Homework 8 Advanced Analytics and Metaheuristics

Group 8: Nicholas Jacob

April 24, 2024

1. Genetic Algorithm

   (a) Finalize Code

      i. To create the initial chromosomes, we kept it simple, generating a random number between -500 and 500. We used the global variable n as the number of dimensions.

```
def createChromosome(n):
    x = []
    for i in range(n):
      x.append(myPRNG.uniform(-500,500))
    return x
```

      ii. For mutation, we played with this quite a bit. Originally we randomly selected an index and the just reseeded it with a random. This seemed to work but we wanted more control over it so we introduced a new parameter that we could tweak, $mutationFactor$. We used this to change one of the indicies by a random, $r \in [-1, 1]$, with $x[i] = x[i] + r * mutationFactor$. We could then tweak this factor to get more or less mutation. We did check that it was inside the interval and if not, randomly reseed.

```
def mutate(x):

    if mutationRate > myPRNG.random():
      i = myPRNG.randint(0,n-1)
      x[i] += mutationFactor*myPRNG.uniform(-1,1)
      if (x[i] > 500) or (x[i]<-500):
        x[i] = myPRNG.uniform(-500,500)
```

```
    return x
```

iii. For cross over, we picked on index to swap start the swap. Everything after that index would be traded between the parents. This happened with a *crossOverRate*. Some times just the parents would be returned.

```
def crossover(x1,x2):
  p = myPRNG.random()
  if crossOverRate>p:
    z = myPRNG.randint(1,n-1)
    offspring1 = x1[:z] + x2[z:]
    offspring2 = x2[:z] + x1[z:]

  else:
    offspring1 = x1[:]
    offspring2 = x2[:]

  return offspring1, offspring2  #two offspring are returned
```

iv. Elitism was achieved by keeping the best numbering to *eliteSolutions* from the parent generation. It was important to resort these as the children were often better. We also made sure to change it so the minimum was first as desired here.

```
def insert(pop,kids):

    newlist = []
    for i in range(eliteSolutions):
      newlist.append(pop[i])
    for kid in kids:
      newlist.append(kid)
    popVals = sorted(newlist, key=lambda newlist: newlist[1], reverse = F
    return popVals
```

v. I want to share the code for computing the function. With *evaluate* we made sure to utilize the numpy speed in dealing with large dimensional data.

```
def evaluate(x):
  x = np.array(x)
  return 418.982887272443*n -sum(x*np.sin(np.sqrt(np.abs(x))))
```