# Komputasi dengan Julia

Novalio Daratha

5 Nopember 2023

# Bab 1

# Bilangan Kompleks

Bahasa Julia memiliki tipe yang sudah didefinisikan untuk bilangan kompleks. Konstanta global $im$ dikaitkan dengan bilangan kompleks $i$, yang mewakili nilai akar kuadrat dari $-1$, $\sqrt{-1}$. (Pemakaian simbol $i$ dan $j$ dilarang karena kedua simbol tersebut sering digunakan sebagai nama peubah indeks.) Gambar 1.1 menunjukkan bagaimana memberikan nilai kompleks pada sebuah peubah bernama $S$.

Bilangan kompleks bisa dikalikan dengan bilangan lainnya, ditambah, dikurangi, dikali atau dibagi. Hal ini juga ditunjukkan pada Gambar 1.1.

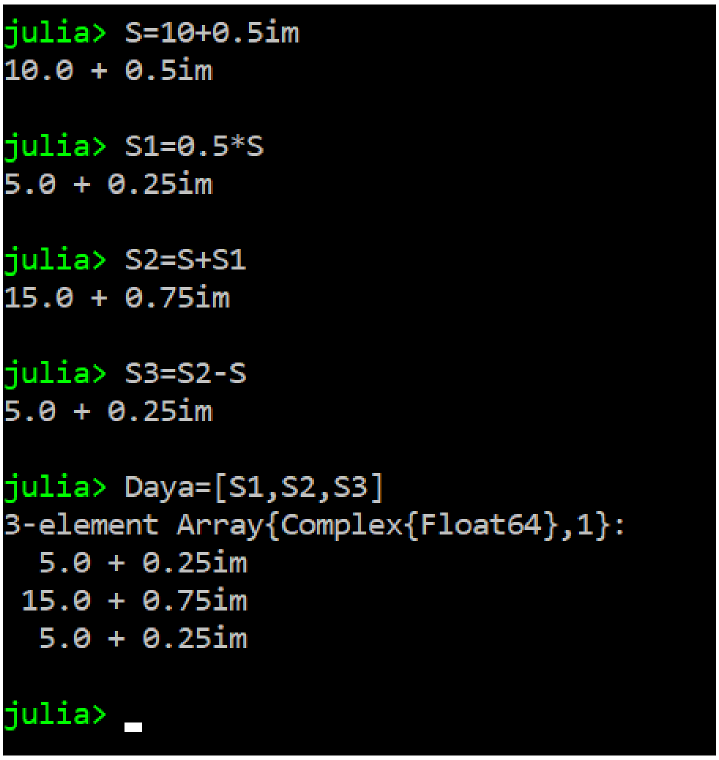
Sudut dan besar sebuah bilangan kompleks dapat dengan mudah menggunakan perintah **angle** dan **abs**. Gambar 1.2 menunjukan hal-hal tersebut.

Untuk mengubah bilangan kompleks dari bentuk polar ke rektangular juga cukup sederhana. Langkah pertama adalah menentukan nilai magnitude dan sudut (dalam radian). Kemudian bilangan kompleks diperoleh dengan menggunakan fungsi **Complex**. Contoh konversi ini dapat dilihat pada Gambar 1.3.

Untuk mendapatkan conjugate sebuah bilangan complex, Anda dapat menggunakan fungsi **conj**. Fungsi **real** dan **imag** digunakan untuk mendapatkan nilai real dan imaginary sebuah bilangan kompleks. Anda bisa melihat contohnya di Gambar 1.4.

## 1.1 Sistem listrik tiga fasa

PLN menggunakan sistem tiga fasa untuk membangkitkan dan menyebarkan energi listrik. Secara matematis, tegangan listrik PLN dapat dimodelkan




Gambar 1.1: Memberikan nilai kompleks kepada sebuah peubah.

dengan vektor bilangan kompleks berikut ini.

$$\mathbf{V}\begin{bmatrix}\mathbf{V}_a\\ \mathbf{V}_b\\ \mathbf{V}_c\end{bmatrix} = \begin{bmatrix}220\angle 0^o\\ 220\angle -120^o\\ 220\angle 220^o\end{bmatrix} = \begin{bmatrix}220\angle 0\\ 220\angle -2.0944\\ 220\angle 2.0944\end{bmatrix} \tag{1.1}$$

Tegangan tiga fasa tersebut dapat dihitung dengan Julia seperti ditunjukkan pada Gambar 1.5.

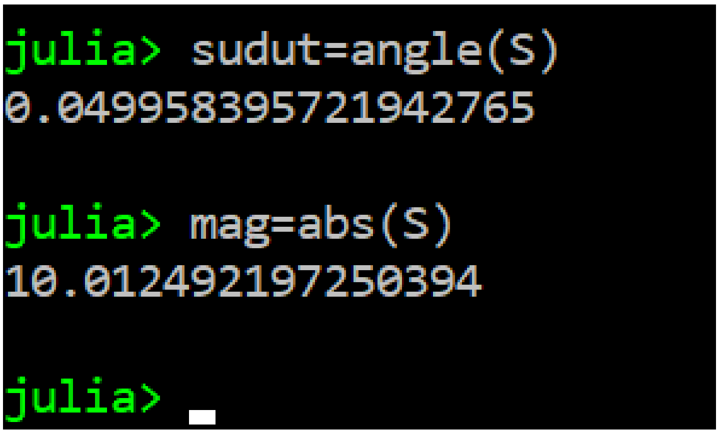
Untuk beban yang memerlukan daya sebesar

$$\mathbf{S} = \begin{bmatrix}\mathbf{S}_a\\ \mathbf{S}_b\\ \mathbf{S}_c\end{bmatrix} = \begin{bmatrix}100+0j\\ 200+100j\\ 0+150j\end{bmatrix}, \tag{1.2}$$

Daya tersebut dapat direpresentasikan dalam Julia seperti ditunjukkan dalam Gambar 1.6.

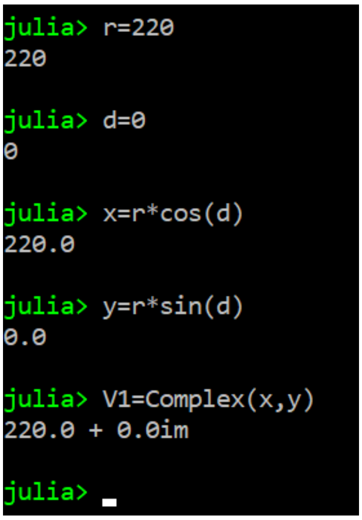
PLN harus menyediakan arus sebesar konjugate dari hasil bagi daya dengan tegangan.

$$\mathbf{I} = (\mathbf{S}/\mathbf{V})^* \tag{1.3}$$



```
julia> sudut=angle(S)
0.049958395721942765

julia> mag=abs(S)
10.012492197250394

julia>
```

Gambar 1.2: Sudut dan magnitude bilangan kompleks.



```
julia> r=220
220

julia> d=0
0

julia> x=r*cos(d)
220.0

julia> y=r*sin(d)
0.0

julia> V1=Complex(x,y)
220.0 + 0.0im

julia>
```

Gambar 1.3: Mengubah bilangan kompleks dari bentuk polar ke rektangular.

```
julia> S
10.0 + 0.5im

julia> conj(S)
10.0 - 0.5im

julia> real(S)
10.0

julia> imag(S)
0.5

julia>
```

Gambar 1.4: Contoh cara mendapatkan konjugate, bagian real dan bagian imajiner sebuah bilangan kompleks.

```
julia> Va=complex(220*cos(0),220*sin(0))
220.0 + 0.0im

julia> Vb=complex(220*cos(-2.0944),220*sin(-2.0944))
-110.00093311810099 - 190.52505009354303im

julia> Vc=complex(220*cos(2.0944),220*sin(2.0944))
-110.00093311810099 + 190.52505009354303im

julia> V=[Va,Vb,Vc]
3-element Array{Complex{Float64},1}:
               220.0 + 0.0im
 -110.00093311810099 - 190.52505009354303im
 -110.00093311810099 + 190.52505009354303im
```

Gambar 1.5: Tegangan tiga fasa yang disediakan PLN.

```
julia> Sa=complex(100,0)
100 + 0im

julia> Sb=complex(200,100)
200 + 100im

julia> Sc=complex(0,150)
0 + 150im

julia> S=[Sa,Sb,Sc]
3-element Array{Complex{Int64},1}:
 100 + 0im
 200 + 100im
   0 + 150im
```

Gambar 1.6: Cara merepresentasikan daya tiga fasa dalam Julia.

```
julia> S/V
3×3 Array{Complex{Float64},2}:
 0.151515-0.0im       -0.0757582+0.131216im  -0.0757582-0.131216im
  0.30303+0.151515im   -0.282732+0.186673im  -0.0203008-0.338189im
      0.0+0.227273im   -0.196823-0.113637im    0.196823-0.113637im

julia> S./V
3-element Array{Complex{Float64},1}:
 0.45454545454545453 + 0.0im
 -0.8481961081193077 + 0.5600189402251757im
  0.5904701965708977 - 0.3409119828040319im

julia>
```

Gambar 1.7: Cara menghitung arus beban apabila daya dan tegangan sudah diketahui. Perhatikan perbedaan simbol "/" dan "./" .

Perhitungan arus tersebut dapat dilakukan dengan proses pembagian per elemen yang menggunakan simbol "./". Simbol "/" tidak akan menghasilkan nilai yang kita maksud karena simbol "/" akan menghasilkan sebuah matriks $3 \times 3$ karena baik tegangan dan daya adalah matriks dengan ukuran $3 \times 1$. Hal ini ditunjukkan pada Gambar 1.7.

# Bab 2

# Komputasi iteratif (berulang)

Dalam banyak situasi, kita perlu mengulang sederetan perhitungan berkali-kali. Komputer adalah alat yang sesuai untuk pekerjaan seperti itu. Dalam Bab ini, kita akan menggunakan struktur **for ... in ... end** untuk melakukan komputasi iteratif (berulang).

## 2.1 Barisan dan Deret

Barisan dan deret adalah contoh hal yang berulang.

## 2.2 Barisan

Marilah kita cetak 10 bilangan bulat positif pertama.

$$B_1 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.$$

Barisan $B_1$ tersebut dapat direpresentasikan dalam Julia dengan menggunakan perintah **range** seperti yang ditunjukan pada Gambar 2.1.

```
julia> range(1,stop=10)
1:10

julia> for i in range(1,stop=10)
       print(i,",")
       end
1,2,3,4,5,6,7,8,9,10,
julia>
```

Gambar 2.1: Barisan sepulu bilangan positif bulat pertama.

Kemudian, marilah kita cetak sepuluh bilangan genap positif pertama.

$$B_2 = 2, 4, 6, 8, 10, 12, 14, 16, 18, 20.$$

Barisan $B_2$ juga dapat kita representasikan dengan perintah **range** seperti yang ditunjukkan pada Gambar.

```
julia> range(2,step=2,length=10)
2:2:20

julia> for i in range(2,step=2,length=10)
       print(i,",")
       end
2,4,6,8,10,12,14,16,18,20,
julia>
```

Gambar 2.2: Barisan sepuluh bilangan genap positif pertama.

Barisan $B_1$ dan $B_2$ adalah barisan yang sederhana. Bagaimana dengan barisan yang sedikit rumit yaitu barisan fibonaci berikut ini:

$$B_3 = 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.$$

Dalam hal ini, kita perlu mengingat kembali aturan dasar barisan fibonaci, yaitu

$$fibonaci(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ f(n-1) + f(n-2) & n > 2 \end{cases}$$

Fungsi $fibonaci$ dapat dibuat dalam julia dengan menggunakan konsep **function ... end** dan **if ... elseif ... end** seperti ditunjukkan pada Gambar 2.3.

Setelah fungsi fibonaci terdefinisi dalam Julia, kita bisa membuat barisan $B_3$ seperti yang ditunjukkan pada Gambar 2.4.

## 2.3 Deret

Deret adalah jumlah dari elemen-elemen sebuah barisan. Misalnya, Jumlah sepuluh bilangan genap yang lebih besar dari 10 adalah

$$D_1 = \sum_{n=6}^{15} 2n = 12 + 14 + 16 + ... + 30.$$

```
julia> function fibonaci(n)
       if n==1
        return 1
       elseif n==2
        return 1
       elseif n>2
        return fibonaci(n-1)+fibonaci(n-2)
        end
       end
fibonaci (generic function with 1 method)
```

Gambar 2.3: Fungsi fibonaci.

```
julia> for i in range(1,stop=10)
       print(fibonaci(i),",")
       end
1,1,2,3,5,8,13,21,34,55,
julia>
```

Gambar 2.4: Barisan fibonaci yang dihitung menggunakan fungsi yang didefinisikan sebelumnya di Gambar 2.3

```
julia> hasil=0
0

julia> for i in range(12,step=2,length=10)
        global hasil=hasil+i
        end

julia> print(hasil)
210
julia>
```

Gambar 2.5: Deret dapat dihitung dengan menggunakan Julia dan konsep perulangan dengan menggunakan **for ... in ... end**.

Deret $D_1$ dapat dihitung dengan Julia seperti ditunjukkan pada Gambar 2.5.

Kemudian, kita juga bisa menghitung deret fibonaci yang merupakan jumlah n bilangan fibonaci pertama.

$$D_2[n] = 1 + 1 + 2 + 3 + 5 + 8 + ... + fibonaci(n)$$

Deret $D_2$ dapat juga dihitung dengan Julia seperti ditunjukkan pada gam-

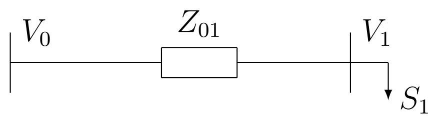
```
julia> function dfibo(n)
         hasil=0
         for i in range(1,length=n)
           hasil=fibonaci(i)+hasil
         end
         return hasil
       end
dfibo (generic function with 1 method)

julia> dfibo(10)
143
```

Gambar 2.6: Deret fibonaci.

bar

## 2.4 Aliran Daya Listrik

Dalam analisis sistem tenaga listrik, ada kajian yang disebut kajian aliran daya. Di bagian ini, kita akan mencoba memecahkan soal aliran daya yang cukup sederhana. Perhatikan sebuah sistem sederhana yang ditunjukkan pada Gambar 2.7. Jika diketahui $V_0 = 220 + 0j\ V$, $Z_{01} = 0.1 + 0.01j\Omega$ dan $S_1 = 100 + 50j$ VA, carilah tegangan $V_1$.




Gambar 2.7: Sistem tenaga listrik sederhana

Untuk mencari besar $V_1$, langkah-langkah yang dapat dilakukan adalah:

1. Mulai dengan $i = 0$.
2. Misalkan $V_1^0 = V_0$.
3. Hitung arus dengan cara $I_1^0 = \left(\frac{S_1}{V_1}\right)^*$.
4. mulai iterasi selanjutnya $i = i + 1$
5. Ganti nilai tegangan $V_1$ dengan nilai baru menggunakan $V_1^i = V_0 - I_1^{i-1} Z_{01}$

```
V0=220+0im
V1=V0 # Nilai dugaan awal
Z01=0.1+0.01im
S1=100+50im
error=1
epsilon=1e-5
iterasi=0
while error>epsilon
    I1=conj(S1/V1)
    V1last=V1
    V1=V0-I1*Z01
    error=abs(V1-V1last)
    iterasi=iterasi+1
end
print("Jumlah iterasi\t=", iterasi)
print("\nKesalahan\t=",error)
print("\nV1\t\t=",V1)
```

```
Jumlah iterasi  =3
Kesalahan       =2.754317824382332e-9
V1              =219.95226086543937 + 0.018181818181818184im
```

Gambar 2.8: Iterasi aliran daya listrik sederhana

6. Hitung arus dengan cara

$$I_1^i = \left(\frac{S_1}{V_1^i}\right)^*$$

.

7. Apakah syarat berikut terpenuhi?

$$\left\|V_1^i - V_1^{i-1})\right\| \leq \epsilon = 10^{-5}$$

   a) Jika Ya, berhenti karena jawaban sudah diperoleh.
   b) Jika Tidak, ulangi langkah-langkah 4-7.

Langkah-langkah 1-7 adalah contoh algoritma yang memecahkan sebuah permasalah secara iteratif. Implementasi algoritma tersebut dalam Julia ditunjukkan pada Gambar 2.8.

# Bab 3

# Algoritma Genetika

Untuk memahami Bab ini, anda diharapkan sudah memahami algoritma genetika dengan cukup baik. Metode ini cukup mudah dipelajari dan banyak sumber di internet yang membahasnya.

## 3.1 Permainan Kartu

Ada sepuluh kartu yang diberi nomor 1 s/d 10. Kartu-kartu tersebut dibagi menjadi dua kelompok, A dan B. Kemudian, jumlah angka untuk masing-masing kelompok dihitung. Carilah cara pembagian yang meminimalkan selisih jumlah angka di kelompok A dan jumlah angka di kelompok B.

### Encoding

Dalam pemanfaatan algoritma genetika, tahap encoding adalah tahap yang paling penting. Output dari tahap ini adalah struktur data yang cocok untuk masalah yang ingin dipecahkan dengan algoritma genetika.

Dalam hal ini sebuah gen adalah sebuah string yang memiliki 10 karakter karena kita memiliki 10 buah kartu. Setiap karakter dalam gen tersebut memiliki informasi yang berisi "A" ("B") jika kartu tersebut termasuk dalam kelompok "A" ("B"). Dalam gen tersebut, kartu ke-$i$ diwakili oleh karakter ke-$i$. Angka yang dimiliki oleh kartu ke-$i$ disimpan pada elemen ke-$i$ sebuah list bernama "w".

Sebagai contoh, untuk permainan kartu yang kita bahas di Sub bab ini, gen akan kita simpan dengan data berjenis **string**. Kemudian, setiap individu memiliki satu gen, satu sumA, satu sumB dan satu nilai fitness. Peubah sumA (sumB) adalah jumlah angka dari seluruh kartu yang ada di kelompok A (B). Nilai fitness didefinisikan sebagai berikut

$$fitness = |sumA - sumB|\,,$$

dimana, | | menunjukkan nilai absolut. Kita mencari gen yang memiliki fitness sekecil mungkin atau sama dengan nol.

Encoding dapat dilakukan dalam Julia dengan memanfaatkan konsep **DataFrames**, **string** dan **rand** seperti ditunjukkan pada Gambar 3.1.

```
using DataFrames
s=""
for i in rand(["A","B"],10)
    s=string(s,i)
end
print(s)
populasi=DataFrame(gen=s,sumA=0,sumB=10,fitness=10)
w=[1,2,3,4,5,6,7,8,9,10]
```

Gambar 3.1: Encoding permainan kartu untuk algoritma genetika. Sebuah gen adalah sebuah string yang memiliki 10 karakter karena kita memiliki 10 buah kartu. Setiap karakter dalam gen tersebut memiliki informasi yang berisi "A" ("B") jika kartu tersebut termasuk dalam kelompok "A" ("B"). Dalam gen tersebut, kartu ke-$i$ diwakili oleh karakter ke-$i$. Angka yang dimiliki oleh kartu ke-$i$ disimpan pada elemen ke-$i$ sebuah list bernama "w".

## Penghitungan nilai fitness

Untuk setiap individu, berdasarkan gen yang dimilikinya, kita perlu menghitung fitnessnya. Oleh karena itu, kita perlu mendefinisikan fungsi yang sesuai kebutuhan tersebut dalam julia. Salah satu implementasinya ditunjukkan dalam Gambar 3.2.

## Gen dengan Nilai Acak

Algoritma genetika memerlukan kemamputan untuk menghasilkan gen secara acak. Sesuai dengan encoding yang telah dipilih (lihat Gambar 3.1). Kita dapat mendefinisikan sebuah fungsi yang menghasilkan gen secara acak seperti ditunjukkan pada Gambar 3.3

```
function fitness(individu,w)
    s=individu.gen
    #@show s
    sumA=0
    sumB=0
    for (i,j) in zip(s,w)
        #@show i,j
        if i=='A'
            sumA=sumA+j
        elseif i=='B'
            sumB=sumB+j
        end
    end
    #@show sumA,sumB
    individu.sumA=sumA
    individu.sumB=sumB
    individu.fitness=abs(sumA-sumB)
    return individu
end

```

Gambar 3.2: Perhitungan nilai fitness sebuah gen.

## Populasi Acak

Algoritma Genetika juga membutuhkan fungsi untuk menghasilkan populasi dengan jumlah individu tertentu dengan gen acak. Hal ini dapat diimplementasikan dengan sebuah fungsi yang ditunjukkan pada Gambar 3.4.

Fungsi untuk mencari populasi yang terdiri dari individu yang memiliki gen acak. Populasi disimpan dalam sebuah DataFrame. Jumlah individu dalam populasi sama dengan jumlah baris dalam dataframe yang dipakai. Kolom-kolom dalam DataFrame memuat informasi tentang gen,sumA, sumB dan fitness masing-masing individu.

## Evaluasi Populasi

Algoritma Genetika juga membutuhkan fungsi evaluasi populasi. Dalam evaluasi ini ada dua hal utama yang dilakukan yaitu: penghitungan nilai fitness dan pengurutan berdasarkan nilai fitness.

```
function randomGen(ukuran)
    s=""
    for i in rand(["A","B"],ukuran)
    s=string(s,i)
    end
    return s
end
```

Gambar 3.3: Sebuah fungsi yang menghasilkan string secara acak yang terdiri dar abjad "A" atau "B" dengan jumlah huruf sesuai dengan nilai peubah "ukuran".

```
function randomPop(ukuranGen=10,ukuranPop=100)
    populasi=DataFrame(gen=randomGen(ukuranGen),sumA=0,sumB=10,fitness=10)
    for i in range(1,length=ukuranPop-1)
        append!(populasi,DataFrame(gen=randomGen(ukuranGen),sumA=0,sumB=10,fitness=10))
    end
    return populasi
end
```

Gambar 3.4: Fungsi untuk mencari populasi yang terdiri dari individu yang memiliki gen acak. Populasi disimpan dalam sebuah DataFrame. Jumlah individu dalam populasi sama dengan jumlah baris dalam dataframe yang dipakai. Kolom-kolom dalam DataFrame memuat informasi tentang gen,sumA, sumB dan fitness masing-masing individu.

## Mutasi

Mutasi adalah perubahan nilai gen. Hal ini juga bisa diimplementasikan dengan Julia seperti yang ditunjukkan dalam Gambar 3.6 yang menunjukan bagaimana merubah sebuah huruf dalam gen.

Kemudian, fungsi mutasi dalam sebuah populasi dapat dibuat. Input fungsi ini adalah populasi dan peluang terjadinya mutasi. Berdasarkan ukuran populasi dan peluang mutasi, individu yang mengalami mutasi dipilih secara acak.

Input fungsi ini adalah populasi dan peluang terjadinya mutasi. Berdasarkan ukuran populasi dan peluang mutasi, individu yang mengalami mutasi dipilih secara acak. Lokasi terjadinya mutasi pun dipilih secara acak. Output fungsi ini adalah populasi baru yang sebagian kecil individunya telah mengalami mutasi.

```
function evaluatePopulation(population)
    for i in eachrow(population)
        fitness(i,w)
    end
    sort!(population,4)
    return population
end
```

Gambar 3.5: Fungsi evaluasi populasi terdiri dari penghitungan nilai fitness sesuai dengan Gambar 3.2 dan pengurutan berdasarkan nilai fitness dengan menggunakan fungsi **sort!**. DataFrame populasi dijelaskan dalam Gambar 3.4.

```
function changeGen(gen,location,set=['A','B'])
    data=[c for c in gen]
    lanjut=true

    while lanjut
        newValue=rand(set)
        if data[location]!=newValue
            data[location]=newValue
            lanjut=false
        end
    end
    hasil=string()
    for c in data
        hasil=string(hasil,c)
    end
    return hasil
end
```

Gambar 3.6: Penggantian sebuah huruf dalam sebuah gen. Struktur **if .. then** memastikan bahwa penggantian hanya dilakukan apabila nilai baru berbeda dengan nilai yang ada. Struktur **while .. then** menjamin bahwa penggantian harus terjadi.

```
function mutation(population,probability=0.1)
    populationSize=size(population)[1]
    jumlahMutan=populationSize*probability
    mutan=rand(range(1,length=populationSize),jumlahMutan)
    for m in mutan
        gen=population[m,1]
        location=rand(range(1,length=length(gen)))
        gen=changeGen(gen,location)
    end
end
```

Gambar 3.7: Fungsi mutasi dalam sebuah populasi. Input fungsi ini adalah populasi dan peluang terjadinya mutasi. Berdasarkan ukuran populasi dan peluang mutasi, individu yang mengalami mutasi dipilih secara acak. Lokasi terjadinya mutasi pun dipilih secara acak. Output fungsi ini adalah populasi baru yang sebagian kecil individunya telah mengalami mutasi.