# Genetic Algorithms

**Dr. Mahmoud Nabil Mahmoud**
*mnmahmoud@ncat.edu*

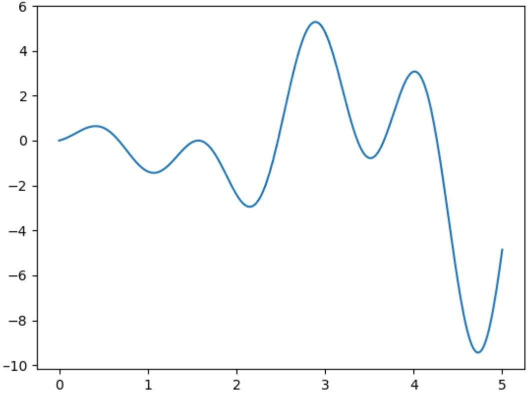
North Carolina A & T State University

September 6, 2021

# A Working Example

Find $x$ that maximize the following function?

$$f(x) = x\sin(5x) + x\cos(2x)$$



# Initialize random pobulation

```
pop = np.random.randint(2, size=(POP_SIZE, CHROMOSOME_SIZE))
```

- A binary population of size `POP_SIZE` and the size of each chromosome is `CHROMOSOME_SIZE`

# Initialize random pobulation

```
pop = np.random.randint(2, size=(POP_SIZE, CHROMOSOME_SIZE))
```

- A binary population of size `POP_SIZE` and the size of each chromosome is `CHROMOSOME_SIZE`
- The `CHROMOSOME_SIZE` will be chosen such that the precision of the solution is up to certain value.

# Initialize random pobulation

```
pop = np.random.randint(2, size=(POP_SIZE, CHROMOSOME_SIZE))
```

- A binary population of size `POP_SIZE` and the size of each chromosome is `CHROMOSOME_SIZE`
- The `CHROMOSOME_SIZE` will be chosen such that the precision of the solution is up to certain value.
- For instance, what is the float point precession of the solution that we need from 0 to 5?

# Decoding an individual

- Assume we need to achieve precision of 1024 steps between 0 and 5

# Decoding an individual

- Assume we need to achieve precision of 1024 steps between 0 and 5
- `CHROMOSOME_SIZE` will have a size 10

# Decoding an individual

- Assume we need to achieve precision of 1024 steps between 0 and 5
- CHROMOSOME_SIZE will have a size 10
- If CHROMOSOME = 0000000111 then a decoded value will be

$$X_{min} + \mathtt{BASE}_{10}(0000000111) * \frac{X_{max} - X_{min}}{2^{\mathtt{CHROMOSOME\_SIZE}} - 1}$$

# Decoding an individual

- Assume we need to achieve precision of 1024 steps between 0 and 5
- `CHROMOSOME_SIZE` will have a size 10
- If `CHROMOSOME` = 0000000111 then a decoded value will be

$$X_{min} + \mathtt{BASE}_{10}(0000000111) * \frac{X_{max} - X_{min}}{2^{\mathtt{CHROMOSOME\_SIZE}} - 1}$$

- To decode the entire population at once:

# Decoding an individual

- Assume we need to achieve precision of 1024 steps between 0 and 5
- `CHROMOSOME_SIZE` will have a size 10
- If `CHROMOSOME` = 0000000111 then a decoded value will be

$$X_{min} + \mathtt{BASE}_{10}(0000000111) * \frac{X_{max} - X_{min}}{2^{\mathtt{CHROMOSOME\_SIZE}} - 1}$$

- To decode the entire population at once:

# Decoding an individual

- Assume we need to achieve precision of 1024 steps between 0 and 5
- CHROMOSOME_SIZE will have a size 10
- If CHROMOSOME = 0000000111 then a decoded value will be
$$X_{min} + \mathtt{BASE}_{10}(0000000111) * \frac{X_{max} - X_{min}}{2^{\mathtt{CHROMOSOME\_SIZE}} - 1}$$
- To decode the entire population at once:

```
def Decode(pop):
    return pop.dot(2 ** np.arange(CHROMOSOME_SIZE)[::-1])* X_MAX / float(2**CHROMOSOME_SIZE-1)
```

# Reproduction

```python
def reproduction(pop, fitness):    # nature selection wrt pop's fitness
    idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, replace=True,
                           p=fitness/fitness.sum())
    return pop[idx]
```

- `np.random.choice` can take a probability `p` of associated with each individual in the population `pop`

# Reproduction

```
def reproduction(pop, fitness):    # nature selection wrt pop's fitness
    idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, replace=True,
                           p=fitness/fitness.sum())
    return pop[idx]
```

- `np.random.choice` can take a probability `p` of associated with each individual in the population `pop`
- A roulette wheel selection in just a single line.

# Crossover

```
def crossover(parent1, parent2):     # mating process (genes crossover)
    if np.random.rand() < CROSS_RATE:
        cross_over_site = np.random.randint(0, CHROMOSOME_SIZE)   # choose crossover site
        child1 =  np.concatenate((parent1[0:cross_over_site],parent2[cross_over_site:]))
        child2 =  np.concatenate((parent2[0:cross_over_site],parent1[cross_over_site:]))
    else:
        child1, child2 = parent1, parent2
    return child1, child2
```

- Mating between two parents `parent1` and `parent1` if the probability is less than the `CROSS_RATE`

# Crossover

```
def crossover(parent1, parent2):     # mating process (genes crossover)
    if np.random.rand() < CROSS_RATE:
        cross_over_site = np.random.randint(0, CHROMOSOME_SIZE)   # choose crossover site
        child1 =  np.concatenate((parent1[0:cross_over_site],parent2[cross_over_site:]))
        child2 =  np.concatenate((parent2[0:cross_over_site],parent1[cross_over_site:]))
    else:
        child1, child2 = parent1, parent2
    return child1, child2
```

- Mating between two parents `parent1` and `parent1` if the probability is less than the `CROSS_RATE`
- Single point crossover.

# Crossover

```
def crossover(parent1, parent2):     # mating process (genes crossover)
    if np.random.rand() < CROSS_RATE:
        cross_over_site = np.random.randint(0, CHROMOSOME_SIZE)   # choose crossover site
        child1 =  np.concatenate((parent1[0:cross_over_site],parent2[cross_over_site:]))
        child2 =  np.concatenate((parent2[0:cross_over_site],parent1[cross_over_site:]))
    else:
        child1, child2 = parent1, parent2
    return child1, child2
```

- Mating between two parents `parent1` and `parent1` if the probability is less than the `CROSS_RATE`
- Single point crossover.
- Two offsprings are generated.

# Mutation

```
def mutate(child):
    for point in range(CHROMOSOME_SIZE):
        if np.random.rand() < MUTATION_RATE:
            child[point] = 1 - child[point]
    return child
```

- Each bit can be mutated (i.e., flipped) with probability `MUTATION_RATE`
- Single point crossover.
- Two offsprings are generated.

# Main Loop

- Below is the main operations done by a simple GA
- Mating the first have of the population with the second half.

```
pop = reproduction(pop, fitness)
pop = np.random.permutation(pop)
i = 0
while i< len(pop)//2   :
    parent1 = pop[i]
    parent2 = pop[-(i+1)]
    child1, child2 = crossover(parent1, parent2)
    child1 = mutate(child1)
    child2 = mutate(child2)
    pop[i] = child1
    pop[-i] = child2
    i+=1
```

# References


- Goldenberg, D.E., 1989. Genetic algorithms in search, optimization and machine learning.
- Michalewicz, Z., 2013. Genetic algorithms + data structures= evolution programs. Springer Science & Business Media


Questions