tolong_dibantu_poli_fit = function(x,y,banyak_order){
  # definisi matriks
  A = matrix(ncol = banyak_order+1,
             nrow = length(x))

  # untuk isi c0 hingga cn
  for(i in 1:(banyak_order+1)){
    A[,i] = x^(i-1)
  }

  # hitung transpose A
  A_T = t(A)

  # solve
  C = solve(A_T %*% A) %*% A_T %*% y
  Values = as.numeric(C) %>% round(4)

  # output data
  hasil = data.frame(C = paste0("C",0:(banyak_order)),
                      Values)

  # bikin plot
  # dimulai dari vektor nol
  f = rep(0,nrow(hasil))
  # iterasi
  for(i in 1:(banyak_order + 1)){
    temp = x^(i-1) * C[i]
    f = f + temp
  }

  plot =
  data.frame(x,y,f) %>%
  ggplot() +
  geom_point(aes(x,y),color = "blue") +
  geom_point(aes(x,f), color = "red") +
  theme_minimal() +
  labs(title = "Hasil Regresi Polinom",
       subtitle = "Biru = Real values\nMerah = Fitted values")

  # menghitung MAE
```

```
error = mean(abs(f - y))

# membuat output
output = list("Konstanta" = hasil,
              "MAE" = error,
              "Grafik" = plot)
return(output)
}
```

Sekarang kita akan coba untuk kasus ini:

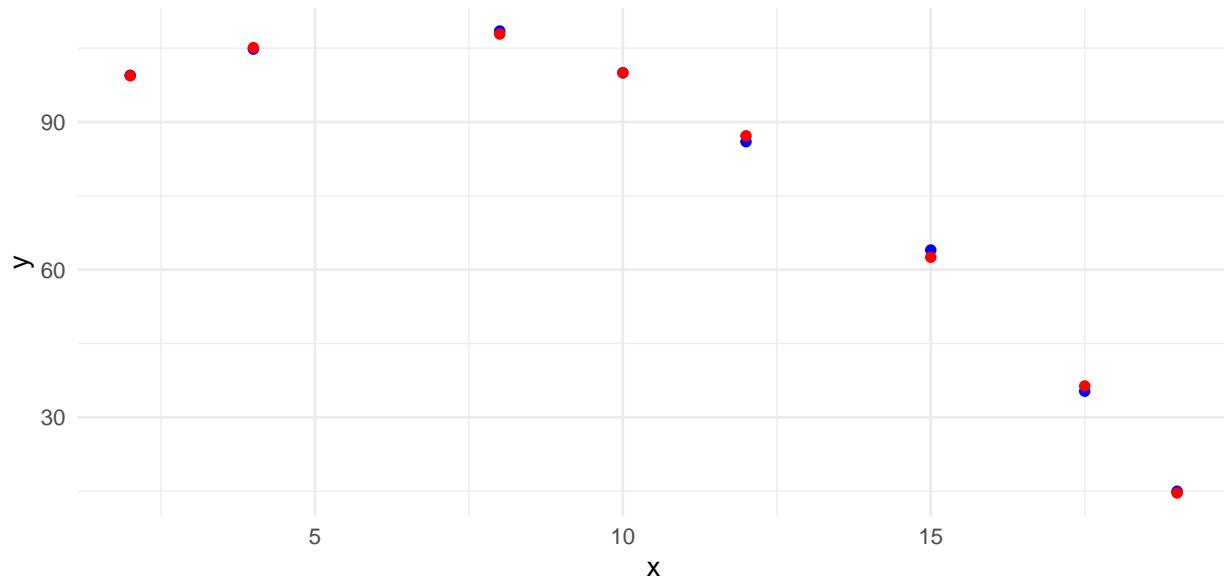
```
tolong_dibantu_poli_fit(x,y,5)
```

```
## $Konstanta
##      C    Values
## 1 C0 101.8721
## 2 C1 -5.7323
## 3 C2  3.0164
## 4 C3 -0.4284
## 5 C4  0.0225
## 6 C5 -0.0004
##
## $MAE
## [1] 0.648487
##
## $Grafik
```

Hasil Regresi Polinom

Biru = Real values

Merah = Fitted values



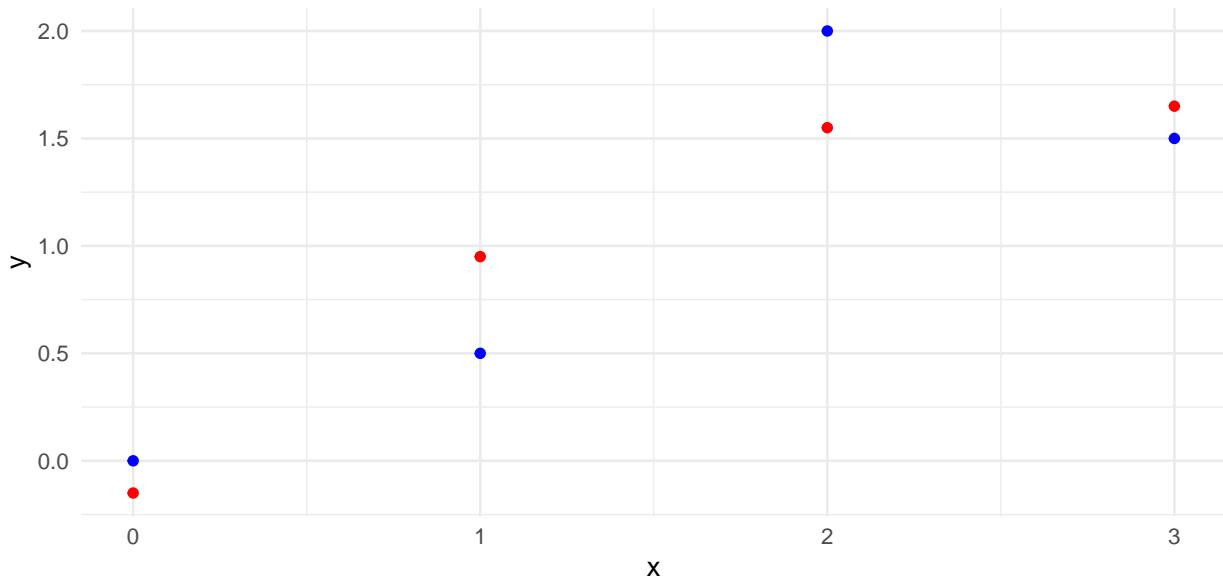
Sekarang kita akan coba untuk kasus lain:

```
x = c(0,1,2,3)
y = c(0,.5,2,1.5)
tolong_dibantu_poli_fit(x,y,2)
```

```
## $Konstanta
##      C Values
## 1  C0   -0.15
## 2  C1    1.35
## 3  C2   -0.25
##
## $MAE
## [1] 0.3
##
## $Grafik
```

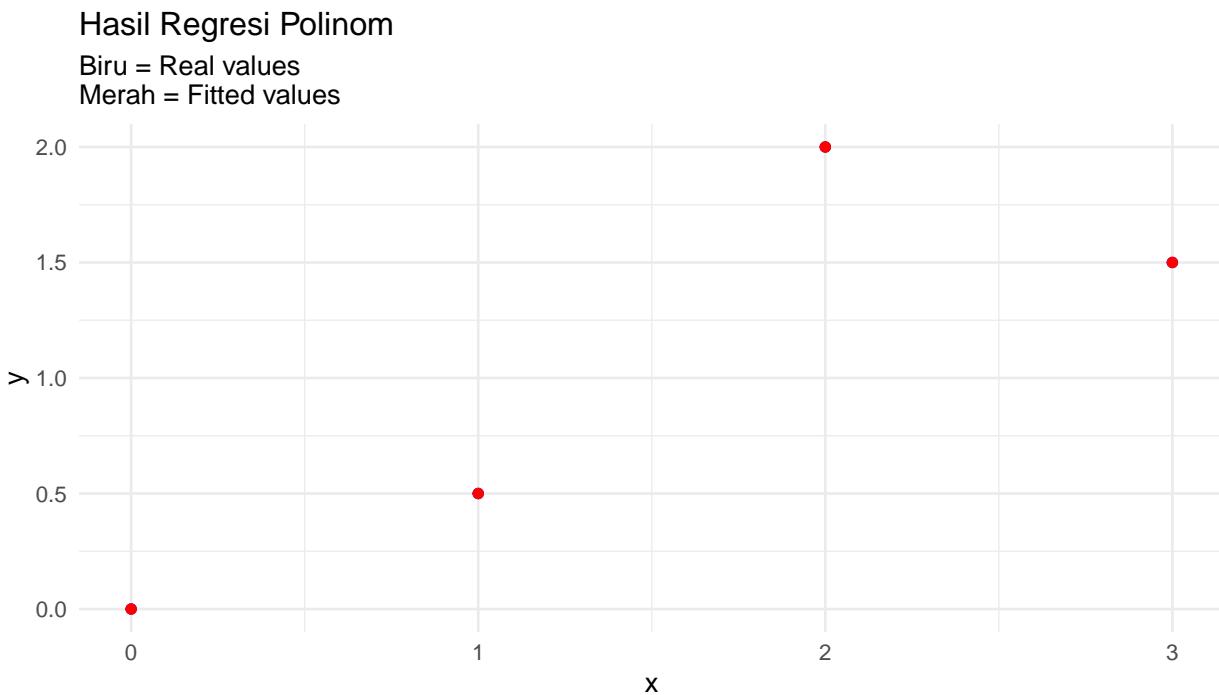
Hasil Regresi Polinom

Biru = Real values
Merah = Fitted values



```
tolong_dibantu_poli_fit(x,y,3)
```

```
## $Konstanta
##      C Values
## 1  C0    0.0
## 2  C1   -1.0
## 3  C2    2.0
## 4  C3   -0.5
##
## $MAE
## [1] 0.0000000000009345996
##
## $Grafik
```



8.3.2 Modifikasi Polinom Banyak Peubah

Kita bisa memodifikasi *function* di atas untuk menyelesaikan permasalahan *curve fitting* polinomial untuk banyak peubah dengan cara mengubah matriks yang berkaitan dengan order polinomnya.

8.4 Cubic Spline Curve Fitting

Definisi:

Cubic Spline Interpolant

Suppose that $\{(x_k, y_k)\}_{k=0}^N$ are $N + 1$ points, where $a = x_0 < x_1 < \dots < x_N = b$. The function $S(x)$ is called a *cubic spline* if there exist N cubic polynomials $S_k(x)$ with coefficients $s_{k,0}, s_{k,1}, s_{k,2}$ and $s_{k,3}$ that satisfy the properties :

$$\text{I. } S(x) = S_k(x) = s_{k,0} + s_{k,1}(x - x_k) + s_{k,2}(x - x_k)^2 + s_{k,3}(x - x_k)^3$$

for $x \in [x_k, x_{k+1}]$ and $k = 0, 1, \dots, N-1$.

$$\text{II. } S(x_k) = y_k \quad \text{for } k = 0, 1, 2, \dots, N.$$

$$\text{III. } S_k(x_{k+1}) = S_{k+1}(x_{k+1}) \quad \text{for } k = 0, 1, 2, \dots, N-2.$$

$$\text{IV. } S'_k(x_{k+1}) = S'_{k+1}(x_{k+1}) \quad \text{for } k = 0, 1, 2, \dots, N-2.$$

$$\text{V. } S''_k(x_{k+1}) = S''_{k+1}(x_{k+1}) \quad \text{for } k = 0, 1, 2, \dots, N-2.$$

Figure 43: Definisi Cubic Spline

Berikut adalah contohnya dengan **base** di **R**:

```
X = c(0, 1, 2, 3)
Y = c(0, 0.5, 2.0, 1.5)
f_x = spline(X, Y)
```

Ini adalah grafiknya:

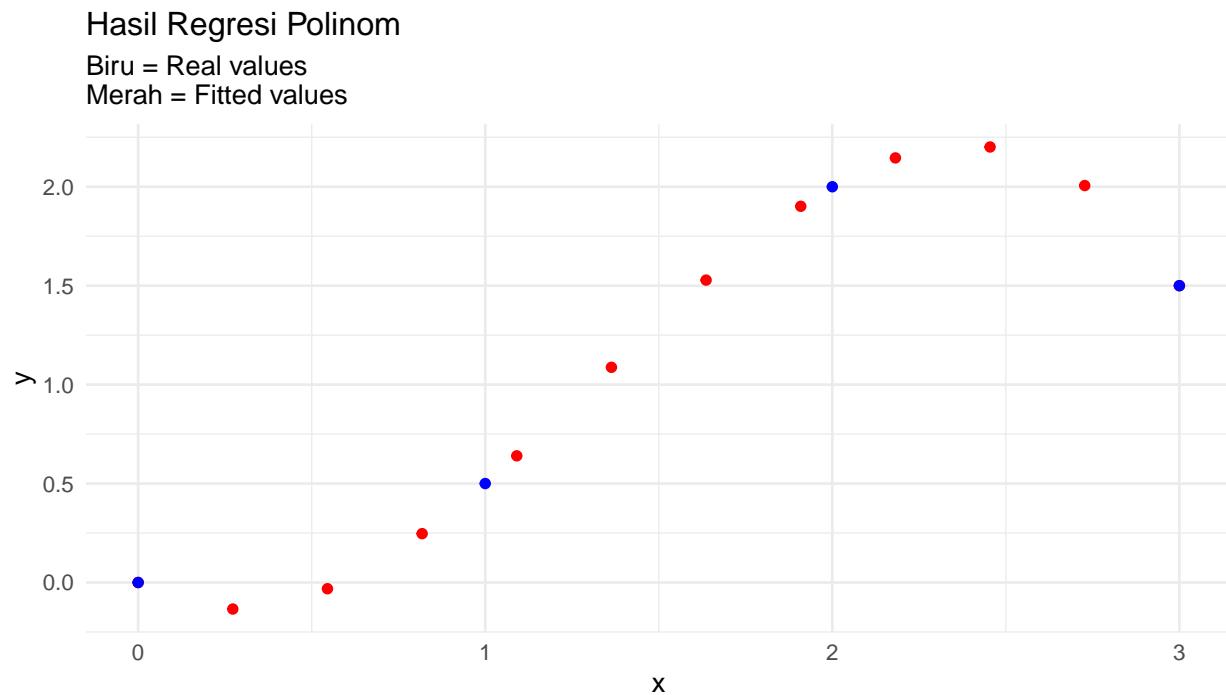


Figure 44: Hasil Spline Base R

END