Computational Intelligence for the Internet of Things

Lab 12 - Exercise 5

Daniel Lopes
METI, 90590
Group 4

Department of Electrical Engineering,
Computer Engineering & Informatics

June 25, 2022

1 Objective

The objective of this lab was to implement GA and PSO models that solved the function described by:

$$Z1 = \sqrt{X1^2 + X2^2}$$

$$X2 = \sqrt{(X1 - 1)^2 + (X2 + 1)^2}$$

$$f1 = (sin(4 * Z1)/Z1) + (sin(2.5 * Z2)/Z2)$$

1.1 Theoretical Value

This problem has a particularity which is the possibility of plotting its graph to try to visualize what the objective is. Figure 1 shows us that there is a maximum around 3.5 which we're not sure about. That's what these algorithms are going to try to figure out using probabilistic searching. This graph also gives us an idea of which values should be considered when trying to maximize the function for the given X1 and X2 inputs.

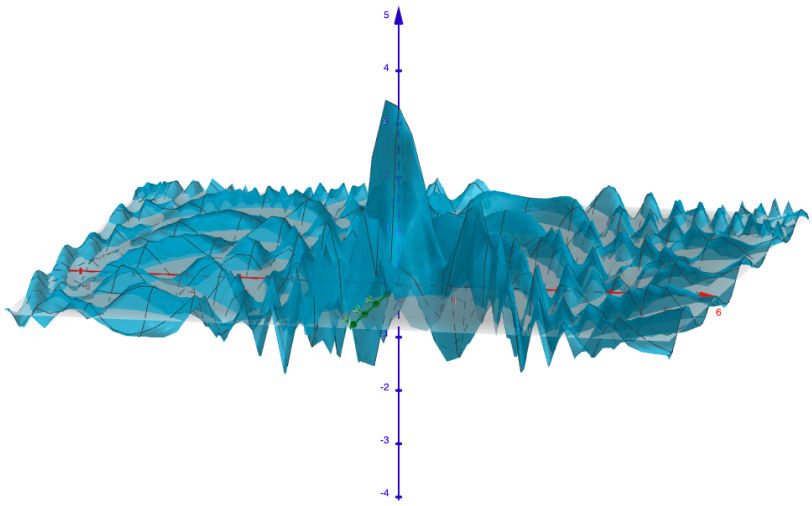

Figure 1: F1 graph

1.2 Fitness Function

The following Listing 1 contains the code that describes the functions that were implemented in order to calculate f1:

```python
from math import sin, sqrt

def z1(x1, x2):
    return sqrt(x1**2 + x2**2)

def z2(x1, x2):
    return sqrt((x1-1)**2 + (x2+1)**2)

def f1(individual):
    x1 = individual[0]
    x2 = individual[1]
    _z1 = z1(x1, x2)
    _z2 = z2(x1, x2)
    return (sin(4*_z1)/_z1) + (sin(2.5*_z2)/_z2)
```

Listing 1: Python f1 function

2 Genetic Algorithms

To solve this problem using Genetic Algorithms and with the help of the *Deap* library the following section contains the code that was used.

2.1 Code Appendix

```python
def mutate(individual, indpb):
    prob = random.random()
    if prob < indpb:
        pos_to_mutate = random.randint(0, 1)
        individual[pos_to_mutate] += random.uniform(-1, 1)

creator.create("FitnessMax", base.Fitness, weights=(1.0, ))
creator.create("Individual", list, fitness=creator.FitnessMax)
toolbox = base.Toolbox()
toolbox.register("attr_float", random.uniform, 0.01, 10)
toolbox.register("individual", tools.initRepeat, creator.Individual,
                 toolbox.attr_float, 2)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", f1)
toolbox.register("mate", tools.cxOnePoint)
toolbox.register("mutate", mutate, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)
hof = tools.HallOfFame(1)
```

Listing 2: Python GA Mutation Function and Toolbox Initialization

```python
def main():
    random.seed(64)
    pop = toolbox.population(n=100)
    CXPB, MUTPB = 0.5, 0.2

    print("Start of evolution")
    fitnesses = list(map(toolbox.evaluate, pop))
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = [fit]

    print("  Evaluated %i individuals" % len(pop))
    g = 0
    while g < 100:
        g = g + 1
        print("-- Generation %i --" % g)
        offspring = toolbox.select(pop, len(pop))
        offspring = list(map(toolbox.clone, offspring))

        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < CXPB:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = list(map(toolbox.evaluate, invalid_ind))
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = [fit]

        print("  Evaluated %i individuals" % len(invalid_ind))
        pop[:] = offspring

        hof.update(pop)

    print(f"-- End of (successful) evolution (Generations: {g}) --")

    best_ind = hof[0]
    fitness = f1(best_ind)
    print("Best individual is %s: %s" % (best_ind, fitness))


main()
```

Listing 3: Python GA Main function for iterating over generations

2.2 Results

Running this code for 100 generations, the best individual for this genetic algorithm was: [0.062053384633855735, -0.05996977896976613]

Where X1 = 0.062053384633855735, X2 = -0.05996977896976613 and the max value found for the f1 function was 3.787526866047997.

3 Particle Swarm Optimization

To solve this problem using PSO and with the help of the *Deap* library the following section contains the code that was used.

3.1 Code Appendix

```
import operator
import random
import numpy
import math
from deap import base
from deap import tools
from deap import creator
creator.create("FitnessMax", base.Fitness, weights=(1.0, ))
creator.create("Particle",list,fitness=creator.FitnessMax,speed=list,smin=None, smax=None,
        best=None)

def generate(size, pmin, pmax, smin, smax):
    part = creator.Particle(random.uniform(pmin, pmax) for _ in range(size))
    part.speed = [random.uniform(smin, smax) for _ in range(size)]
    part.smin = smin
    part.smax = smax
    return part

def updateParticle(part, best, phi1, phi2):
    u1 = (random.uniform(0, phi1) for _ in range(len(part)))
    u2 = (random.uniform(0, phi2) for _ in range(len(part)))
    v_u1 = map(operator.mul, u1, map(operator.sub, part.best, part))
    v_u2 = map(operator.mul, u2, map(operator.sub, best, part))
    part.speed = list(map(operator.add, part.speed, map(operator.add, v_u1, v_u2)))
    for i, speed in enumerate(part.speed):
        if abs(speed) < part.smin:
            part.speed[i] = math.copysign(part.smin, speed)
        elif abs(speed) > part.smax:
            part.speed[i] = math.copysign(part.smax, speed)
    part[:] = list(map(operator.add, part, part.speed))
```

Listing 4: Python PSO Part 1 - Auxiliary Functions

```
toolbox = base.Toolbox()
toolbox.register("particle",generate,size=2,pmin=-6,pmax=6,smin=-3,smax=3)
toolbox.register("population", tools.initRepeat, list, toolbox.particle)
toolbox.register("update", updateParticle, phi1=2.0, phi2=2.0)
toolbox.register("evaluate", f1)


def main():
    pop = toolbox.population(n=100)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", numpy.mean)
    stats.register("std", numpy.std)
    stats.register("min", numpy.min)
    stats.register("max", numpy.max)

    logbook = tools.Logbook()
    logbook.header = ["gen", "evals"] + stats.fields

    GEN = 1000
    best = None

    for g in range(GEN):
        for part in pop:
            part.fitness.values = toolbox.evaluate(part)
            if not part.best or part.best.fitness < part.fitness:
                part.best = creator.Particle(part)
                part.best.fitness.values = part.fitness.values
            if not best or best.fitness < part.fitness:
                best = creator.Particle(part)
                best.fitness.values = part.fitness.values
        for part in pop:
            toolbox.update(part, best)

        logbook.record(gen=g, evals=len(pop), **stats.compile(pop))
        print(logbook.stream)

    return pop, logbook, best


pop, logbook, best = main()

fitness = f1(best)
print("Best individual is %s: %s" % (best, fitness))
```

Listing 5: Python PSO Part 2 - Main Function and toolbox setup

3.2 Results

Running this code for 1000 generations, the best individual for this genetic algorithm was: [0.05910360798507408, -0.05758265614708025]

Where X1 = 0.05910360798507408, X2 = -0.05758265614708025 and the max value found for the f1 function was 3.787667697138008.