Nama : Stephen Prasetya Chrismawan
NIM : H1D021025

UTS ALGEN (Shubert)

1. Menggunakan library untuk melakukan pengacakan, menggambarkan grafik

```
import random

import matplotlib.pyplot as plt

import numpy as np
```

2. Membuat fungsi untuk mengubah binary menjadi desimal, yaitu dengan cara membagi 17 bit menjadi 3 bagian yaitu, penanda positif atau negatif, nilai bulat, dan nilai koma

```
def bintodec(binary):

   sign = binary[0]

   bulat = binary[1:3]

   koma = binary[3:]

   decimalbulat = 0

   tertinggi_pangkat_bit = 1

   for k in bulat:

       decimalbulat = decimalbulat+ ( k * 2 **
tertinggi_pangkat_bit)

       tertinggi_pangkat_bit = tertinggi_pangkat_bit-1

   decimalkoma = 0

   tertinggi_pangkat_bit = 14

   for k in koma:

       decimalkoma = decimalkoma+ ( k / (2 **
tertinggi_pangkat_bit))

       tertinggi_pangkat_bit = tertinggi_pangkat_bit-1

   total =  round(decimalbulat + decimalkoma,4)

   if  sign == 0:

       return total
```

```
    else:
        mintotal = 0-total
        return mintotal
```

3. Membuat fungsi untuk menghitung nilai fitnes fungsi shubert dengan parameter bit yang berisi 34 bit, terdiri dari 17 bit X1 dan 17 bit X2

```
def fitness(bit):
    bitx = bit[:17]
    bity = bit[17:]
    decbitx = bintodec(bitx)
    decbity = bintodec(bity)

    res = 0
    res1 = 0
    res2 = 0

    for i in range(1, 6):
        res1 = res1 + (i*np.cos((i+1)*decbitx + i))
        res2 = res2 + (i*np.cos((i+1)*decbity + i))



    res = res1 * res2
    return res
```

4. Mendefinisikan variabel awal

```
populasi = []
jml_individu = 100
jml_gen = 34
p = 0.5
jumlah_generasi = 500
sumbuy=[]
```

```
sumbuy2 = []

rentangatas = 3

rentangbawah=-3
```

5. Membuat class Individu untuk menampung detail dari sebuah kromosom

```
#Membuat class Individu untuk dapat mengenali bit dan fitness

class Individu :

   def __init__(self, bit, id) :

       self.id = id

       self.bit = bit

       self.jadiparent = 0

       self.probabilitas = 0

       self.probkumulatif = 0

       self.fitness = fitness(bit)

       self.fakefitness = 0

       j = 0
```

6. Memulai pembuatan generasi awal dengan jumlah individu yang ditentukan. Dimulai dari pembuatan kromosom sesuai jumlah gen, kemudian, dijadikan 1 array, lalu dimasukkan sebagai class Individu. Kemudian semua kromosom yang telah terbentuk dijadikan 1 di populasi. Disini juga terdapat pengecekan untuk mengecek apakah nilai decimal sesuai dengan rentang yang diberikan (-3,3). Jika tidak sesuai maka akan digenerate ulang.

```
iloop = 1

while iloop <= jml_individu :

   calon = []


   #Melakukan perulangan untuk membuat 1 per satu gen

   for j in range(jml_gen):
```

```
        #melakukan pemilihan pengacakan antara 0 sampai 1 dengan
digit maksimal 4
        r = round(random.uniform(0, 1), 4)
        if r<p :
            r = 0 #jika nilai random kurang dari 0,5 maka jadi 0
        else :
            r = 1 #jika nilai random lebih dari 0,5 maka jadi 1

        #Menyisipkan gen (nilai r) ke calon individu
        calon.append(r)

    #pengecekan rentang
    bitx = calon[:17]
    bity = calon[17:]
    decbitx = bintodec(bitx)
    decbity = bintodec(bity)
    #Memasukan setiap individu ke kelas individu
    if (decbitx <= rentangatas and decbitx >=rentangbawah and
decbity <= rentangatas and decbity >=rentangbawah) :
        individu = Individu(calon, iloop)

        #Menyisipkan individu ke populasi
        populasi.append(individu)
        iloop+=1
```

7. Mendefinisikan fungsi bantuan untuk membuat nilai fitness menjadi positif dan akan dimasukkan ke dalam variabel ‘fakefitness’ yang terdapat di dalam class individu dari populasi yang terbentuk.

```
def nonnegatif(data):
    min_val = min(data)
    positive_data = [x + abs(min_val) + 0.1 for x in data]
    return positive_data
```

```
data = [x.fitness for x in populasi]
fake_fitness = nonnegatif(data)

sum = 0
u = 0
for i in populasi   :
   i.fakefitness = fake_fitness[u]
   u+=1
   sum += i.fakefitness
   sumbuy.append(i.fitness)

avg = sum/jml_individu
sumbuy2.append(avg)
```

8. Menghitung nilai probabilitas kumulatif pada setiap individu, namun berdasarkan nilai ‘fakefitness’ supaya tidak menjadi negatif.

```
urut = 1
probkumulatif = 0
for i in populasi   :
   i.probabilitas = i.fakefitness /sum
   probkumulatif+=i.probabilitas
   i.probkumulatif = probkumulatif
   urut +=1
```

9. Selanjutnya adalah memulai tahapan untuk melakukan algoritma genetika pada setiap generasi.

```
#Tahapan Cycle Generasi
igen = 1
```

```
while igen <= jumlah_generasi:
```

10. Selanjutnya adalah melakukan tahapan seleksi Parent1 dan Parent2. Diawali dengan melakukan perulangan sebanyak ½ dari jumlah individu supaya semuanya menjadi parent dan menghasilkan child. Selanjutnya terdapat perulangan untuk mengecek jika sudah mendapat 2 parent maka berhenti jika belum maka akan terus mencari parent. Pencarian Parent didaarkan pada Probabilitas Kumulatif, jika nilai probabilitas kumulatif yang di loop lebih dari angka yang diacak maka individu tersebut akan dipilih menjadi parent. Jika terpilih menjadi parent maka tidak bisa lagi menjadi parent.

```
# Tahap Seleksi menggunakan Probabilitas Kumulatif
   #random angka dari 0 - 1
   semua_parent = []
   for _ in range(int(jml_individu/2)):
       parents_pair = []
       Parent1 =[]
       idp1 = 0
       dapet = 0
       while dapet  == 0:
           s = round(random.uniform(0,1), 4)
           for l in populasi :
               if l.probkumulatif < s or l.jadiparent == 1:
                   continue
               elif l.probkumulatif >= s:
                   l.jadiparent = 1
                   Parent1 = l.bit.copy()
                   idp1 = l.id
                   dapet =1
                   break
       Parent2 = []
       idp2 = 0
       dapet = 0
```

```
        while dapet  == 0:
            r = round(random.uniform(0,1), 4)
            for j in populasi :
                if j.probkumulatif < r or j.jadiparent == 1 :
                    continue
                elif j.probkumulatif >= r:

                    j.jadiparent = 1
                    Parent2 = j.bit.copy()
                    idp2 = j.id
                    dapet =1
                    break
        parents_pair.append(Parent1)
        parents_pair.append(Parent2)
        semua_parent.append(parents_pair)
```

11. Selanjutnya jadikan individu jadi parent menjadi 0 lagi supaya bisa terpilih di generasi berikutnya.

```
    #jadikan self.jadiparent = 0 lagi
    for j in populasi :
            j.jadiparent = 0
```

12. Selanjutnya Melakukan CrossOver. Diawali dengan membuat masking corssover. Kemudian untuk semua parent jika masking crossoverya bernilai 1 maka terbentuk child 1 yang diisi index Parent 2 dan child 2 terisi index Parent 1 dan sebaliknya jika 0 maskingnya.

```
#Crossover
    #masking crossover
    # Membuat list masking
    masking_co1 = []
```

```
    # Mengisi list masking dengan nilai acak antara 0 dan 1
    for _ in range(jml_gen):
        nilai_acak = random.randint(0, 1)
        masking_co1.append(nilai_acak)
    childtotal = []
    for pairs in semua_parent:
        ParentOne = []
        ParentTwo = []


        ParentOne = pairs[0]


        ParentTwo = pairs[1]
        #print(f"\n Parent 0 = {ParentOne}")
        #print(f"\n Parent 1 = {ParentTwo}\n")
        child1 = []
        child2 = []
        i = 0
        while i < len(masking_co1) and i < len(ParentOne) and i <
len(ParentTwo):
            if masking_co1[i]==1 :
                child1.append(ParentTwo[i])
                child2.append(ParentOne[i])
            elif masking_co1[i]==0 :
                child1.append(ParentOne[i])
                child2.append(ParentTwo[i])
            i+=1
```

13. Selanjutnya adalah melakukan mutasi, jika angka random yang diberikan lebih kecil dari probabilitas mutasi maka child akan dibalik dari 0 menjadi 1 dan sebaliknya.

```
```

```
#Mutasi

#Probabilitas Mutasi

prob_mutasi = 1/(jml_gen)


#Mutasi Child 1

child1termutasi = []

randommutasi = random.uniform(0,1)

for k in child1 :

    if randommutasi<prob_mutasi :

        if k==0:

            child1termutasi.append(1)

        elif k==1:

            child1termutasi.append(0)

    else :

        child1termutasi.append(k)

#Mutasi Child 2

child2termutasi = []

randommutasi = random.uniform(0,1)

for k in child1 :

    if randommutasi<prob_mutasi :

        if k==0:

            child2termutasi.append(1)

        elif k==1:

            child2termutasi.append(0)

    else :

        child2termutasi.append(k)
```

14. Kemudian, tahap selanjutnya melakukan mutasi kedua yaitu dengan swap mutation, kita akan memilih secara random 2 index yang akan saling bertukar.

```
#MUTASI KEDUA
```

```
        #menggunakan swap mutation
        #CHILD 1
        index1=0
        index2 =0
        while 1 :
            index1 = random.randint(0,(jml_gen)-1)
            index2 = random.randint(0,(jml_gen)-1)
            if index1!=index2:
                break

        child1termutasi[index1],  child1termutasi[index2] =
child1termutasi[index2],  child1termutasi[index1]

        #CHILD 2
        index1=0

        index2 =0
        while 1 :

            index1 = random.randint(0,(jml_gen)-1)
            index2 = random.randint(0,(jml_gen)-1)
            if index1!=index2:
                break

        child2termutasi[index1],  child2termutasi[index2] =
child2termutasi[index2],  child2termutasi[index1]

        childtotal.append(child1termutasi)
        childtotal.append(child2termutasi)
```

15. Langkah selanjutnya, yaitu dengan tahapan Subtitusi, disinii kita akan menggantikan nilai fitness yang terburuk (fitness populasi lebih besar dari fitness child) dengan child yang baru dibentuk. Namun sebelum itu dilakukan pengecekan apakah nilai decimal dari Child memenuhi rentang atau tidak (-3 , 3).

```
   #mencari dan menggantikan nilai fitness terendah
   for c in childtotal:


       fitnesschild = fitness(c)
       bitx = c[:17]
       bity = c[17:]
       decbitx = bintodec(bitx)
       decbity = bintodec(bity)
       #Perhitungan dan Penggantian dari setiap child
       if (decbitx >= rentangbawah and decbitx<=rentangatas) and
(decbity >= rentangbawah and decbity<=rentangatas) :
           for q in populasi :
               if fitnesschild < q.fitness:
                   q.bit =  c
                   q.fitness = fitnesschild


                   break
               else:
                   continue
```

16. Selanjutnya adalah memasukkan nilai probabilitas dan probabilitas sesuai dengan perhitungan setiap individu lagi.

```
   # membuat nilai probabilitas dan prob. kumulatif sesuai
perhitungan lagi setiap generasi.
   sum = 0
   u = 0
   for i in populasi   :
       i.fakefitness = fake_fitness[u]
```

```
        u+=1
        sum += i.fakefitness
        sumbuy.append(i.fitness)
    urut = 1
    probkumulatif = 0
    for i in populasi   :
        i.probabilitas = abs(i.fakefitness /sum)
        probkumulatif+=i.probabilitas
        i.probkumulatif = probkumulatif
        urut +=1
```

17. Kemudian kita akan mencari nilai Fitness terkecil, X1,X2 yang dipilih dan ditampilkan di terminal untuk setiap generasi.

```
    #Perhitungan best  nilai fitness generasi ke generasi
    avg = sum/jml_individu
    minfit = 0
    minBit = 0
    minX1 = 0
    minX2 = 0
    ax = 1
    for p in populasi :
        if ax == 1:
            minfit = p.fitness
            minBit = p.bit
        else :
            if p.fitness < minfit :
                minfit = p.fitness
                minBit = p.bit
        ax+=1
    minX1 = bintodec(minBit[17:])
```

```
    minX2 = bintodec(minBit[:17])
    sumbuy2.append(minfit)
    print(f"\nBest Fitness Generasi ke - {igen} = {minfit}")
    print(f"Best BIT Generasi ke - {igen} = {minBit}")
    print(f"Best X1 Generasi ke - {igen} = {minX1}")
    print(f"Best X2 Generasi ke - {igen} = {minX2}")
    igen+=1
```

18. Terakhir adalah dengan membuat tampilan grafik untuk menampilkan seluruh titik fitness setiap individu, dan juga untuk menampilkan nilai fitness paling rendah dari generasi ke generasi.

```
#Membuat tampilan grafik
i=0
sumbux=[]
while i <= jumlah_generasi:
    j = 0
    while j < jml_individu:
        sumbux.append(i)
        j+=1
    i+=1
xpoints = sumbux
ypoints = sumbuy
i=0
sumbux2=[]
while i <= jumlah_generasi:
    sumbux2.append(i)
    i+=1
plt.plot(xpoints, ypoints, 'o', markersize=2, label='Semua
Fitness')
plt.plot(sumbux2, sumbuy2, 'o', markersize=3,color="red",
linestyle="-", label='Best Fitness')
```

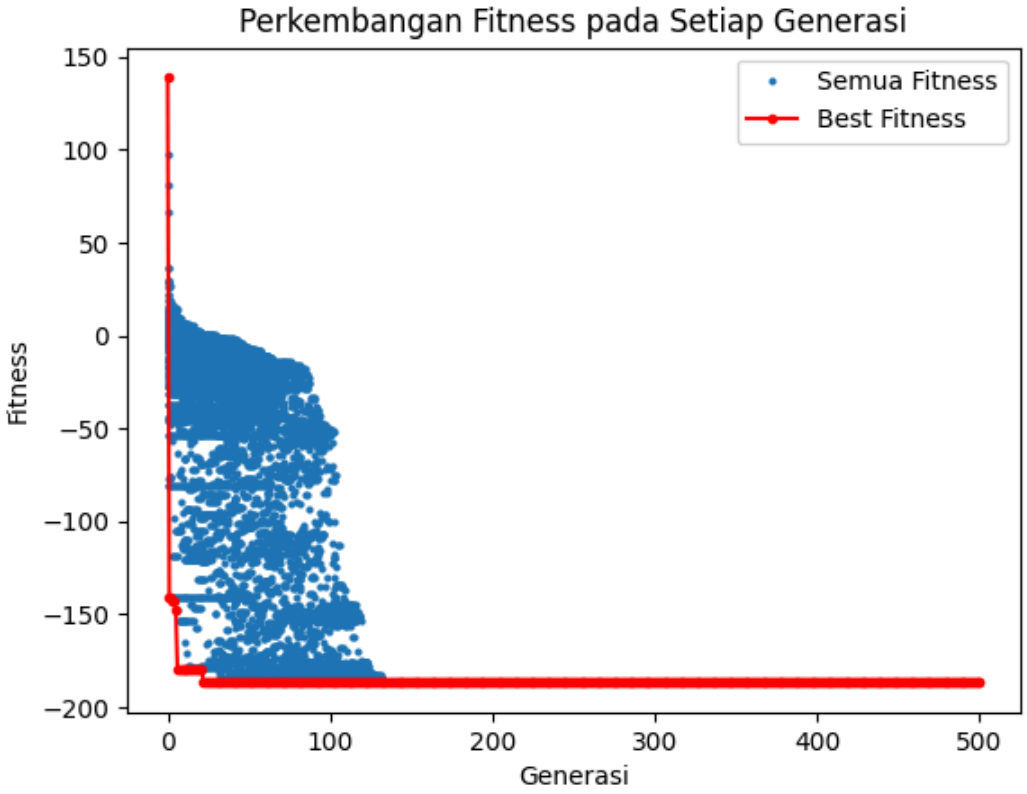
```
plt.xlabel('Generasi')

plt.ylabel('Fitness')

plt.title('Perkembangan Fitness pada Setiap Generasi')

plt.legend()

plt.show()
```

Berikut hasil Terminal untuk perulangan terakhir

```
Best Fitness Generasi ke - 500 = -186.730
9059635426
Best BIT Generasi ke - 500 = [1, 0, 1, 1,
 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1
, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0,
0, 1, 1]
Best X1 Generasi ke - 500 = -0.8003
Best X2 Generasi ke - 500 = -1.4251
```

Nilai X1 untuk Fitness terendah = -0.8003
Nilai X2 untuk Fitness terendah = -1.4251
Fitness Terendah = -186.730

Saya melakukan dengan 100 individu untuk 500 generasi.



Gambar Perkembangan Nilai Fitness dari Generasi ke Generasi