Marko Smiljanic

CSc – 180

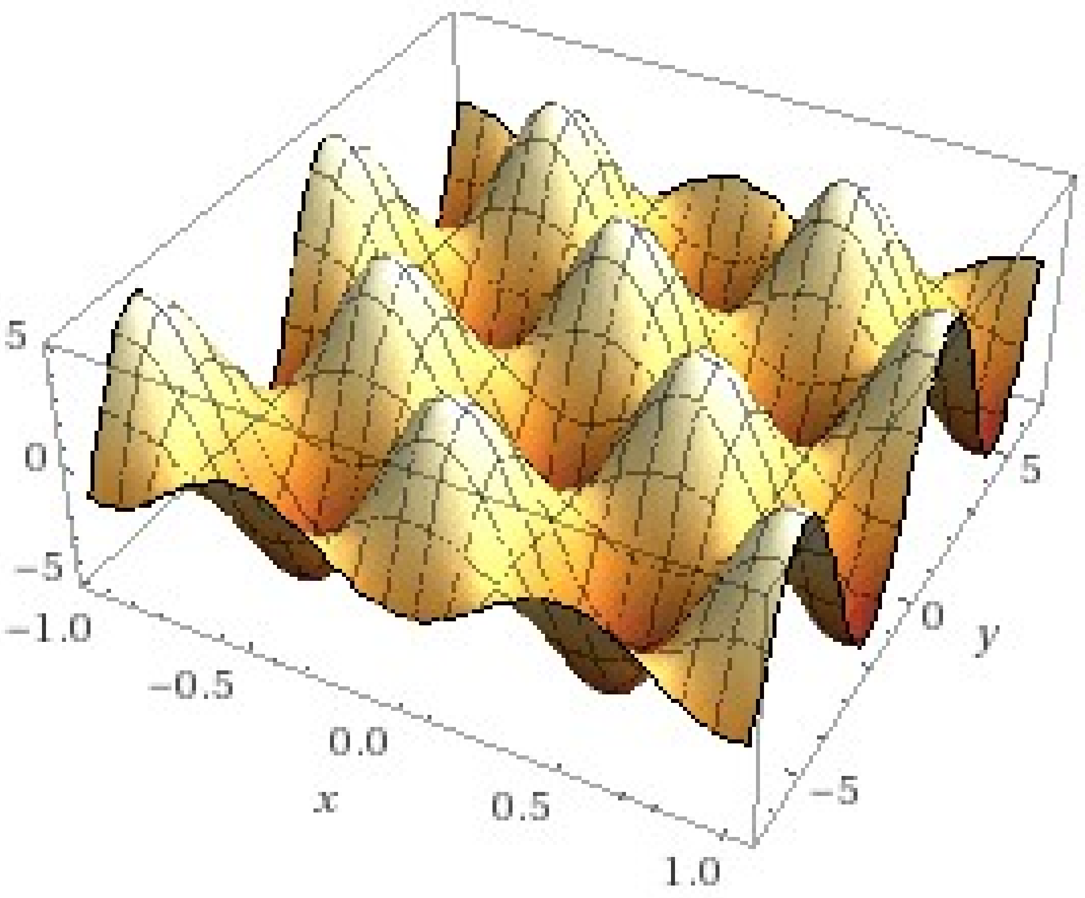
Spring 2019

Project 4 – Genetic Algorithms and Genetic Programming

1. **Part 1: Genetic Algorithms**
    a. **Function f(x,y)**

The function I have used is listed below and is followed by its 3D representation. This function is a has many local minima and maxima, the minimum being 5 and the maximum being -5. The equation to determine potential solutions for the minima is listed below.

5(cos(2 * pi * x) * sin(y))



Graph were automatically generated by inputting the equation into wolfram alpha.

Local Minima can be described using the following equations.

$$\min\{5\cos(2\pi x)\sin(y)\} = -5 \text{ at } (x, y) = \left(2n_2 - \frac{1}{2}, 2\pi n_1 + \frac{\pi}{2}\right) \text{ for integer } n_1 \text{ and } n_2$$

$$\min\{5\cos(2\pi x)\sin(y)\} = -5 \text{ at } (x, y) = \left(2n_2 + \frac{1}{2}, 2\pi n_1 + \frac{\pi}{2}\right) \text{ for integer } n_1 \text{ and } n_2$$

Local Minima equations were automatically generated by inputting the equation into wolfram alpha.

b. Genetic Algorithm

Using the function from Part A, I was able to modify the code we were provided by the professor to now included two inputs. This allowed me to use the code to solve for a random local minimum in the range of negative five to five for x and y values. Many portions of the code had to be changed to include and x and a y value including the initial struct, the evaluate function, and others. The decode portion also had to be separated into two portions, one that would determine the integer from the beginning 20 bits, and one that could determine the integer from the end 20 bits. The total number of calls to the evaluate function turned out to be 1519 and after 30 trials, all results ended with a set of X and Y coordinates that equaled to the local minimum of negative five.

2. Part 2: Genetic Programming

a. fungp Input

For this portion of the assignment, I had created a collection of 25 in points consisting of an X and Y value and their respective output, however I found that having this many points made it take far too long for the program to run, and the resulting equations that were generated were far to complicated to be made into anything sensical. The following data set is the original 25 points I used, and the resulting generated equation and its fitness:

```
(test-regression3 100 100)

(def in-list1 '(0 1 1    0.5  -1    -1  1    -2  -1    -3   2    3   1   1.25 1.5  1.75 0    -0.25 -0.25 -0.5 -0.5
0.5  2    2    -2))

(def in-list2 '(0 0 1    0.25 -1     0 -1    -1.5 1    -1.5 1.57 0.5 0.5 0.5  0.5  0.5  1.57 1     -1    -0.5 0.5
0.5  3.14 7.85 7.85))

(def out-list '(0 0 4.21 -1.23 -4.21  0 -4.21 -5   4.21 -5   5    2.4 2.4 0    -2.4 0    5    0     -4.08 -2.4 -
2.4 -2.4 0    5    5))
```

```
(let
 []
 (+
  (+
   (+
    (Math/cos (* 6.0 (dec (- x 2.0))))
    (+
     (+
      (Math/sin (+ (* 6.0 (dec (- x 2.0))) (dec (Math/sin y))))
      (Math/sin (+ (* 6.0 (dec (- x 2.0))) (dec (Math/sin y)))))
     (Math/sin
      (+
       (Math/cos (* 6.0 (dec (- x 2.0))))
       (dec (Math/sin (Math/sin y)))))))
   (+ (Math/sin y) (Math/sin y)))
  (Math/sin
   (+
    (Math/sin
     (Math/sin
      (+
       (Math/cos (* 6.0 (dec (- x 2.0))))
       (+ (Math/sin y) (Math/sin y)))))
    (dec (Math/cos (* 6.0 (dec (- x 2.0)))))))))
Error:  8.793699216795547
```

I ended up using the following inputs and outputs to generate a usable equation:

```
(def in-list1 '(0  1  1      0.5   -1     -1   1     -2   -1     -3))
(def in-list2 '(0  0  1      0.25  -1      0  -1     -1.5  1     -1.5))
```

```
(def out-list '(0  0  4.21  -1.23  -4.21   0  -4.21  -5    4.21  -5))
```

Using this input/output set up and the running call, `(test-regression3 30 30)`, gave the following result:

```
(let
  []
    (-
      (- (- (- (* 6.0 y) x) (Math/sin (Math/sin y))) (Math/sin y))
      (Math/sin (- (* 6.0 y) x))))
Error:  2.0581416217470396
```

This equation can be simplified to:

```
(((((6.0 * y) - x) - (sin(siny))) - (siny)) - (sin((6.0 * y) - x))))
```

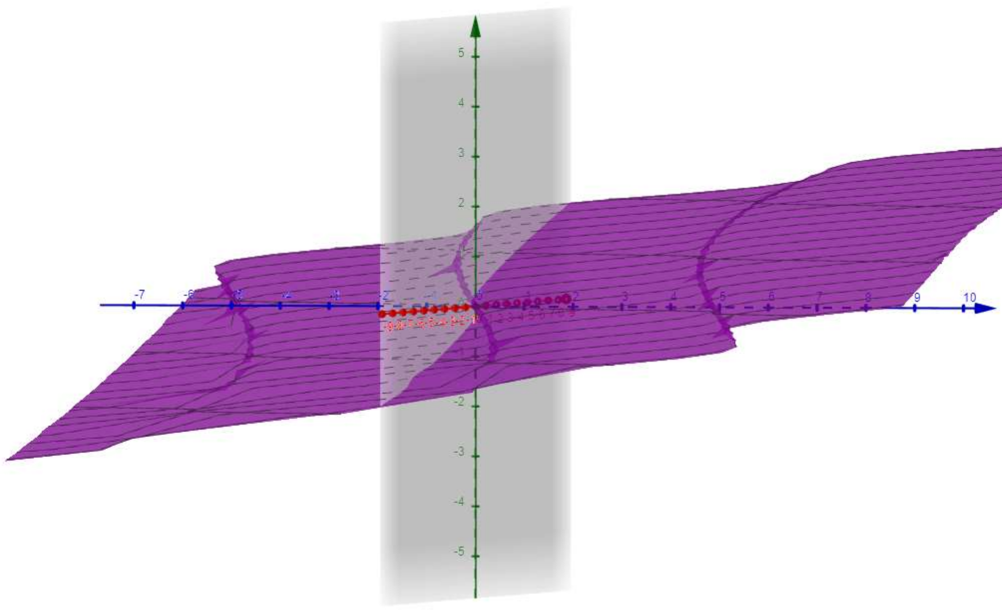
or

```
sin(x - 6y) - x + 6y - sin(y) - sin(sin(y))
```

I kept the program fairly similar to the example program that was provided by the professor. I removed the ‘fungp.util/abs 1’ and ‘fungp.util/sdiv 2’ functions, changed the inputs and outputs, and experimented with using different max depths.

b. fungp 3D Graph

Shown below is the 3D graph of the equation given by the fungp equation.



As you can see, this 3D graph is fairly different from the graph we began with. This was the most reasonable equation I was able to find that could be interpreted and entered into a 3D calculator. I believe I made this portion of the project harder for myself in the equation I chose to use in part A. Had I used a simpler equation; I may have been able to solve for a graph that was closer to the original.