기관차 자동주행 시뮬레이션

기관차는 역을 올바른 방향으로 출발해서 다음 정거장에 정확히 정지해야 한다.

다음 역에 정확하게 도착할 수록, 더 빨리 도착할수록 뛰어난 주행이다

이러한 기차의 가속과 감속을 어떻게 제어하여야 다음정거장에 정확히 도착할지 GA을 통해 예측해 보았습니다

GA\_시스템 설계

1. 크로모즘  
   열차가 위치한 구간(0~119)에 따라 엔진을 가속할 것인지   
   관성으로 등속 주행할 것인지  
   브레이크로 감속할 것인지  
   -2, -1, 0, 1, 2 사이의 정수로 가속도 값을 조합하여 크로모즘을 구현하였다.  
   곧 5가지 값을 120개의 구간별로 지정하므로 각 자리별 5가지 경우 총 120자리의 크로모즘을 구현하였다.
2. 시뮬레이션 코드  
   크로모즘에 따라 열차의 속도를 다음과 같은 물리적분식을 통하여 계산하였다.  
     
   velocity = velocity + acceleration[n,section]\*dt

#최대 속력 제한 조건

if velocity > 300:

velocity = 300

#후진 제한

if velocity < -300:

velocity = -300

position = position + velocity\*dt #+ 0.5\*acceleration[n,section]\*dt\*dt

1. 적합도 평가  
   열차의 도착여부 조건만으로는 원하는 방향으로 진화를 이끌어 가기 어려울 것으로 생각되어 3가지 평가 기준을 두어 중요한 것 에 가중치를 크게 주는 방법으로 각각의 평가 기준의 도달 목표를 순차적으로 달성 할 수 있도록 유도 하였다.  
     
   제1 평가 기준  
   가중치 : 1  
   열차의 최대 이동거리와 도착점 사이의 오차  
   최대값과 도착점이 가까울수록 유리  
   제2 평가 기준  
   가중치 : 10  
   도착점에서의 감속 정도  
   정차역에서 속력이 작을수록 유리  
   제3 평가 기준  
   가중치 : 10000  
   도착여부 및 도달 시간  
   도착 못한 경우 400부터 0까지 도달 시간이 작을수록 유리

요약

1차 일단 도달하는 객들이 살아남기 적합한 조건

2차 1차 세대 중에서 정확히 도달하는 객체들이 살아남기 적합한 조건

3차 빠르게 도달하는 객체들이 살아남기 적합한 조건

최종 평가식  
(target\_position - abs(target-max\_position)) - 10\*math.sqrt(end\_velocity\*\*2)- 10000\*time\_elapsed  
  
가중치를 통해 일단 제대로 정차하는 개체가 생존에 압도적으로 유리하도록 적합도 평가식을 설계하였다.

1. 적자생존자 선택(부모 세대)  
   직접 구현한 물리 시뮬레이터로 매번 평가하여서 열차의 이동하는 모습을 계산하고 적합도 평가가 가장 높은 2개체(individual)를 선택하였다.   
   적합도 평가식의 구조상 적합도가 음수가 될 수도 있어서 전체에 대한 개체의 적합도 값을 0~1사이의 값을 통한 비율로 나타내지는 못하였다.
2. 재생산  
   교차와 변이를 모두 활용하였으며  
   가장 적합하여 부모세대로 선택된 2개체는 다음세대에도 유지되도록 하였다.  
   총인구수도 변수로 두어 조절이 가능하도록 하였고 학습 속도, 유전자 다양성 등을 조절하는데 활용하였다.  
     
   교차는 threshold가 넘어가는 곳에서 교차가 일어나는 방법을 채택하였다  
   변이도 크로모즘과 같은 길이의 0~1 랜덤 변수값을 갖는 배열을 주어 이 배열이 threshold를 넘어 가는 부분에서 변이가 일어나게 하는 방법을 채택하였다.

코드

import numpy as np

import math

import matplotlib.pyplot as plt

import random

#fitted\_individual

def elvaluater(target, position, end\_velocity, time\_elapsed):

max\_position = np.max(position)

return (target - abs(target-max\_position)) - 10\*math.sqrt(end\_velocity\*\*2)- 10000\*time\_elapsed

def simulater(population):

acceleration = population

target = 10000

dt = 1

position\_0 = 0

velocity\_0 = 0

position\_data = np.zeros((acceleration.shape[0],400))

velocity\_data = np.zeros((acceleration.shape[0],1))

time\_elapsed\_data = np.zeros((acceleration.shape[0],1))

section = 0#구간 은 항 상 정수여야 하므로 다른 변수를 만들어 관리 하도록 함

n = 0

time\_elapsed = 0

while (n < acceleration.shape[0]):

position = position\_0

velocity = velocity\_0

j = 0

while j < 400:

velocity = velocity + acceleration[n,section]\*dt

#최대 속력 제한 조건

if velocity > 300:

velocity = 300

#후진 제한

if velocity < -300:

velocity = -300

position = position + velocity\*dt #+ 0.5\*acceleration[n,section]\*dt\*dt

position\_data[n, j] = position

section = math.floor(0.01\*position)

#크로모즘 초과 방지 조건

if section > 119:

section = 119

if section < 0:

section = 0

if abs(position - target) < 100 and abs(velocity) < 0.1:

velocity = 0

break;#개체별 반복을 끝냄

#print(j, time\_elapsed, position, velocity, acceleration[n,section], section)

j = j+1

time\_elapsed = dt\*j

print(n, target, np.max(position\_data[n, : ]), velocity, time\_elapsed)

#print(elvaluater(target, position\_data[n, : ], velocity, time\_elapsed))

#plt.plot(position\_data[n, : ])

#plt.ylabel('postion')

#plt.show()

position\_data[n] = position\_data[n, : ]

velocity\_data[n] = velocity

time\_elapsed\_data[n] = time\_elapsed

n = n+1

return target, position\_data, velocity\_data, time\_elapsed\_data

#가속도 외부에서 주어짐 위치 초기값을 통해 결정 됨

#속도 가속도를 통해 계산됨

#위치 속도와 가속도를 통해 계산됨

def simulater\_with\_plt(population):

target, position\_data, velocity\_data, time\_elapsed\_data = simulater(population)

for n in range(population.shape[0]):

plt.plot(position\_data[n, : ])

plt.ylabel('postion')

plt.show()

def GA\_get\_fitted\_index(population):

value = list(range(population.shape[0]))

print(value)

target , position, velocity, time\_elapsed = simulater(population)

for i in range(population.shape[0]):

value[i] = elvaluater(target, position[i, : ], velocity[i, 0], time\_elapsed[i, 0])

rank = value[:]#얕은 복사

rank.sort(reverse = 1)

print(rank)

print(value)

print(value.index(rank[0]), value.index(rank[1]))

return value.index(rank[0]), value.index(rank[1])

"""

def GA\_reproduction\_2(population, index1, index2, Number\_pop):

chromosome\_1 = population[index1, :]

chromosome\_2 = population[index2, :]

next\_generation\_pop = np.zeros((population.shape[0], 120))

next\_generation\_pop[0] = chromosome\_1

next\_generation\_pop[1] = chromosome\_2

if population.shape[0]%2 ==0:

for i in range(2, population.shape[0]):

GA\_crossover(chromosome\_1, chromosome\_2, 0.8)

"""

def GA\_reproduction\_1(population, index1, index2, Number\_pop):

chromosome\_1 = population[index1, :]

chromosome\_2 = population[index2, :]

next\_generation\_pop = np.zeros((population.shape[0], 120))

next\_generation\_pop[0] = chromosome\_1[:]

next\_generation\_pop[1] = chromosome\_2[:]

#print("#1", chromosome\_1[:])

#print("#2",GA\_crossover(chromosome\_1[:], chromosome\_2[:], 0.8))

for i in range(2, population.shape[0]):

temp = GA\_crossover(chromosome\_1[:], chromosome\_2[:], 0.8)

#print("#4", temp)

#print("#5", np.array(temp)[0,:])

next\_generation\_pop[i] = np.array(temp)[0,:]

#print("#6", next\_generation\_pop)

return next\_generation\_pop

def GA\_crossover(chromosome\_1, chromosome\_2, threshold\_crossover):

cross = False

rand\_choice = random.random()

new\_chromosome\_1 = [0 for i in range(len(chromosome\_1))]

#new\_chromosome\_1 = np.zeros((1, 120))

#new\_chromosome\_2 = np.zeros((1, 120))

#print("#2", new\_chromosome\_1)

#print("#3", chromosome\_1)

for i in range(len(chromosome\_1)):

cross\_value = np.random.random((1,120))

if cross\_value[0, i] > 0.8:

if cross == False:

cross = True

else:

cross = False

#print(i, new\_chromosome\_1)

if random.random() >= 0.5:

if cross:

new\_chromosome\_1[i] = chromosome\_1[i]

#new\_chromosome\_2[i] = chromosome\_2[i]

else:

new\_chromosome\_1[i] = chromosome\_2[i]

#new\_chromosome\_2[i] = chromosome\_1[i]

else:

if cross:

new\_chromosome\_1[i] = chromosome\_2[i]

#new\_chromosome\_2[i] = chromosome\_1[i]

else:

new\_chromosome\_1[i] = chromosome\_1[i]

#new\_chromosome\_2[i] = chromosome\_2[i]

return new\_chromosome\_1, #new\_chromosome\_2

"""

if rand\_choice >= 0.5:

if cross:

new\_chromosome\_1[0, i] = chromosome\_1[0, i]

#new\_chromosome\_2[0, i] = chromosome\_2[0, i]

else:

new\_chromosome\_1[0, i] = chromosome\_2[0, i]

#new\_chromosome\_2[0, i] = chromosome\_1[0, i]

else:

if cross:

new\_chromosome\_1[0, i] = chromosome\_2[0, i]

#new\_chromosome\_2[0, i] = chromosome\_1[0, i]

else:

new\_chromosome\_1[0, i] = chromosome\_1[0, i]

#new\_chromosome\_2[0, i] = chromosome\_2[0, i]

"""

def GA\_mutate(next\_generation\_pop, NPOP\_threshold, NCHO\_threshold):

mutate\_choice\_value = np.random.random(population.shape[0])

for i in range(2, population.shape[0]):

if mutate\_choice\_value[i] > NPOP\_threshold:

for j in range(population.shape[1]):

mutate\_index\_value = np.random.random(population.shape[1])

if mutate\_index\_value[j] > NCHO\_threshold:

next\_generation\_pop[i, j] = np.random.randint(-2, 3)

return next\_generation\_pop

def GA\_system(population):

index1, index2 = GA\_get\_fitted\_index(population)

next\_generation\_pop = GA\_reproduction\_1(population, index1, index2, population.shape[0])

next\_generation\_pop\_mutate = GA\_mutate(next\_generation\_pop, 1, 0.7)

print("#7", next\_generation\_pop)

print("#8", next\_generation\_pop\_mutate)

return next\_generation\_pop\_mutate

def GA\_system\_iter(population, iteration):

n = 0

while(n < iteration):

population = GA\_system(population)

#simulater(population)

n = n + 1

return population

population = np.random.randint(-2, 3, size=(40, 120))#인구 랜덤 생성 40개 크로모즘크기 : 100

#simulater\_with\_plt(population)

population = GA\_system\_iter(population, 100)

#simulater\_with\_plt(population)

population = GA\_system\_iter(population, 100)

simulater\_with\_plt(population)

population = GA\_system\_iter(population, 100)

simulater\_with\_plt(population)

구현결과 테스트

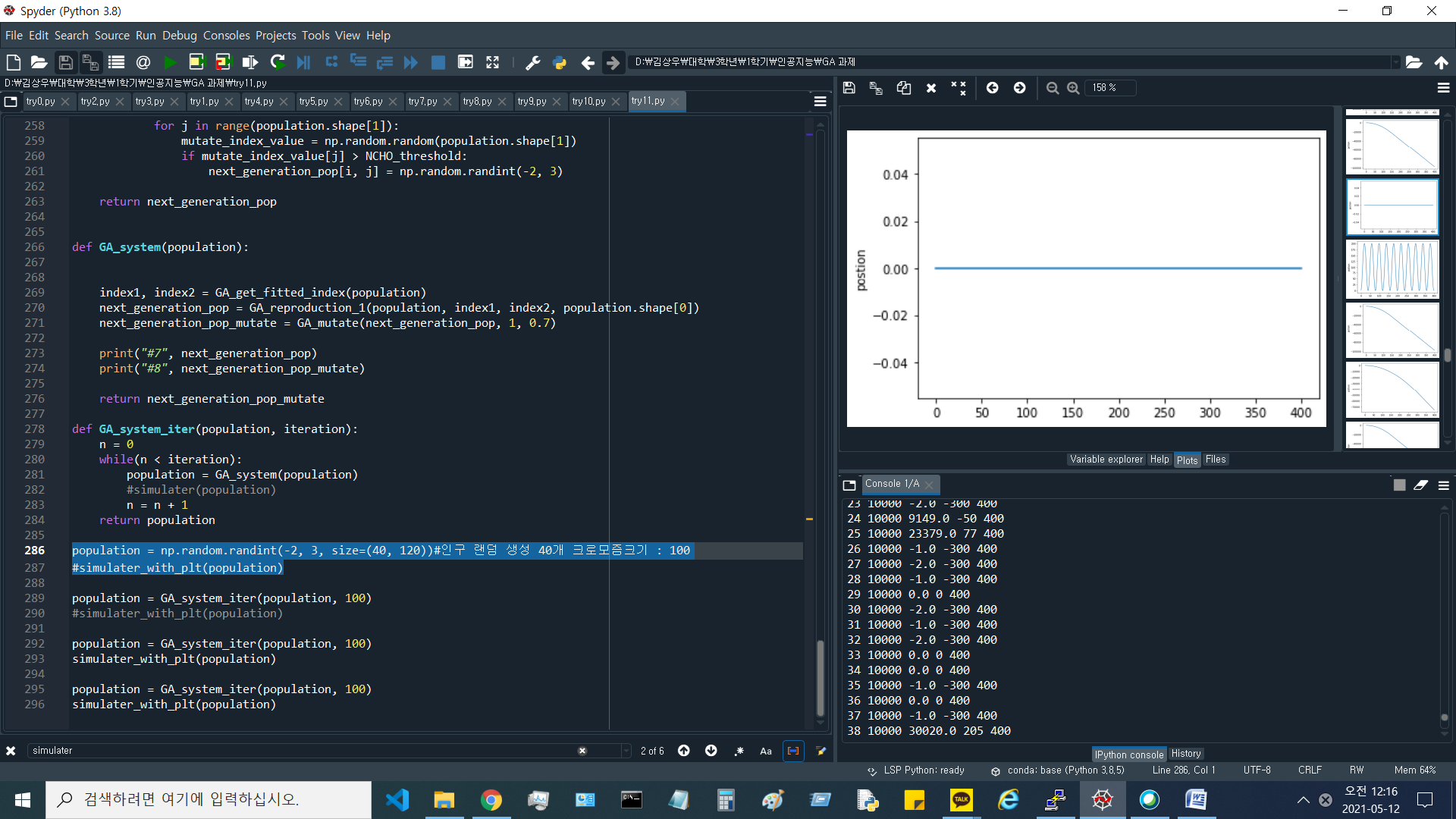
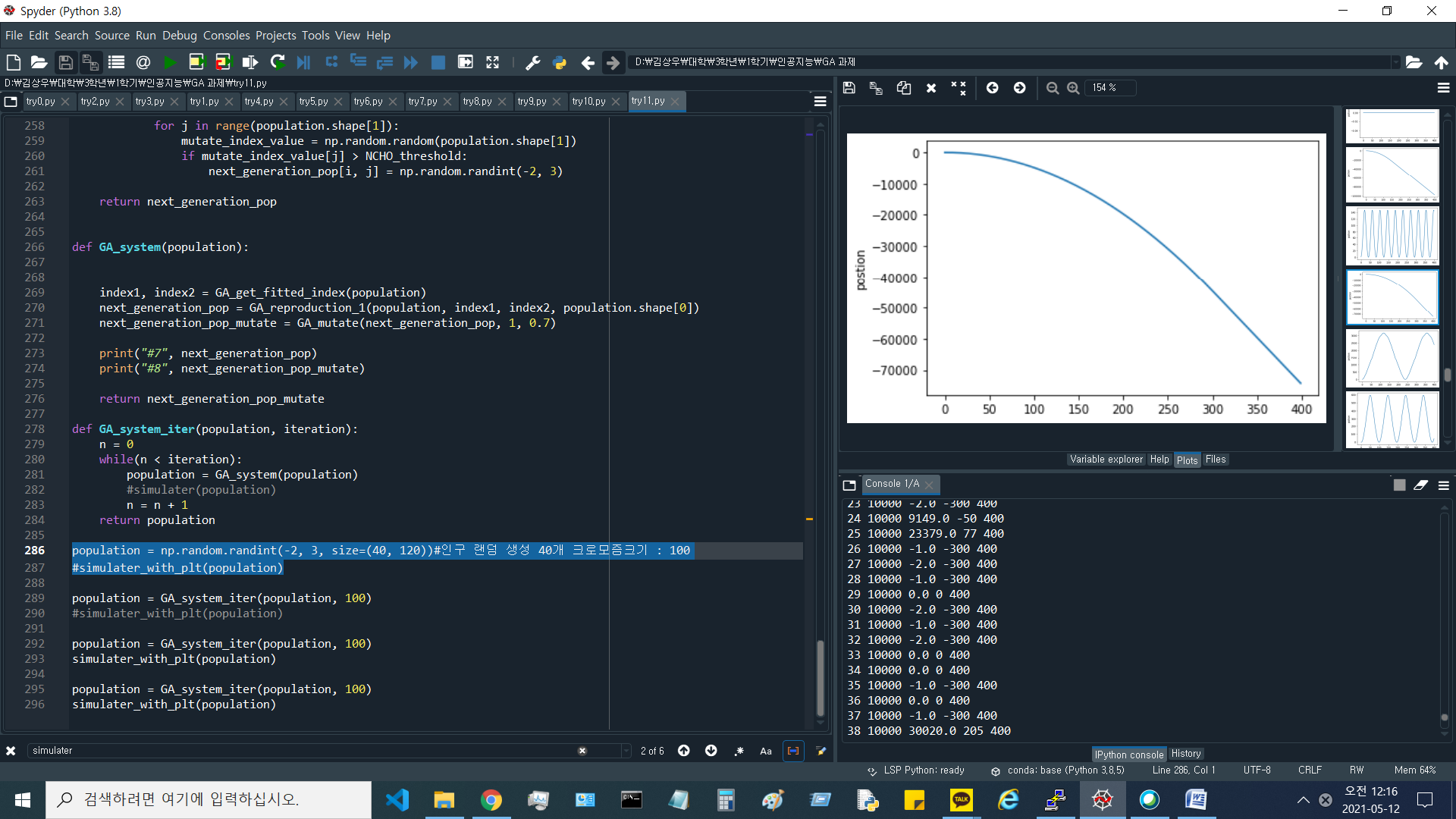
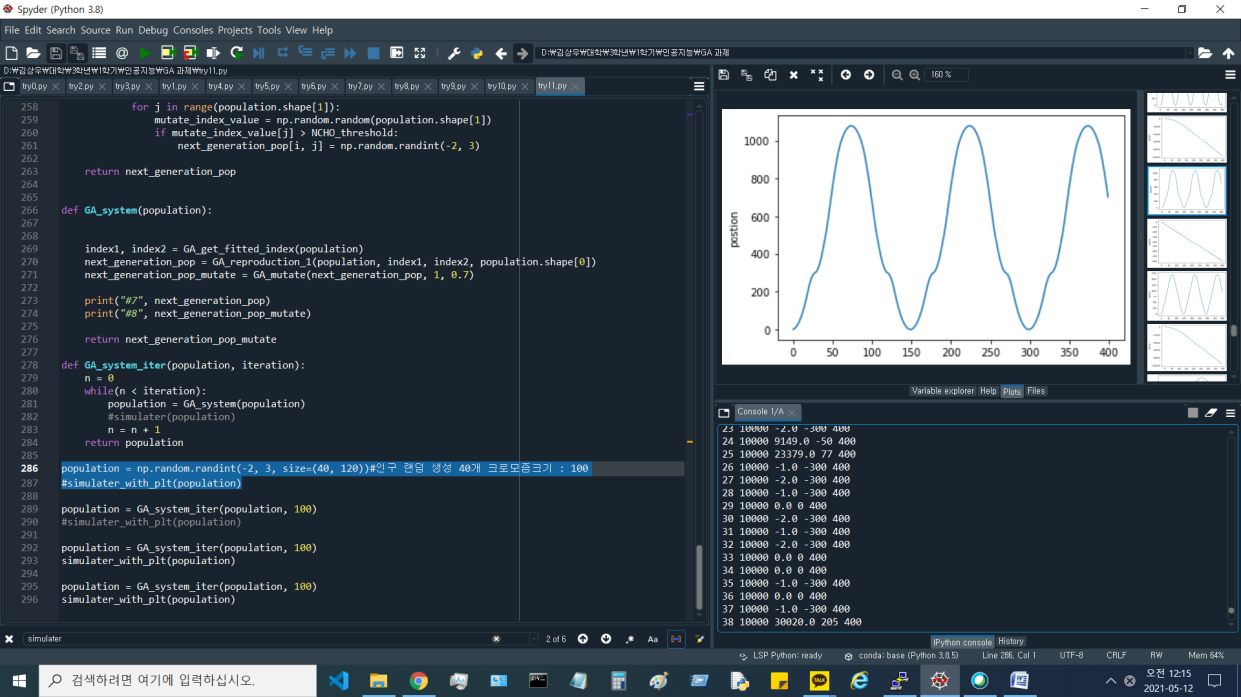
첫 랜덤 생성 인구에서는

열차가 가야할 방햐의 반대로가서 위치가 –로 나오는 경우도 많고

애초에 달리지 않는 경우도 많다.

그래프로 나타내면 우하향 그래프나 값이 0인 상수함수 X축과 곂치는 경우로 각각 나타난다.

그 이외에도 가다가 다시 돌아오는 진동하는 그래프 정차역을 지나 발산하는 그래프등 모습이 다양하다.



그래도 이중에서 가장 멀리가는 유전자들만 살려서 적자생존 선택을

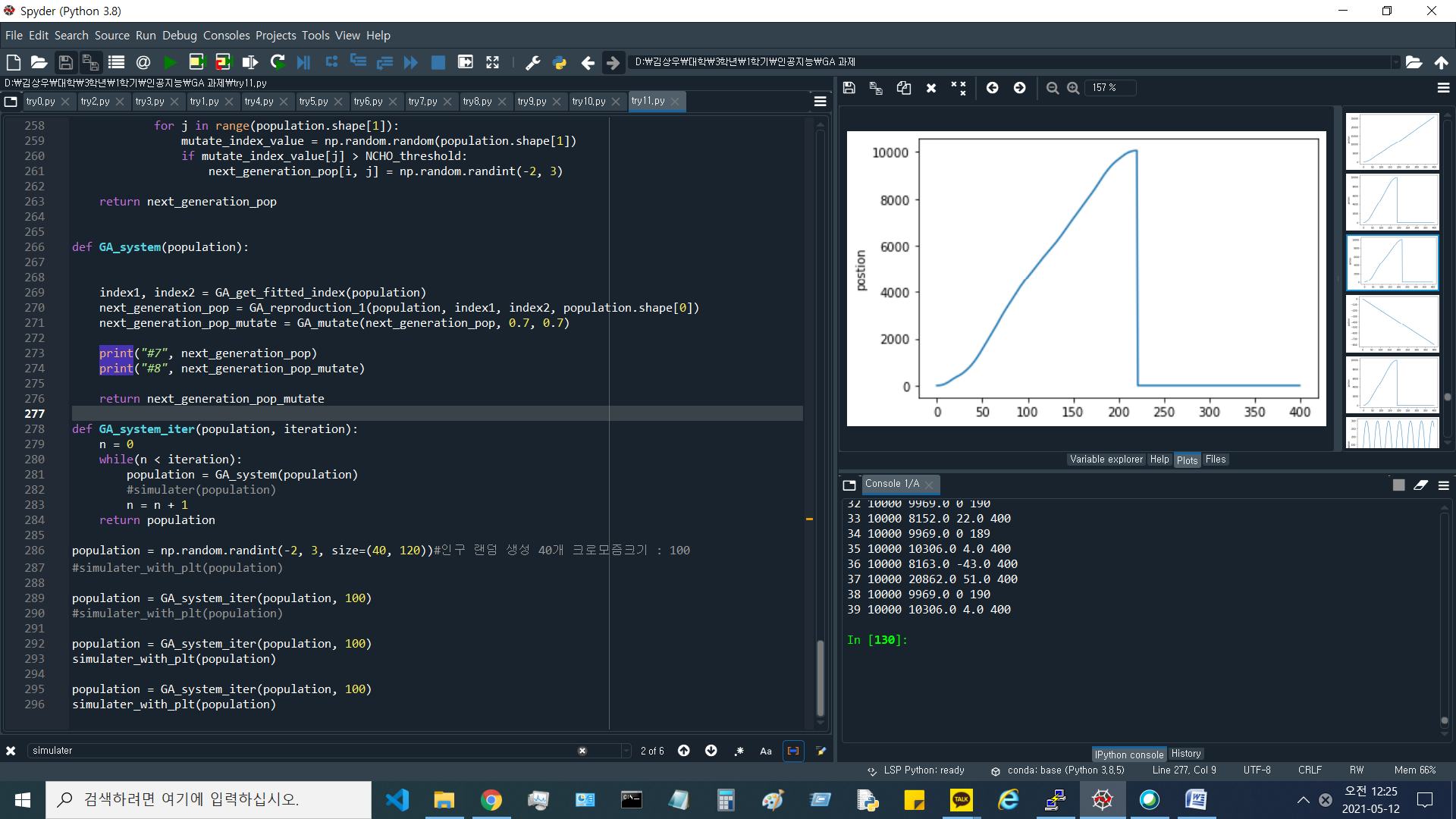
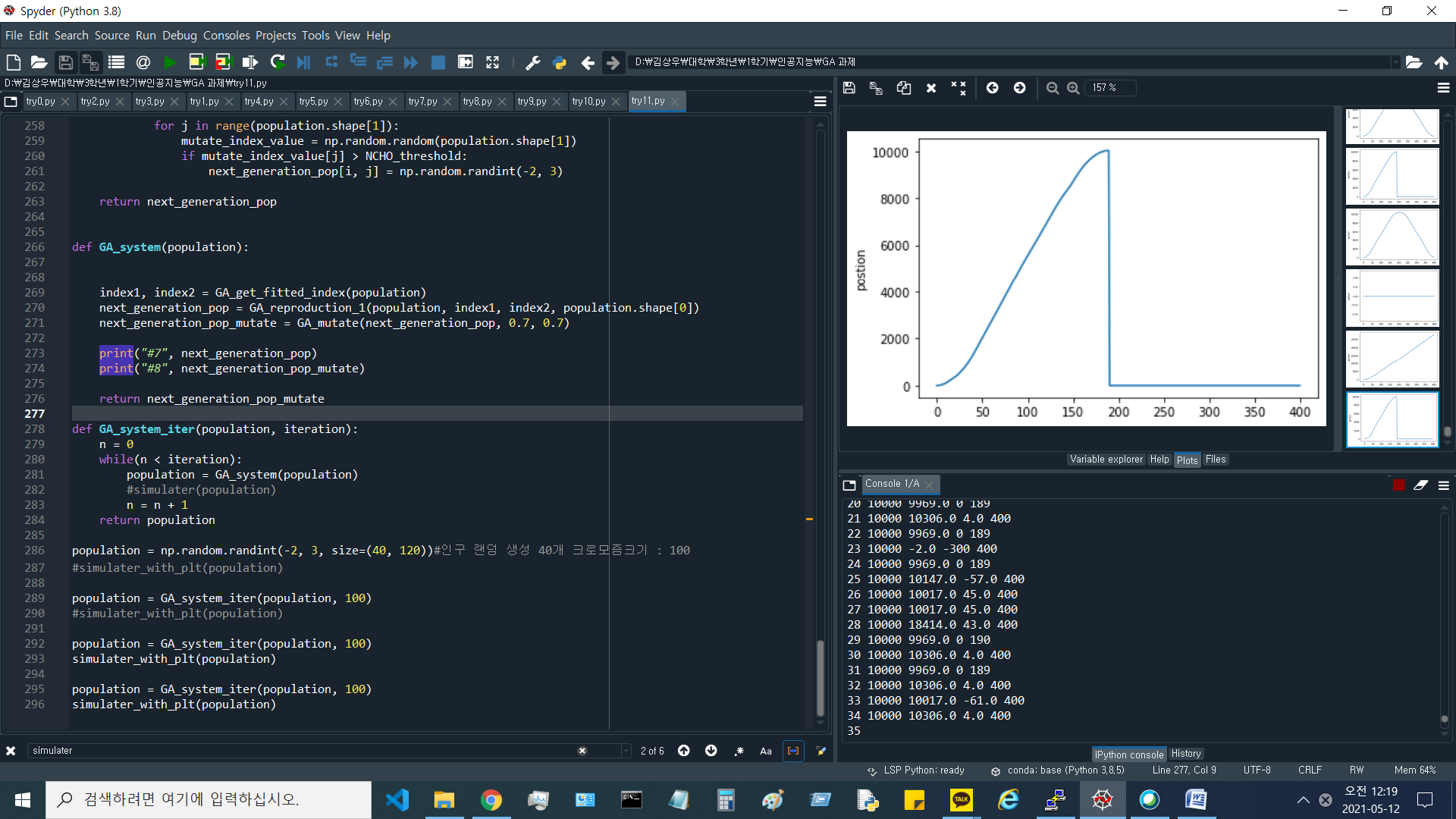
인구수 40에서 200회 정도로 적절히 반복하면 10개중에 3개 정도 꼴로 역에 정확히 도착하는 개체들이 발생한다.

반복수가 많아질수록 정확히 도달하는 개체들이 증가한다.

개체들의 도착여부 확인은

도착점 이후부터 기록을 하지 않으므로 정확하게 도착하면 그래프가 불연속이 되는 부분이 생기는지 확인하여 판별할 수 있다.

도착시간이 다르지만 도착조건을 만족하는 그래프들 10000지점에 100의 오차 이내로 정차한다.



또한 정확히 도착하는 유전자들이 만족하는 평가기준에 굉장히 큰 가중치를 부여함으로써

도착하지 못한 개체의 적합도 평가와 도착한 개체의 적합도가 매우 큰 차이가 나도록 하였다.

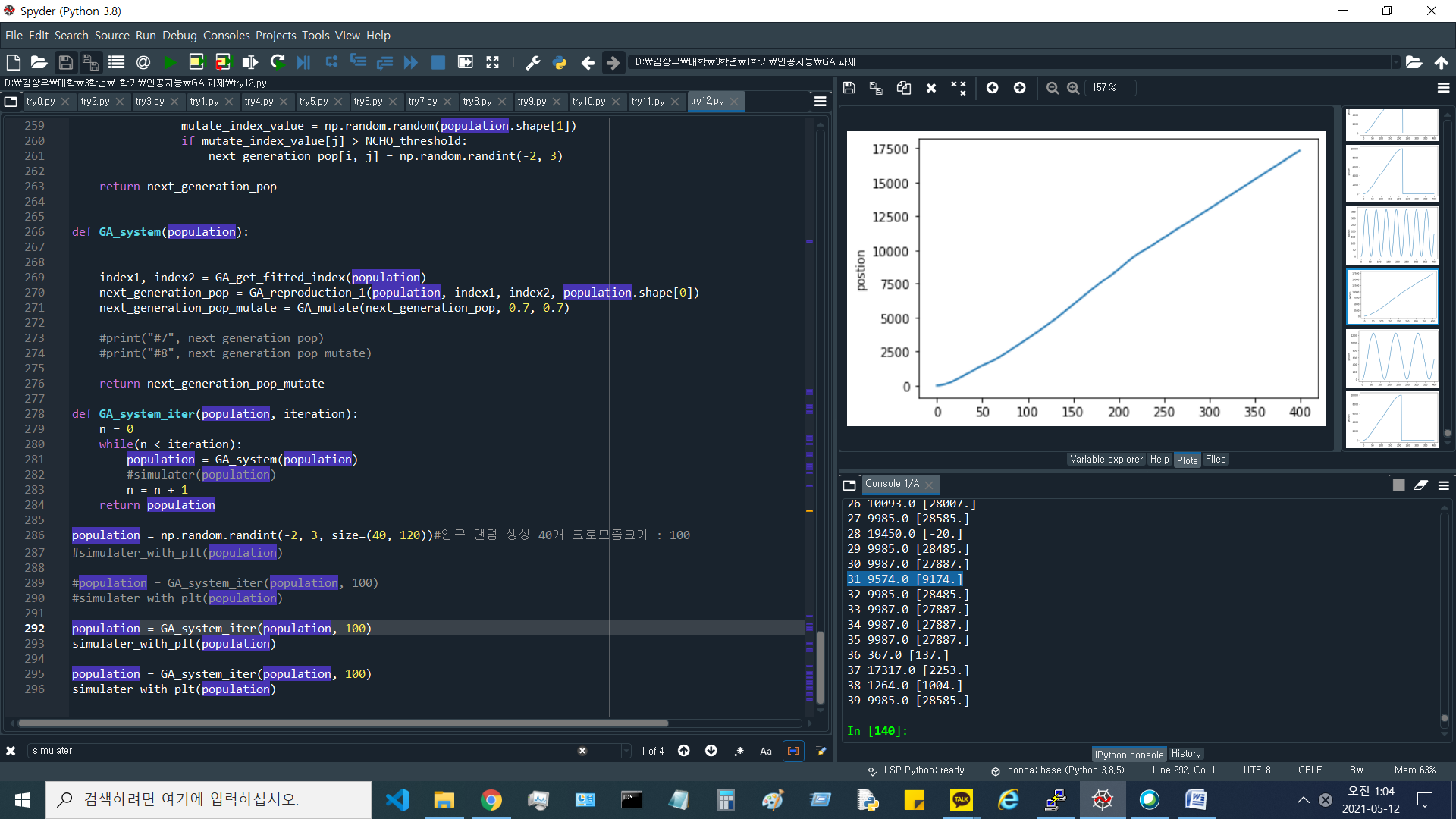
이를 통해 일단 도착하는 개체들의 적합도와 생존율을 극대화하였다.

번호 도달거리 점수

CASE : 1

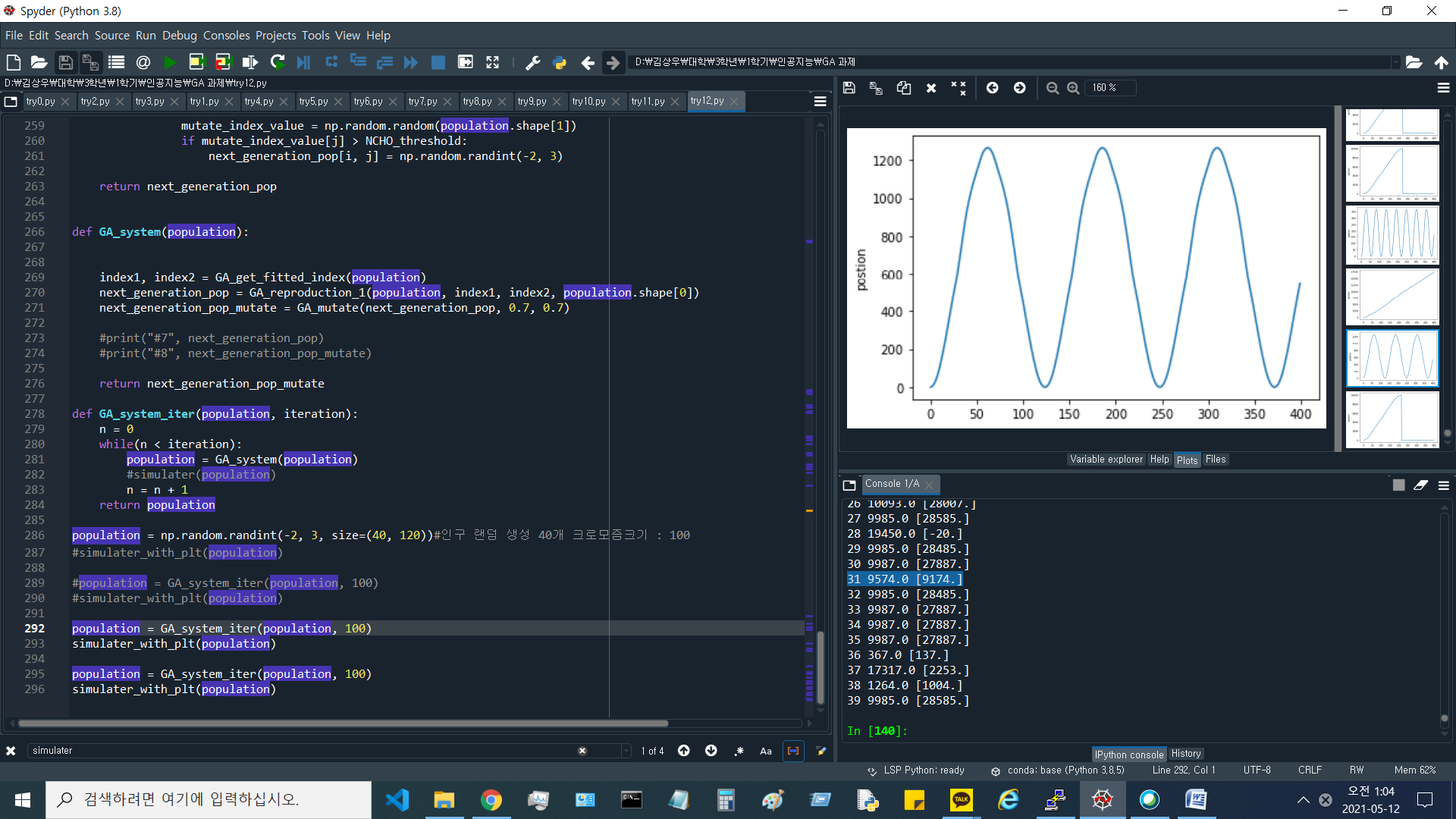
완전 부적합(무정차 통과 발산)

37 17317.0 [2253.]



완전 부적합(목표 도달 불가 진동)

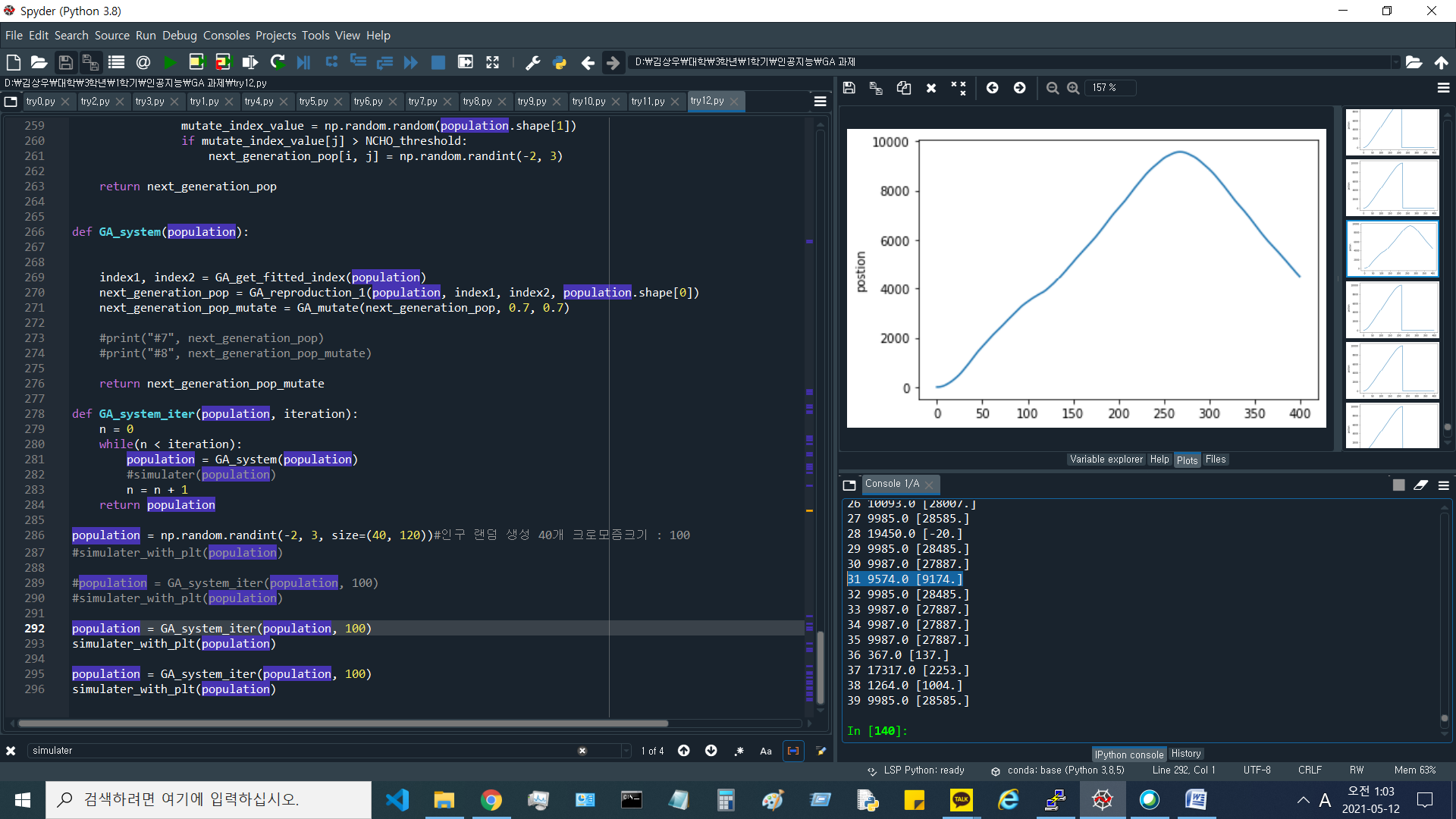
38 1264.0 [1004.]



CASE : 2

1차조건 적합

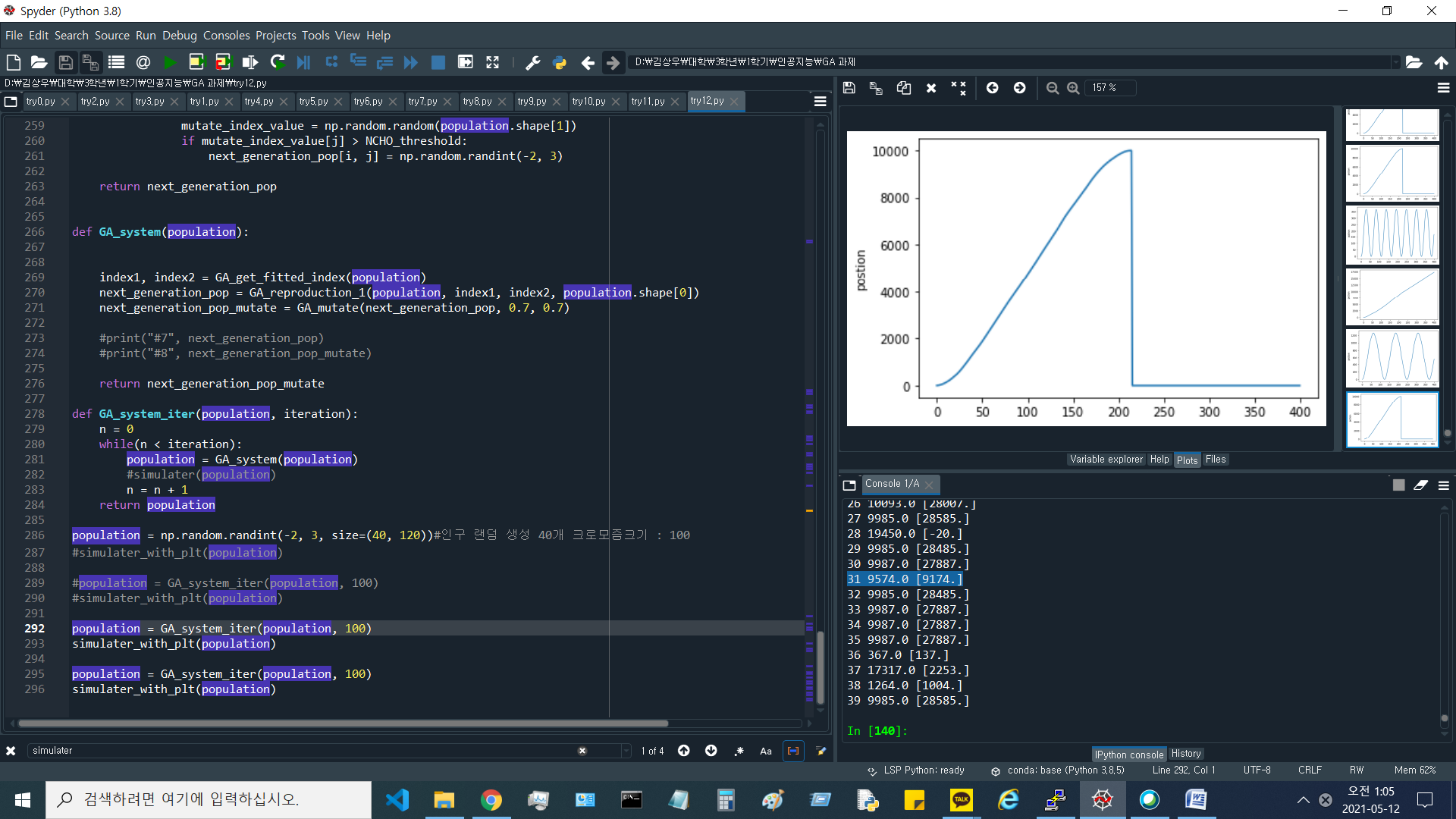
31 9574.0 [9174.]



CASE : 3

2차조건 적합

39 9985.0 [28585.]



이처럼 조건에 따라 적합도 접수가 양자적으로 널뛰도록 적합도 함수를 설계하여 원하는 유전자를 가진 개체의 생존 확률을 대폭 증가시켰다.

이처럼 유전자 알고리즘이 제대로 작동하여 여러 번 반복하면, 적절한 크로모즘을 선별하여 정상적으로 열차를 운행하며 물리이론적 해답과도 상통하는 개체를 선별할 수 있었다.

인구수에 따른 영향

인구수를 40정도로 설정하고 GA를 작동시키면 대부분의 경우 최적값을 찾아내지만

인구수가 10 내외로 적으면 100번 이하의 적은 반복 수 만으로도 유전자가 1종류만 남아버려서 종 다양성이 쉽게 상실되는 상황을 확인하였다.

변이가 있는 경우와 없는 경우

이처럼 단순한 시스템에서는 때때로 변이를 주지 않아도 적절한 값을 찾을 수 있었다.

오히려 변이를 없앰으로써 수렴 양상이 확실해지는 효과가 나타났다. 로컬미니멈에 도달하는 개체들이 더 많아 지는 것이다. 다만 세대를 똑같이 300번 반복하여도

목표값에는 근처 도달할 수 있을지언정 모든 개체가 목표값에 오차 10/10000 이내로 정확하게 도달하는 개체는 단 하나도 없는 상황을 확인 하였다.

0 10000 9761.0 -43.0 400

1 10000 9571.0 -44.0 400

2 10000 9761.0 -43.0 400

3 10000 9571.0 -44.0 400

4 10000 9761.0 -43.0 400

…

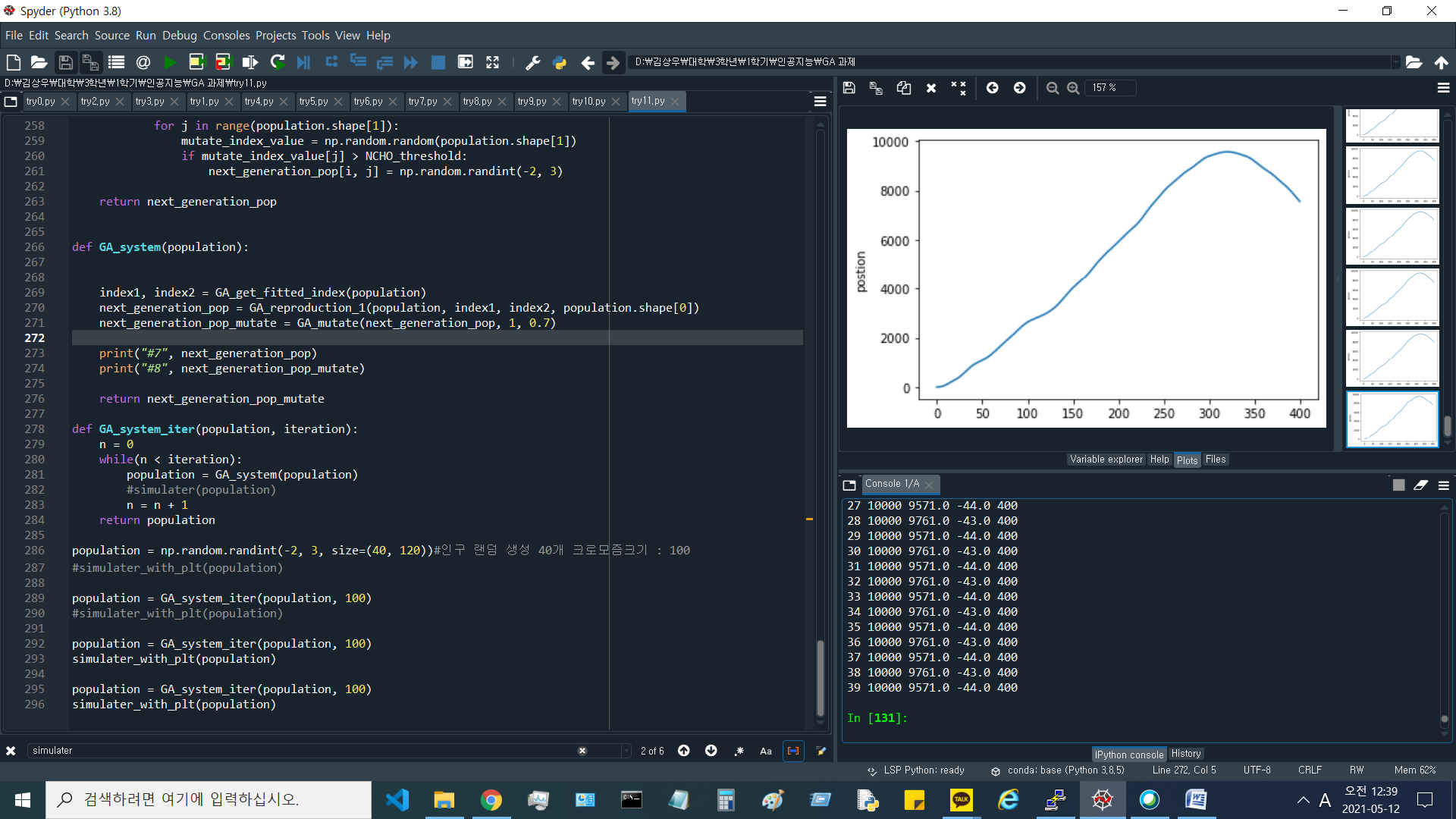
35 10000 9571.0 -44.0 400

36 10000 9761.0 -43.0 400

37 10000 9571.0 -44.0 400

38 10000 9761.0 -43.0 400

39 10000 9571.0 -44.0 400



느낀점

유전자 알고리즘에 대하여 간단히 검색하며 다른 제어분야에서도 사용 될 수도 있겠다라는 생각이 실제 연구 현장에서 이미 많이 활용된다는 것을 알게 되었으며 차후 흥미를 느꼈던 PID 제어 분야에 대한 자료를 찾아보고 싶습니다.

또한 자동 PID 드론의 원리가 유전자 알고리즘과 연관이 있는지도 알아보고 싶습니다

적합도 함수 구하는 방법에 관심

-201933914 김상우