複雑系プログラミング特論
遺伝的アルゴリズム

概要

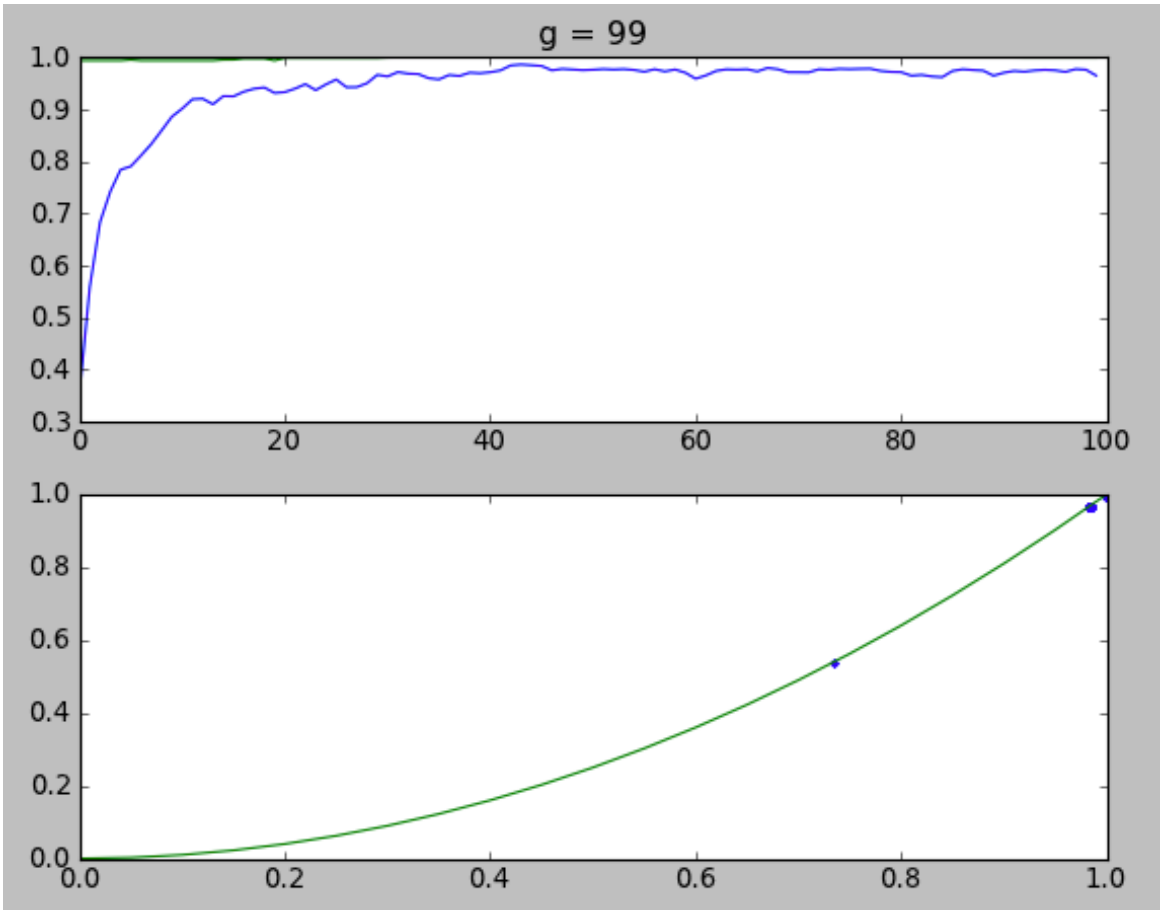
この講義の最後のトピックとして，遺伝的アルゴリズムを紹介します．といっても，このアルゴリズム自体は多くの人が他の講義等を通じて知っているかもしれません．
しかし，百聞は一見にしかず，いや百見は一実行にしかず，いや，百実行は一実装にしかず．．．ということで．．．

遺伝的アルゴリズム

進化的計算手法とは，生物が適応進化する仕組みを模して，さまざまな最適化問題を解く汎用的な解の探索手法の総称です．

単に解を探し出すだけでなく，生物が進化する過程をコンピュータ上で再現するアルゴリズムとしても広く利用されています．

遺伝的アルゴリズム（Genetic Algorithm,GA）は，そのうちHollandが定式化した代表的なものです．そのエッセンスを，f(x)=x^2（0<=x<=1.0)の最大値 （とそれ をもたらすx）を求める関数最適化問題を用いて整理します．

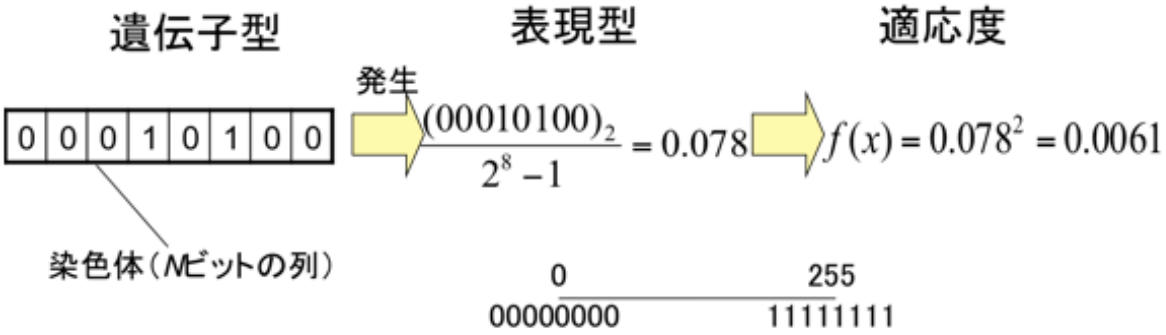


概要

遺伝的アルゴリズムでは，解の候補１つずつを個体，個体がたくさん集まった集合を集団を呼びます．この集団の中から評価の高い個体がより多く子孫を残して次世代の集団をつくる進化操作を繰り返すことで，次第に集団の評価値を上げつつ，より良い解を探索します．

個体の表現と評価

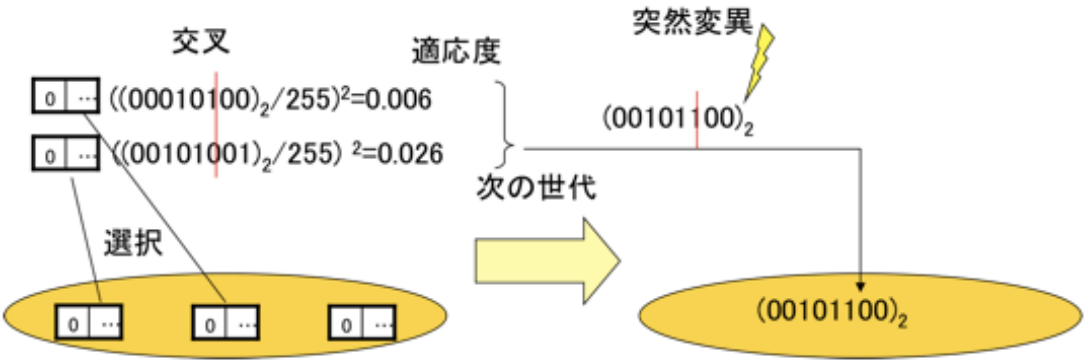
- 遺伝子型：可能な解をコードしたビット情報
  - N個の遺伝子座にそれぞれ0または1の値（Nビットの列）
- 表現型：探索したい解そのもの．
  - Nビットの遺伝子の列を2進数だとみなして，10進数にした値を2^(N-1)で割ったもの（0以上1以下）
- 適応度：表現型に対する評価値．子個体の生き残りやすさ．
  - f(x)= x^2のxに表現型の値を代入してできた値



進化操作

- 選択：適応度の高い個体を親個体として選ぶ
  - ルーレット選択：集団の中から各個体の適応度の比例した確率で個体を選ぶ．
- 交叉
  - 選択した2個体の遺伝子列をランダムな位置で組みかえ，その片方を子の持つ遺伝子とする（両方子孫にしてもよい）．
- 突然変異

- ある小さな確率で遺伝情報が書き換わる
- 各子孫の各遺伝子について，ある一定の確率pでその値が反転する.



サンプルプログラム

```
mport numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import random as rnd

# set parameters
N= 100
T= 100
W= 30
SEED=101
agents= []


def gtypeToPtype(gtype):
    return(np.sum([gtype[i]*(2**i) for i in range(L)])/float(2**L-1))

def fitnessFunction(p):
    return(p**2)

#def fitnessFunction2(p):
#    return(np.sin(17.0*p)*np.cos(35.0*p)+1.5)

## define classes
class Agent(object):

    def __init__(self, gtype):
        self.genotype= gtype[:]
        self.phenotype= None
        self.fitness= 0.0

    def getOffspring(self):
        o= Agent(self.genotype)

        for i in range(L):
            if (rnd.random()<MUT):
                o.genotype[i]= 1-o.genotype[i]

        return(o)
```

```
39
40      def develop(self, dfunc):
41          self.phenotype= dfunc(self.genotype)
42
43      def evaluate(self, efunc):
44          self.fitness= efunc(self.phenotype)
45
46
47  def selectAnAgentByRoulette(pop):
48      total= sum([i.fitness for i in pop])
49      val= rnd.random()*total
50      for i in pop:
51          val-= i.fitness
52          if (val<0):
53              return(i)
54
55
56  def crossover(a1, a2):
57      point= rnd.randint(1, L-1)
58      for i in range(point, L):
59          a1.genotype[i], a2.genotype[i]= a2.genotype[i], a1.genotype[i]
60
61
62
63  # initialize variables
64  SEED=1010
65  N= 100
66  G= 100
67  L= 20
68  MUT= 0.001
69  CROSS= 0.2
70  rnd.seed(SEED)
71  population= [Agent([rnd.randint(0, 1) for j in range(L)]) for i in ran
72  fig= plt.figure()
73
74  #some variables for graphs
75  averageFitness= []
76  bestFitness= []
77  best= population[0]
78  sx= np.arange(0, 1.01, 0.01)
79  sy= [fitnessFunction(i) for i in sx]
80
81
82  # main loop (call back function for animation)
83  def main_loop(t):
84      step()
85      update(t)
86
87  # events in a step
88  def step():
89      global population
90      #fitness evaluation
91      best= population[0]
92
93      for a in population:
94          a.develop(gtypeToPtype)
95          a.evaluate(fitnessFunction)
96          if(a.fitness>best.fitness):
97              best= a
98      averageFitness.append(np.average([a.fitness for a in population]))
```

```
99      bestFitness.append(best.fitness)
100
101     print str(best.genotype)+":"+str(best.fitness)
102     #evolution
103     newpop= []
104     for i in range(N/2):
105         n1= selectAnAgentByRoulette(population).getOffspring()
106         n2= selectAnAgentByRoulette(population).getOffspring()
107
108         if rnd.random()<CROSS:
109             crossover(n1, n2)
110         newpop.append(n1)
111         newpop.append(n2)
112
113     population= newpop
114
115 # update function for graph
116 def update(t):
117     fig.clear()
118
119     ax1= fig.add_subplot(2, 1, 1)
120     ax1.plot(averageFitness)
121     ax1.set_title('g = ' + str(t))
122     ax1.plot(bestFitness)
123     ax1.set_xlabel("generaiton")
124     ax1.set_ylabel("avrage / best fitness")
125
126
127     ax2= fig.add_subplot(2, 1, 2)
128     x= [a.phenotype for a in population]
129     y= [a.fitness for a in population]
130     ax2.plot(x, y, ".")
131     ax2.plot(sx, sy)
132     ax2.set_xlabel("x")
133     ax2.set_ylabel("fitness")
134
135     fig.tight_layout()
136
137
138 ani = animation.FuncAnimation(fig, main_loop, np.arange(0, T), interva
139 plt.show()
```

（練習）最適解を求める関数の形を変えてみよう.
たとえば,
f(x)=sin(17x)*cos(35x)+1.5
ただし，sin(x)はnp.sin(x)，cos(x)はnp.cos(x)

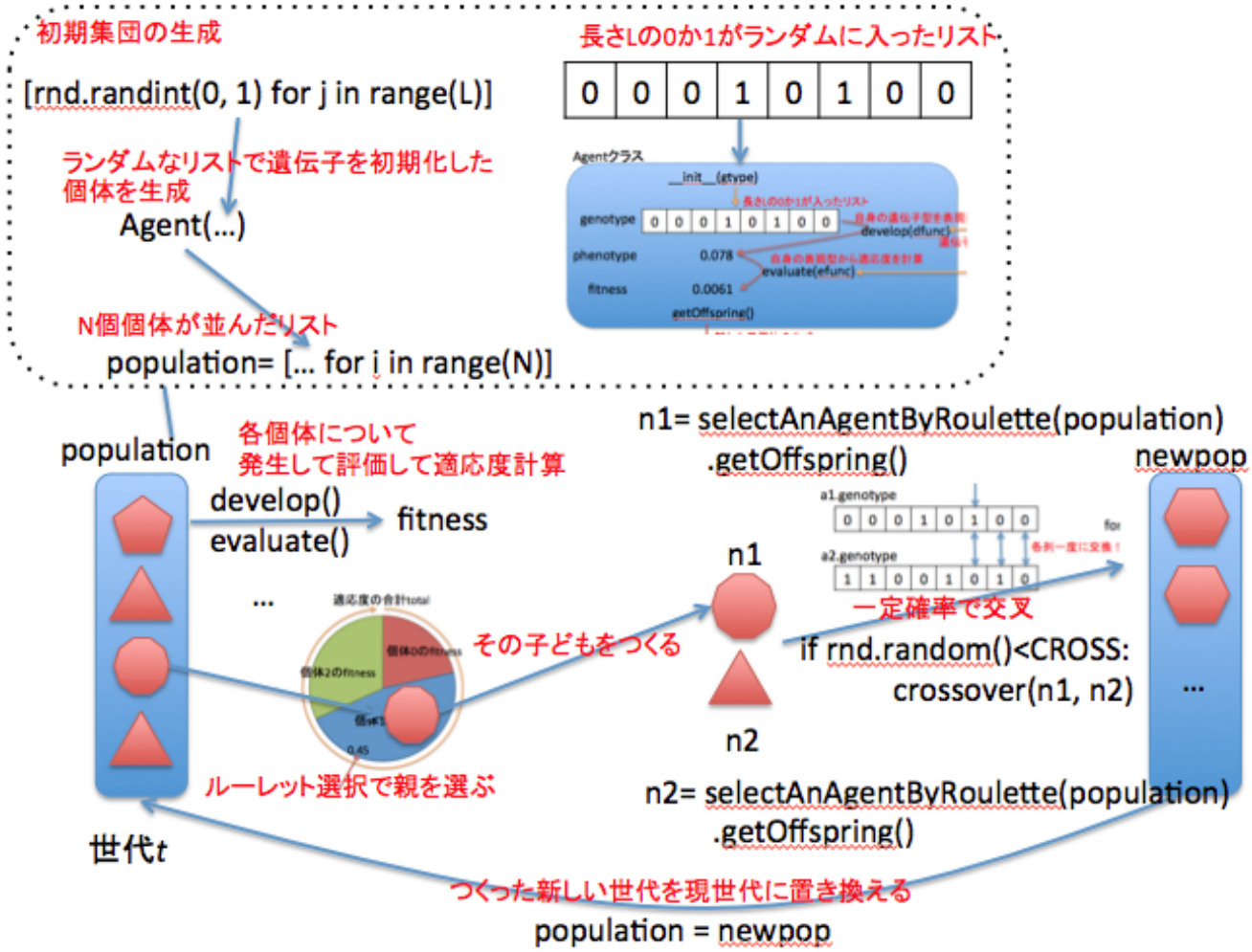
うまくいってる？

プログラム解説

メインループ

個別の要素や操作については下を参照.

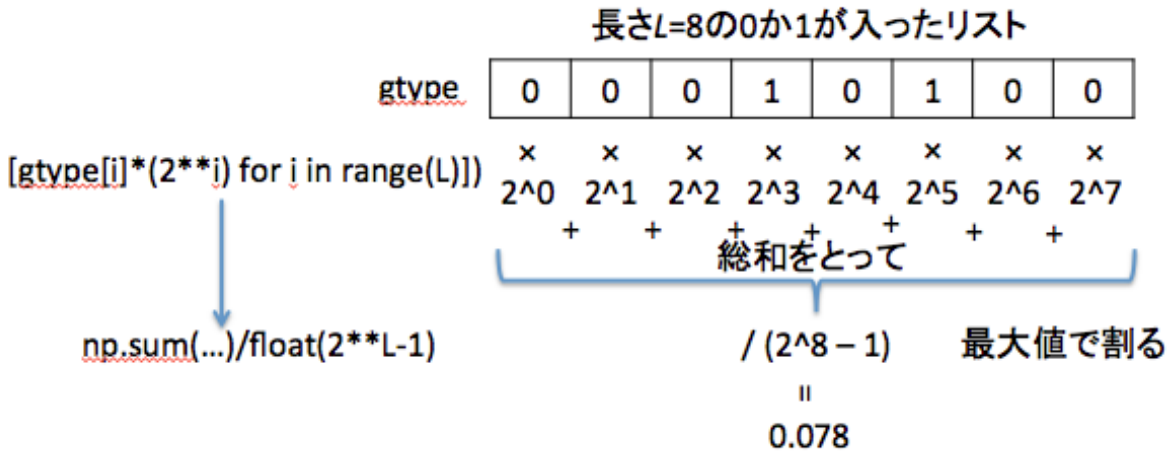
- 各個体について
    - 遺伝子型から表現型を発生させる.
    - 表現型を評価して適応度を算出する.
- 新しい集団用のリストnewpopを用意して，2個体ずつ適応度に比例したルーレット選択で選んだ親個体から子個体を作成する．確率CROSSで2個体の遺伝情報を一点交叉する．両者をnewpopに加える.
- 最後に，newpopをpopulaitonと置き換える.



変換や評価のための関数

gtypeToPtype(gtype)

- 遺伝子列gtypeを引数にして，それを表現型（この場合は0〜1までの実数）に変換してその値を返す関数.



fitnessFunction(p)

- 最適化を行う関数そのもの．引数に個体の表現型の値を入れて出てきた値をその適応度とするのに利用．この場合はp^2を返すごく簡単な関数．

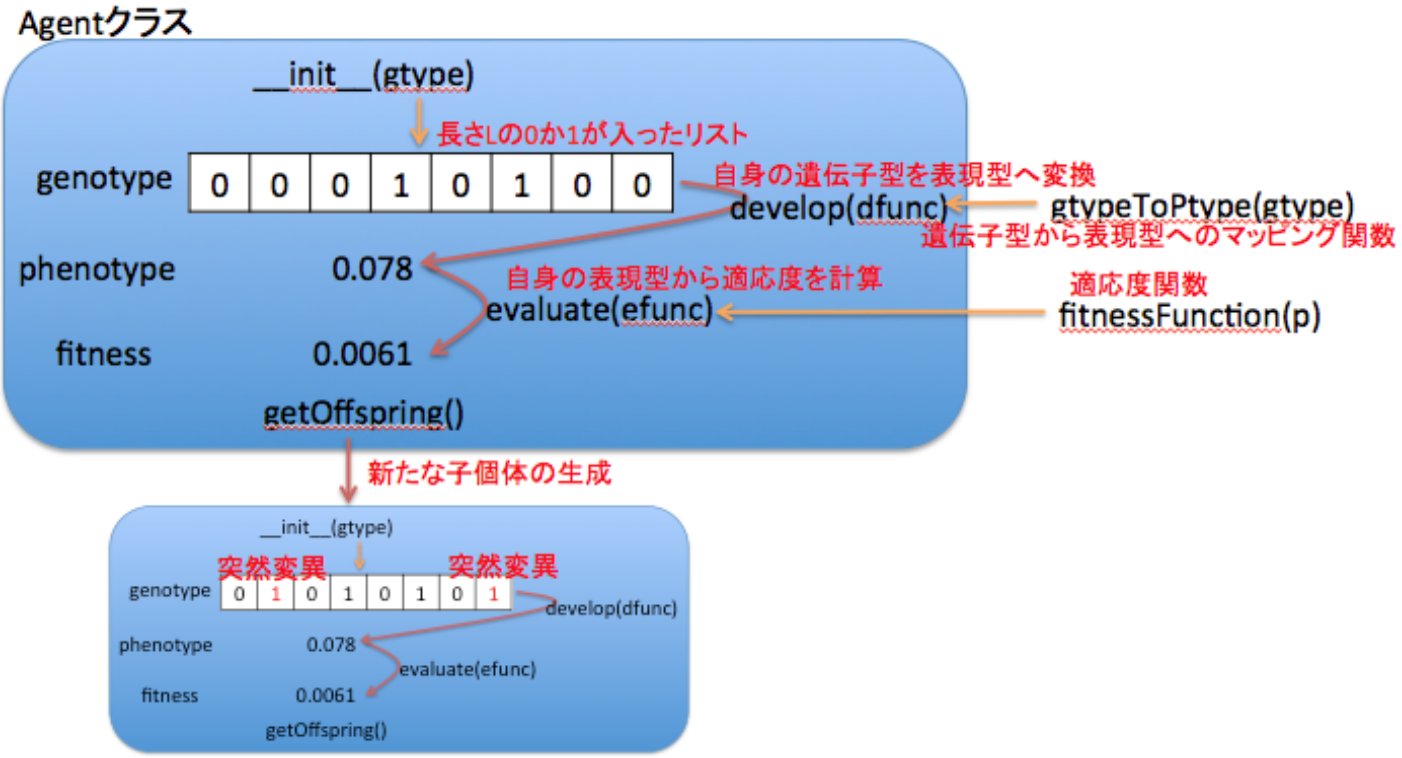
Agentクラス

変数（フィールド）：

- genotype
  - 長さLのビット列からなる遺伝子型を表す，長さLで0か1が入ったリスト
- phenotype
  - 遺伝子型から発生過程（マッピング）を経てつくられる表現型．この場合は遺伝子型を二進数として見た場合の値
- fitness
  - 表現型の良さを評価して得られる適応度

関数（メソッド）：

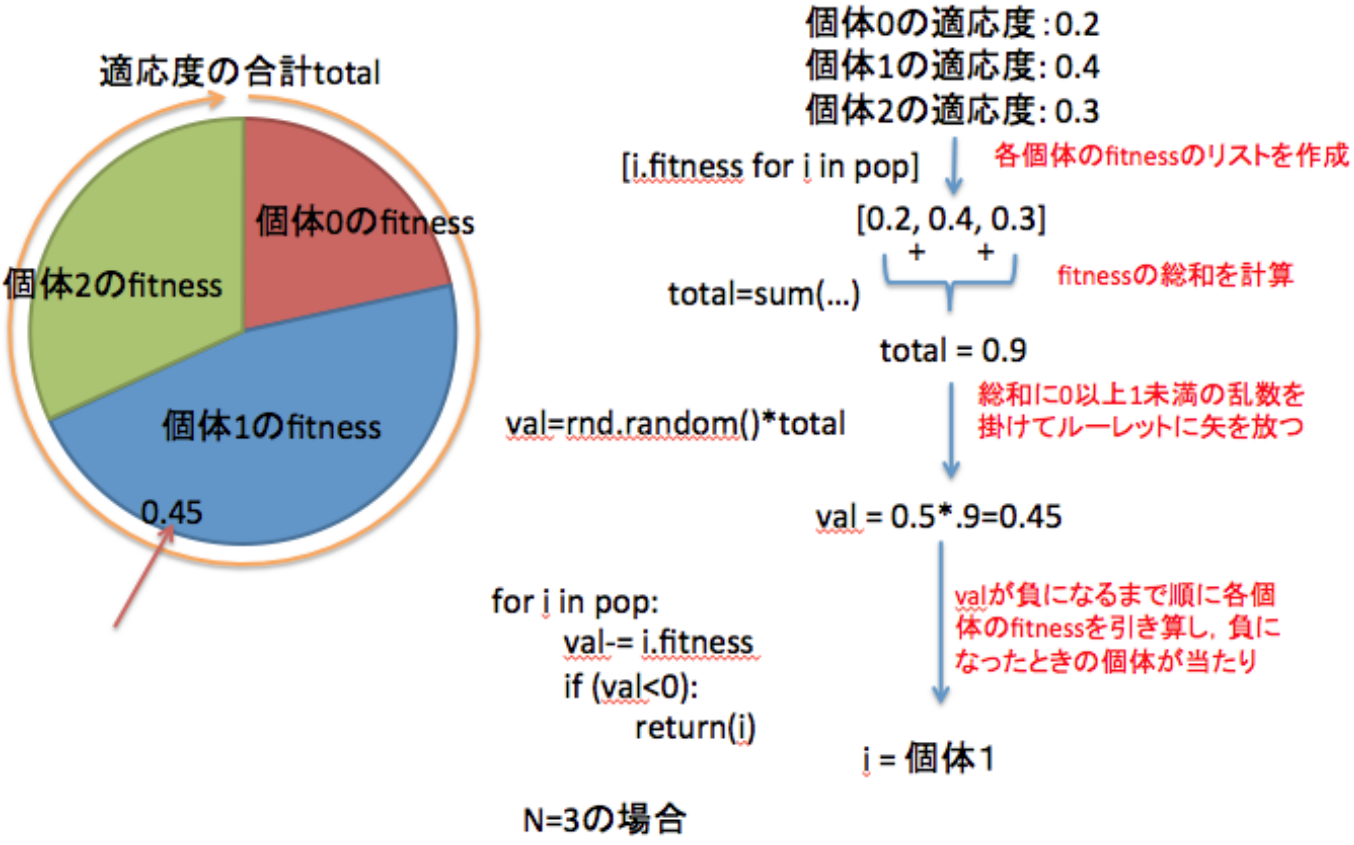
- __init__（初期化）
  - 長さLのビット列（実際には各値は整数値）を表すリストgenotypeとそれを表現型に変換した値を入れるphenotype，適応度を入れるfitness と いう変数を持つAgentクラスを定義．初期化の際にはリストを引数にとり，それのコピーをgenotypeとする．
    - gtype[:]はリストgtypeと同じ中身を持ったコピー
- develop(dfunc)
  - 引数に関数をとり，それを使って遺伝子型を表現型に変換したものをphenotypeに代入する．今回はgtypeToPtypeを代入して使う．pythonで は関数も変数の様に（オブジェクトとして）扱える．（発生に外の関数をつかうというのはやや不自然かもしれないけど，環境が発生過程に関与すると考えたり，工学的に遺伝子型から表現型へのマッピング関数が外から与えられていると考えたりして．）
- evaluate(efunc)
  - 引数に関数をとり，それを使って表現型から適応度を計算し，fitnessに代入する．今回はfitnessFunctionを代入して使う．



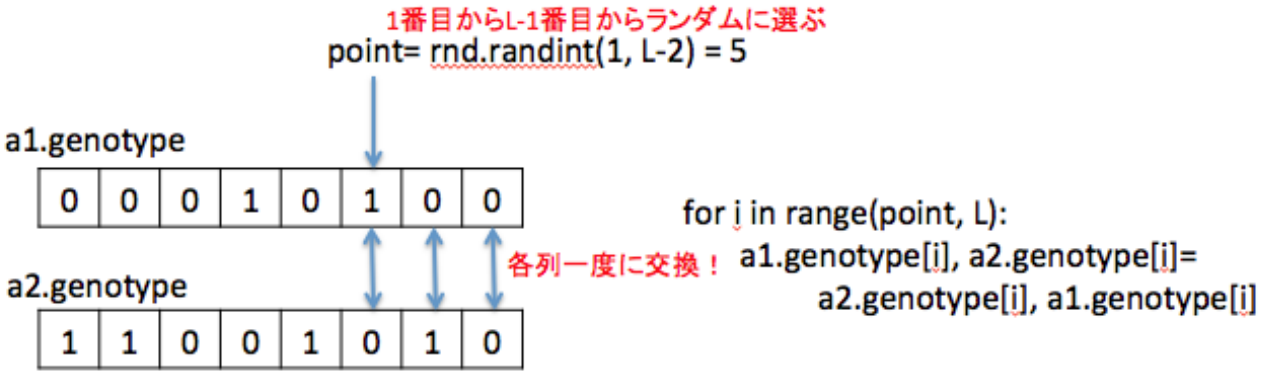
進化操作のための関数

selectAnAgentByRoulette(pop)

- 引数でとったAgentクラスのオブジェクト群popから，その適応度に比例したルーレット選択で1個体を選んで返す関数



crossover(n1, n2)
Agentクラスのオブジェクトn1, n2を引数にとり，染色体上のランダムな位置で一点交叉をおこなう．n1, n2で指定したオブジェクトの遺伝子そのものを書き換える．



（補足：オブジェクト指向の観点からは，本来はgenotypeやfitnessなどはオブジェクトの外側から直接参照せずに，メソッドを定義して取得するべきですが，簡単化のために直接参照しています）

グラフ表示など

- 平均適応度と適応度の最高値の推移を最初の図で表示．
- fitnessFunction関数の形を可視化するために，定義域の値sxと対応する関数の値syを入れたリストを用意し，2つ目の図で個体の分布に重ね合わせる．

いろいろな拡張

上に示したのは，遺伝的アルゴリズムのうちでも最もシンプルなものの一つです．例えば，以下のように，アルゴリズムを構成する各要素に関して様々なバリエーション が可能です．

遺伝子型から表現型への変換：

- 遺伝子中の1の割合を表現型にする表現法．
- グレイコード（ビット列で表現されたある値に対して＋－１した値のビット表現がもとのビット列と一箇所のみ違うコーディング法）

適応度の定義（スケーリング）：

- 現在の集団の適応度の最低値を全個体の適応度から引き算
- 適応度を指数の肩にのせる

選択：

- トーナメント選択：集団全体からランダムに選んだP個体から最も適応度の高い個体を選ぶ

- ランク選択：適応度の高い順に，次世代に選ばれる確率を割り当て直す
- （エリート保存：最良個体をそのまま次世代に残す）

交叉：

- 2点交叉：染色体上の任意の2地点の間だけ交叉
- 一様交叉：各遺伝子についてどちらの親から受け継ぐか決める

これらを上のプログラムに実装するにはどうすれば良いか，考えてみると面白いでしょう．

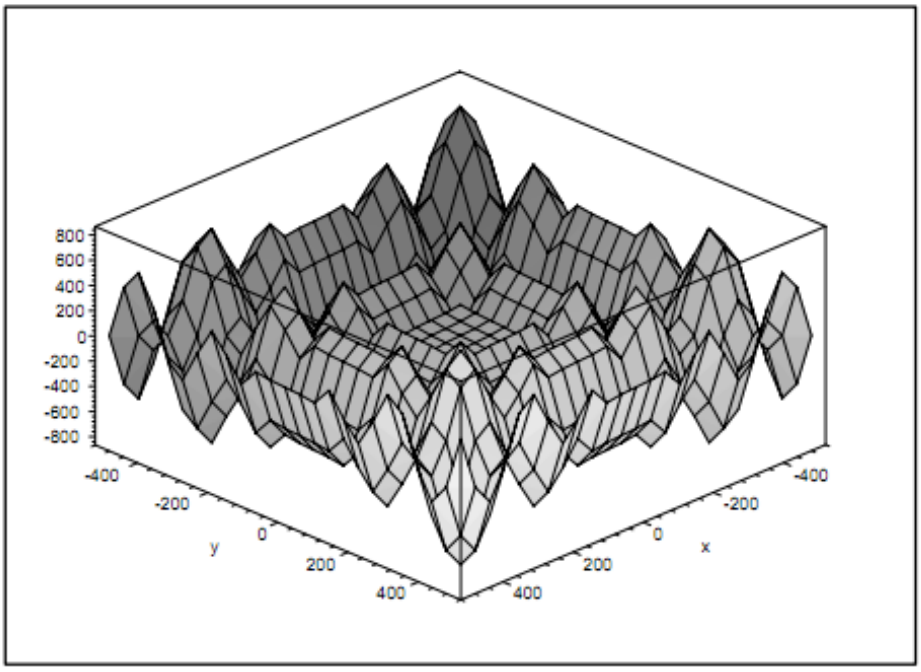
（練習）

1. 2個体ランダムに選んでその中から適応度の高い方を選ぶトーナメント選択をする関数selectAnAgentByTournament(pop)を作成してみましょう．
2. 最初のルーレット選択を使った場合と，作成したトーナメント選択を使った場合で解の探索過程に違いがあるか，調べてみましょう．

いろいろなベンチマーク問題

関数最適化は，遺伝的アルゴリズムのような組み合わせ最適化手法の効率性をはかるベンチマークとしてよく利用され，いくつかのよく知られた関数があります．例えば こんなものがあります．

Schefel's function



Rysunek 7: An overview of Schwefel's function in 2D, $f(x, y) = -x \sin(\sqrt{|x|}) - y \sin(\sqrt{|y|})$

（http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdfよ り）

この関数は任意の数（例えばn個）の変数x_i（定義域を-500<=x_i<=500とする）をとり，いずれもx_i=420.9687となるとき 最小値-4189829*nをとります．
最大値でなくて最小値を求めるベンチマーク問題だということに注意して，以下のように2変数の問題を解くプログラムを作成してみます．

遺伝子型から2変数の表現型への変換
今回は，長さLの遺伝子列を半分に切って，前半と後半でそれぞれ2つの変数の値を2進数で表現しているものとしましょう．なので，genotypeToPhenotype関数を以下のように書き換えます．

```
def gtypeToPtype(gtype):
    x1= np.sum([gtype[i]*(2**i) for i in range(L/2)])/float(2**(L/2)-1)*1000.0-500.0
    x2= np.sum([gtype[i+(L/2)]*(2**i) for i in range(L/2)])/float(2**(L/2)-1)*1000.0-500.0
    return(x1, x2)
```

適応度関数
今回の適応度関数は2つの値を引数にして，それを上の関数で変換した値を返します．

```
def fitnessFunction(p1, p2):
    return(-p1*np.sin(np.sqrt(abs(p1)))-p2*np.sin(np.sqrt(abs(p2))))
```

合わせて，Agentクラス中のevaluate関数を以下のように書き換えます．

```
    def evaluate(self, efunc):
        self.fitness= efunc(self.phenotype[0], self.phenotype[1])
```

ランク選択

今回の適応度関数は，値が小さいほど成績がよいので，fitnessFunctionから出てくる適応度関数の値をそのまま用いることができません．今回は，元の 適応度の良さがk番目の個体が適応度kを得る単純なランク選択を定義し，用いてみます．

```
def selectAnAgentByRank(pop):
    pop.sort(lambda x, y: int(x.fitness-y.fitness))
    rank= range(N, 0, -1)

    total= sum(rank)
    val= rnd.random()*total
    for i in range(N):
        val-= rank[i]
        if (val<0):
            return(pop[i])
```

グラフ表示
最後に，2つ目のグラフを二次元表示に変更します．"#some ..."で始まる該当箇所を以下のように書き換えます．

```
#some variables for graphs
averageFitness= []
bestFitness= []
best= population[0]
sx= np.arange(-500, 500.1, 20)
sy= np.arange(-500, 500.1, 20)
SX, SY= np.meshgrid(sx, sy)
SZ= fitnessFunction(SX, SY)
```

次に，ax2に関する記述を以下の通りに書き換えます．

```
    ax2= fig.add_subplot(2, 1, 2)
    x= [a.phenotype[0] for a in population]
    y= [a.phenotype[1] for a in population]
    ax2.scatter(x, y)
    plt.hold(True)
    ax2= fig.add_subplot(2, 1, 2)
    ax2.imshow(SZ, origin='lower', extent=[-500, 500, -500, 500])
    plt.hold(False)
```

今回は，個体の分布の下にmatplotlibのimshowというイメージマップを表示する関数を用いて関数の二次元図を作成して表示しました．

メインループ：
メインループでは，上で定義したランク選択を用いて個体を選ぶようにします．
さらに，最良個体を選ぶところでは，以前とは逆に適応度の値が小さいものを選ぶことにします．

以上の変更を加えて，正しく最小値に向かう方向に集団が進化していきましたか？
http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf
にはいろんな形のベンチマーク関数が載っています．複雑な形の関数を使って素朴なGAが最適解

を見つけられるか，また，ＧＡの各種設定が探索にどんな影響を与えう るか試してみるとよいでしょう.

ナップザック問題

遺伝的アルゴリズムの便利なところは，基本的な枠組みに従えば，解の遺伝子表現と評価方法さえ決めれば何でも（理想的には）最適化できることで す．先週の関数最適化のベンチマーク問題は答えが直感的にわかるので，いまいち驚きがないですよね．せっかくなので，すぐには答えがわからない問題を解いてみま しょう.
次のような状況設定があります.

押し入れ問題
ただし君は，実家から引っ越してきたところで，荷物の整理をしています．しかし，押し入れが一つしかなく，持ってきた荷物のすべてをしまうことが出来ませ ん．荷物は20種類あり，それぞれ大きさ(s)と彼にとってどれだけ重要か(i）が下記のように決まっているとします．押し入れの大きさを200とすると，どの荷 物を選んでしまえば，しまった荷物の重要さの合計を最大にすることができるでしょうか.

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s	11	6	31	13	10	29	30	35	31	28	15	12	38	43	20	15	45	58	23	34
i	35	34	46	28	38	47	45	55	40	52	44	33	55	63	39	31	60	70	40	43

11,6,31,13,10,29,30,35,31,28,15,12,38,43,20,15,45,58,23,34
35,34,46,28,38,47,45,55,40,52,44,33,55,63,39,31,60,70,40,43

これはナップザック問題と一般には呼ばれていて，ある制約条件のもと，評価を 最適化する問題の典型例の一つです.

この問題を解く遺伝的アルゴリズムをサンプルを改変して作成し，最適解を探してみましょう.
最適解：重要度の合計439，サイズの合計200 [1,1,1,0,1,1,0,0,0,1,1,1,0,0,1,1,0,0,1,0]

実装の方針のヒント：
荷物の情報の表現：i番目の荷物の大きさをsize[i]，重要さをimportance[i]とする.

遺伝子型：Lを荷物の数として，長さLのビット列とする.

表現型：i番目の遺伝子が1の場合にはi番目の荷物をしまう，0の場合にはしまわないとして，荷物の選び方を表現する．実質的に遺伝子型と同じなので，プ ログラムでの遺伝子型から表現型への変換は，遺伝子型の値（つまり長さLのリスト）をそのまま返す（かコピーする）ものとする.

（荷物の数が8の場合）

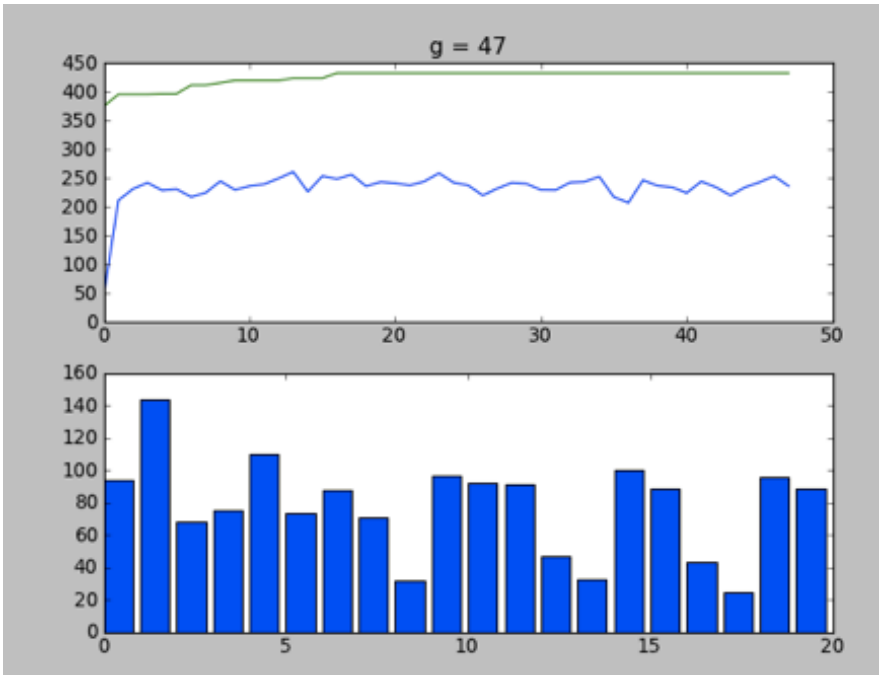
genotype	1	1	0	0	1	0	1	0
				そのままコピー				
phenotype	1	1	0	0	1	0	1	0

意味	0	1	2	3	4	5	6	7
	o	o	×	×	o	×	o	×

oは対応する番号の荷物をしまう，×はしまわない

適応度：しまう荷物の大きさの合計が200を超える場合には0とし，それ以外はしまう荷物の重要さの合計をそのまま適応度とする．

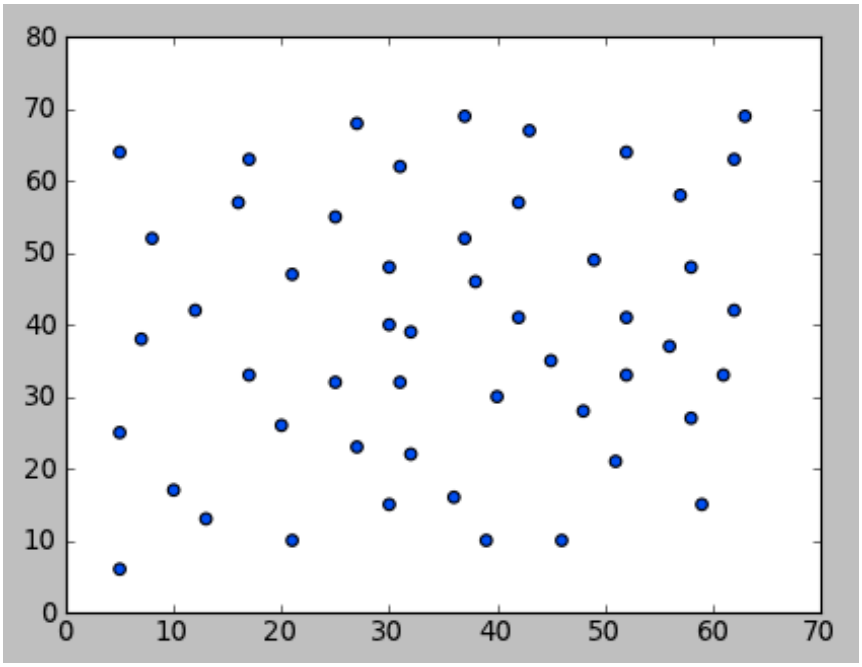
グラフ表示：たとえば，現在の集団の適応度の平均と最大値の推移のグラフに加え，各遺伝子座における1の頻度の分布を表すものとする．つまり，横軸iとし て現在の集団のi番目の遺伝子に1がどれだけ含まれるかカウントしたものを縦軸にとって，全体の探索の様子を示すものとする．



巡回セールスマン問題

もう一つ，組み合わせ最適化問題の代表例を紹介します．巡回セールスマン問題と呼ばれるものです．

あるセールスマンが，営業でいくつかの都市にある営業所をくまなく回らなけれ ばなりません．どれも一度ずつ回るとすると，移動距離を最短にするにはどう回ればよいでしょうか．

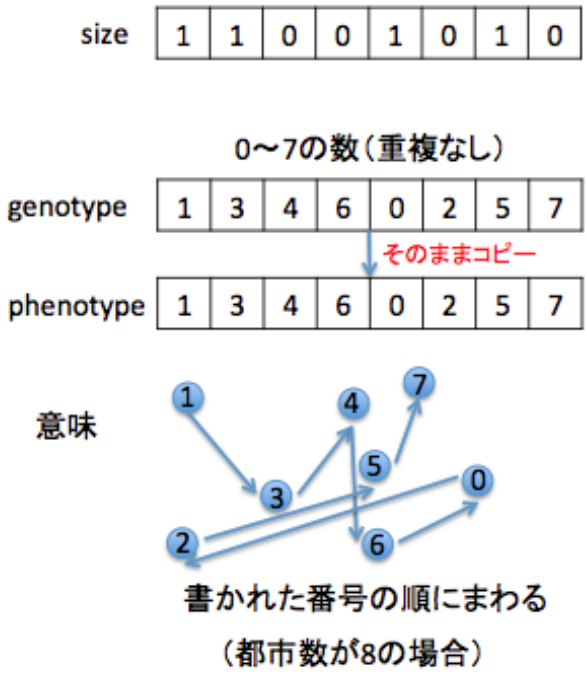


都市配置（eil51）
x座標：[37,49,52,20,40,21,17,31,52,51,42,31,5,12,36,52,27,17,13,57,62,42,
16,8,7,27,30,43,58,58,37,38,46,61,62,63,32,45,59,5,10,21,5,30,39,32,25,25,48,56,30]
y座標： [52,49,64,26,30,47,63,62,33,21,41,32,25,42,16,41,23,33,13,58,42,57,57,52,
38,68,48,67,48,27,69,46,10,33,63,69,22,35,15,6,17,10,64,15,10,39,32,55,28,37,40]
最適解：ユークリッド距離ではかった巡回経路長429.98
[1, 22, 8, 26, 31, 28, 3, 36, 35, 20, 2, 29, 21, 16, 50, 34, 30, 9, 49,10, 39, 33, 45, 15, 44, 42, 40, 19,
41, 13, 25, 14, 24, 43, 7, 23, 48, 6, 27, 51, 46, 12, 47, 18, 4, 17, 37, 5, 38, 11, 32]

実装の方針のヒント：
都市情報の表現：L都市あったとして，都市に0〜L-1までの番号を付け，都市iのx座標をcx[i]，y座標をcy[i]とする.

遺伝子型：0〜L-1までの整数値を重複を許さずにL個並べたリストを遺伝子列とする.

表現型；リストの先頭から書かれた番号の都市を順に回るとして巡回路をつくる．この場合も遺伝子型のリストと結局同じなので，そのままコピー.



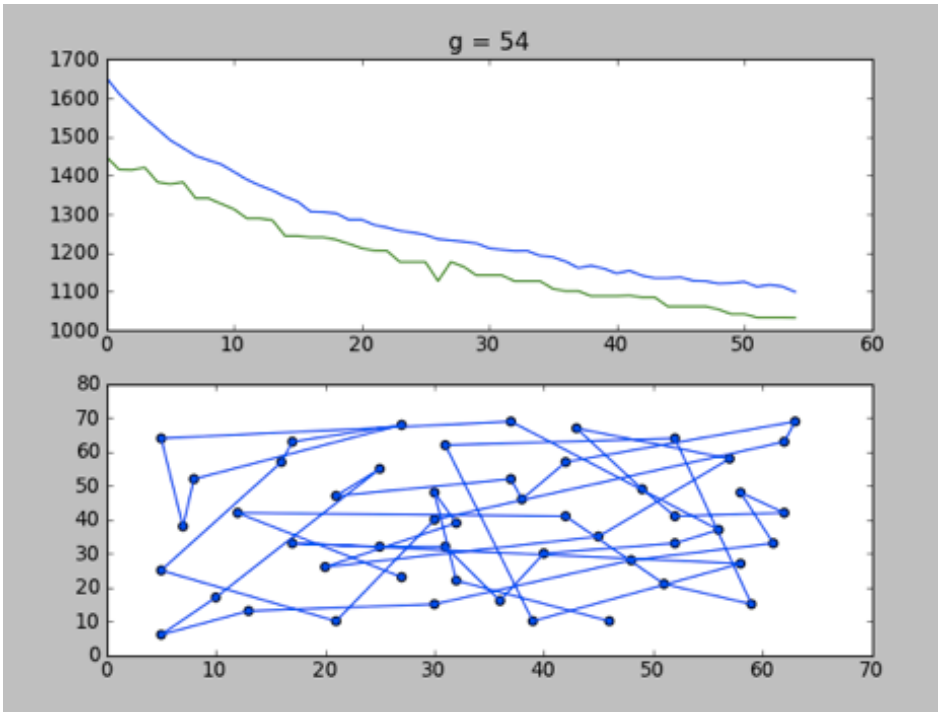
適応度：表現型の巡回路上の都市間のユークリッド距離の合計を適応度とする（ただし適応度が小さいほど高評価とみなす）.

突然変異（広い意味で）：
一番単純なのは，ある遺伝子座と別の遺伝子座の値を入れ換えること.
ほかには，ある一部分を別の場所の移動する挿入とか，遺伝情報の制約を満たす範囲でいろいろ考えられます.

交叉：とりあえずなし（後で説明）.

選択：上の適応度のままでは距離の長い方が適応度が高いので，例えば小さい方からランク選択で選ぶようにする.

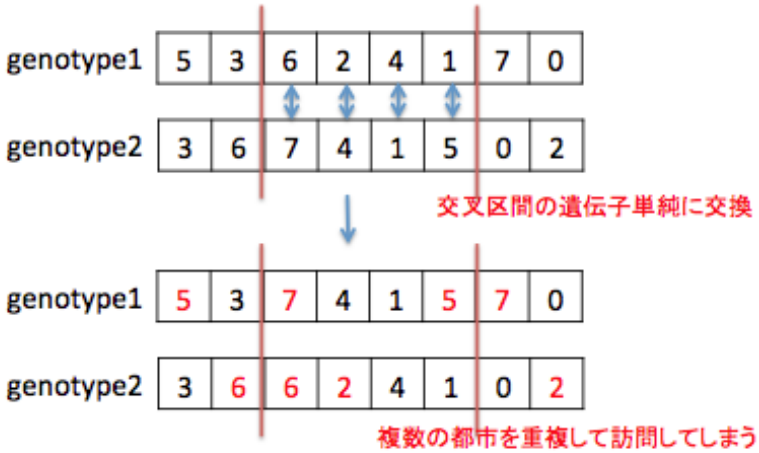
グラフ表示：たとえば，現在の集団の適応度の平均と最大値の推移のグラフに加え，都市の配置を表した散布図に，各世代での現在の最適解を重ね合わせたものなど.



巡回セールスマン問題における交叉

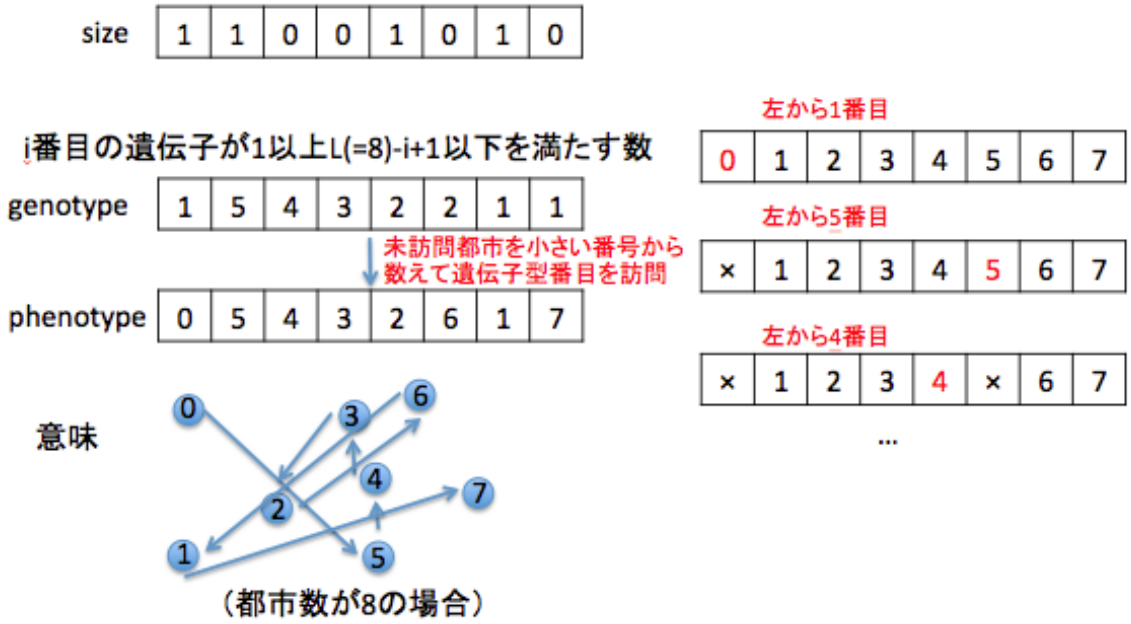
問題点

上の定義で，遺伝子は”0〜L-1までの整数値を重複を許さずにL個並べたリスト”と定義しました．この場合，前回やったような単純な一点交叉などを行うと，整数値が重複し，遺伝子表現の定義から外れてしまうことになり，問題です．



順序表現

このような問題を解決する方法に，順序表現，つまり，i番目の遺伝子の値域を1以上L-i+1以下の整数とし，各遺伝子は，まだ選択されていない都市のうちで小さい方から何番目をとるかを示すという表現法がよく知られています．この方法なら，どのような個体間で交叉が行われても，きちんと巡回路をつくる ことができます．



しかし，その一方で，交叉を開始した位置以降で，実際の経路が保持されないなどの問題もあります．

Partially-mapped crossover (PMX)

ここでは，Partially-mapped crossover (PMX) (Goldberg and Lingle, 1985)というもう一つのよく知られた基本的な交叉手法を紹介します．
以下のような2つの遺伝子列があったとして，PMXは以下の操作を行います．

この方式の特徴は，交叉の区間の巡回路の構造が維持されることです．一方，それ以外の部分は破壊されてしまうデメリットもあります．

様々な交叉法が提案され，巡回セールスマン問題のようなベンチマーク問題を使って，その性能

が競われています.

練習

これまでのプログラム改変して以下のトピックのうち一つを選んでモデルを実装しなさい.

1. 先に挙げたアルゴリズムのバリエーション（選択の仕方や遺伝子の表現方法など）を一つ実装し，f(x)=x^2以外の（単純でない）関数やベンチマー ク関数の最適化に適用する.
2. ナップザック問題を解くGAを実装する.
3. 巡回セールスマン問題を解くGAを実装する．順序表現と交叉を実装したり，PMXなどを実装するとか.
4. その他，各自で最適化問題を考えたり，調べたりして，それを解くGAを実装する.

２）１で実装したモデルについて，GAや問題設定に関するパラメータを一つ選んで，それの解探索過程への影響を端的に示したグラフを作成し，その理由を考察しなさ い.