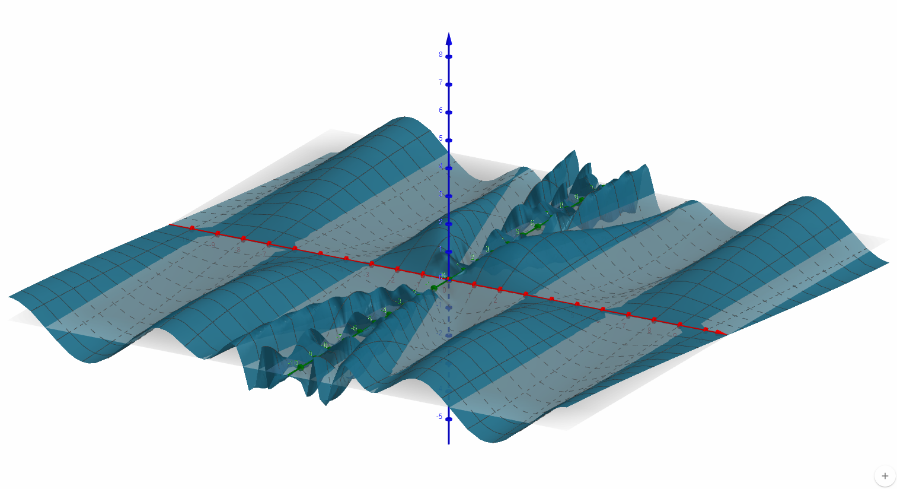
**Project 4**

**Ajinkya Malhotra, Tanveer Bariana**

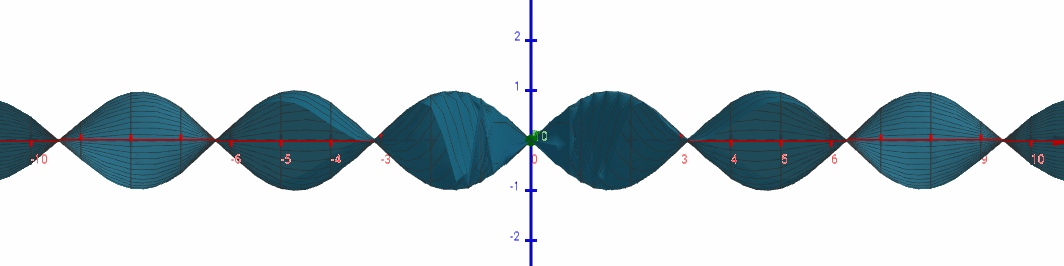
**CSC-180 Fall 2017**

**Genetic Algorithm and Genetic Programming**

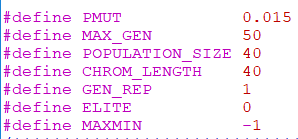
PART 1: Genetic Algorithms

* 1. **Function 🡪** f(x, y) = sin(x) \* sin(y/x).
  2. **Local Minimum 🡪** ~-1.0000
  3. **Selected Range 🡪** 0 to 20
  4. **3D graph:**

**3D view 🡪**

**2D view 🡪**

1. Our goal in part B be was to modify the code and try to reach local minima of our function. The modified Genetic code was able to fetch the minima of our f(x, y), it was hovering around -0.9970 to -1.0000 but was never more than -1.0000 or less than -0.9970. This conclusion was based of 30+ runs tried with various settings. The best settings we found while trying different values were



The optimum solution that the genetic algorithm was able to find was through the setting above.

These are the 28 recorded runs with their settings and their outputs.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **NO.** | **MUT. RATE** | **MAX GEN.** | **POP. SIZE** | **DECODED VALUEX** | **DECODED VALUEY** | **FITNESS** | **FITNESS**  **Count** |
| Default | 0.03 | 100 | 30 | 10.9905 | 17.2645 | -1.0000 | 3426 |
| 1 | 0.015 | 50 | 20 | 4.7009 | 7.5206 | -0.9995 | 1193 |
| 2 | 0.015 | 50 | 40 | 10.9960 | 17.2634 | -1.0000 | 2243 |
| 3 | 0.015 | 50 | 80 | 10.9988 | 17.5012 | -0.9998 | 4277 |
| 4 | 0.015 | 120 | 20 | 10.9996 | 17.2155 | -1.0000 | 2835 |
| 5 | 0.015 | 120 | 40 | 10.9375 | 17.1621 | -0.9983 | 5324 |
| 6 | 0.015 | 120 | 80 | 10.9956 | 17.2726 | -1.0000 | 10204 |
| 7 | 0.015 | 180 | 20 | 4.6872 | 7.3637 | -0.9997 | 4271 |
| 8 | 0.015 | 180 | 40 | 10.9956 | 17.2731 | -1.0000 | 7982 |
| 9 | 0.015 | 180 | 80 | 10.9955 | 17.2654 | -1.0000 | 15325 |
| 10 | 0.05 | 50 | 20 | 4.6985 | 7.1550 | -0.9988 | 1186 |
| 11 | 0.05 | 50 | 40 | 10.9909 | 17.1867 | -1.0000 | 2205 |
| 12 | 0.05 | 50 | 80 | 4.7191 | 7.3843 | -1.0000 | 4239 |
| 13 | 0.05 | 120 | 20 | 11.0168 | 17.5711 | -0.9995 | 2843 |
| 14 | 0.05 | 120 | 40 | 10.9997 | 17.2718 | -1.0000 | 5311 |
| 15 | 0.05 | 120 | 80 | 10.9916 | 17.2530 | -1.0000 | 10180 |
| 16 | 0.05 | 180 | 20 | 11.0147 | 17.5172 | -0.9996 | 4266 |
| 17 | 0.05 | 180 | 40 | 10.9877 | 17.2212 | -1.0000 | 8018 |
| 18 | 0.05 | 180 | 80 | 10.9978 | 17.2236 | -1.0000 | 15324 |
| 19 | 0.5 | 50 | 20 | 11.0288 | 17.0714 | -0.9992 | 1176 |
| 20 | 0.5 | 50 | 40 | 4.6361 | 7.2936 | -0.9971 | 2208 |
| 21 | 0.5 | 50 | 80 | 1.5631 | 17.1707 | -0.9999 | 4252 |
| 22 | 0.5 | 120 | 20 | 10.9945 | 17.8478 | -0.9986 | 2819 |
| 23 | 0.5 | 120 | 40 | 10.9974 | 16.8992 | -0.9994 | 5315 |
| 24 | 0.5 | 120 | 80 | 11.0021 | 17.2665 | -1.0000 | 10206 |
| 25 | 0.5 | 180 | 20 | 1.6314 | 17.9078 | -0.9980 | 4242 |
| 26 | 0.5 | 180 | 40 | 1.5914 | 7.5120 | -0.9998 | 7967 |
| 27 | 0.5 | 180 | 80 | 10.9970 | 17.1416 | -0.9999 | 15294 |

As we can observe below there were many different runs that were able to reach similar fitness value of -1.0000 but when compared to the number of times the fitness function was called run 2 gives the optimized solution.

Other changes for part B, except the top setting, were done in convRange(), evaluate() and decode().

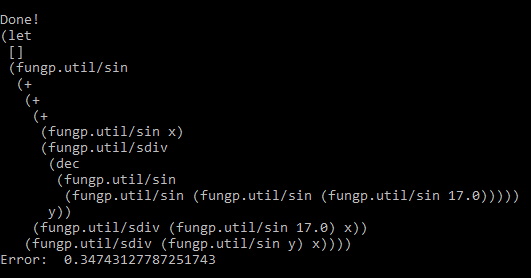
* decode() function was expanded into two different functions decodeX() and decodeY().
* conRange() function was modified to adapt to our 0 to 20 range and to work with 20 bits per input instead of 16 bits
* evaluate() function’s equation was change to our f(x, y) equation.

Except these three many more small changes were made in the code to adapt to 40 bits and to f(x, y).

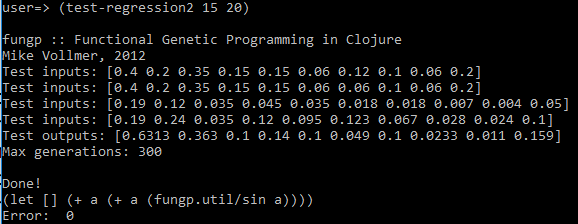
PART2: Genetic Programming

1. **Fungp**
   1. The result below was the best result generated by using 25 sample inputs. We tried to use 50 inputs but because of the points being concentrated for most cases and being scattered for some cases, the fungp was not able to generalize entirely. With 50 sample inputs we received an error of 1.5724 and this best result with 25 sample inputs still produces an error of 0.3474.

**Using (test-regression2 15 25)**

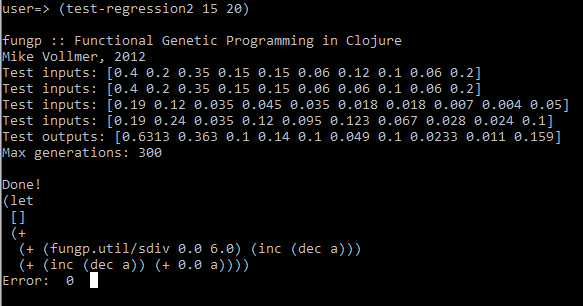


* 1. **Beam Problem:**
     1. **BeamA (Training Set):** For training Set input provided, fungp was able to learn t to mimic the function. Receiving an error of 0, the result we got was



While running the BeamA problem, the program generated a unique function every time compared to the previous answer, even with the same settings.

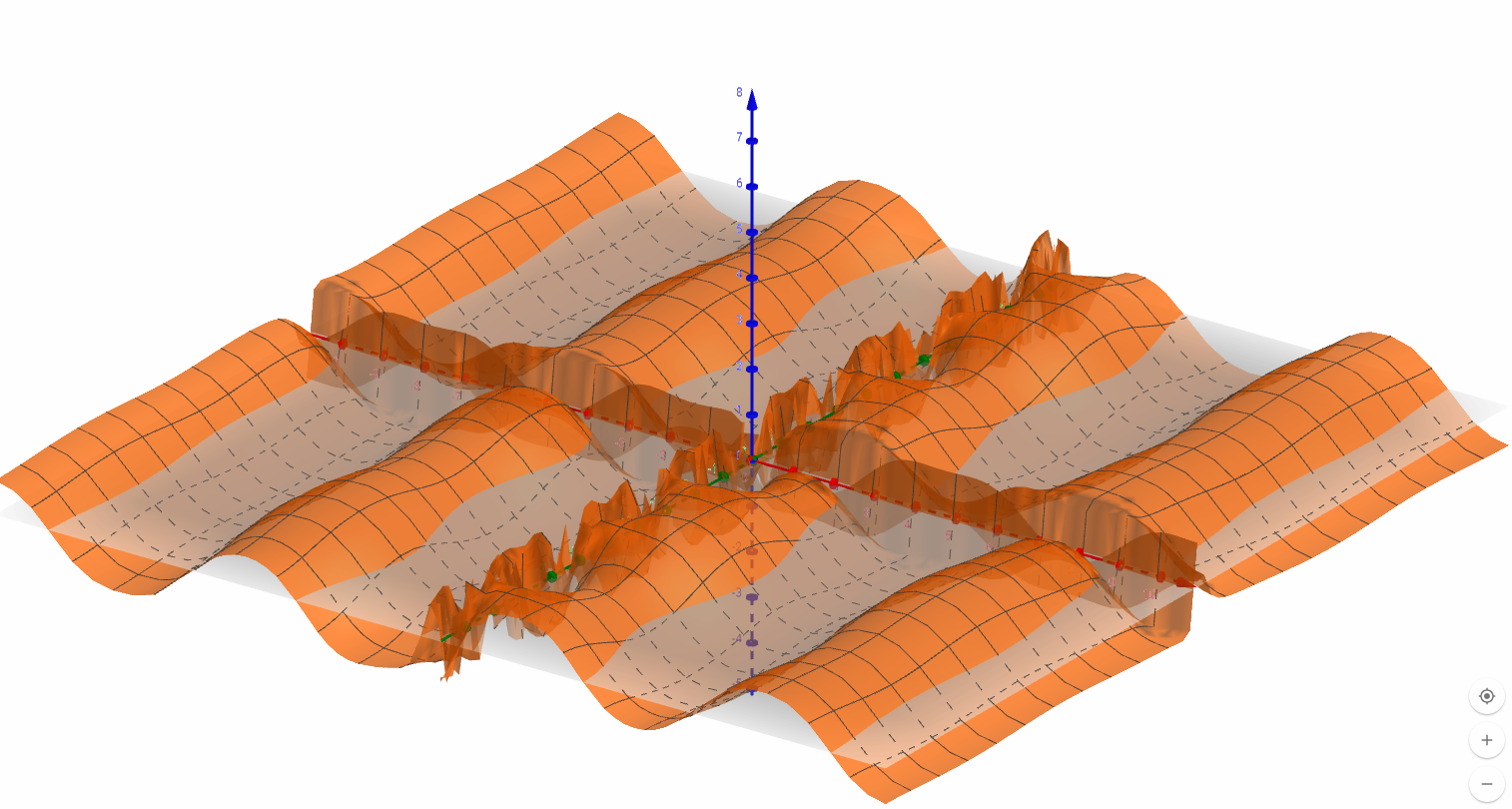
* + 1. **BeamB (Test Set):** For training Set input provided, fungp was able to learn t to mimic the function. Receiving an error of 0, the result we got was



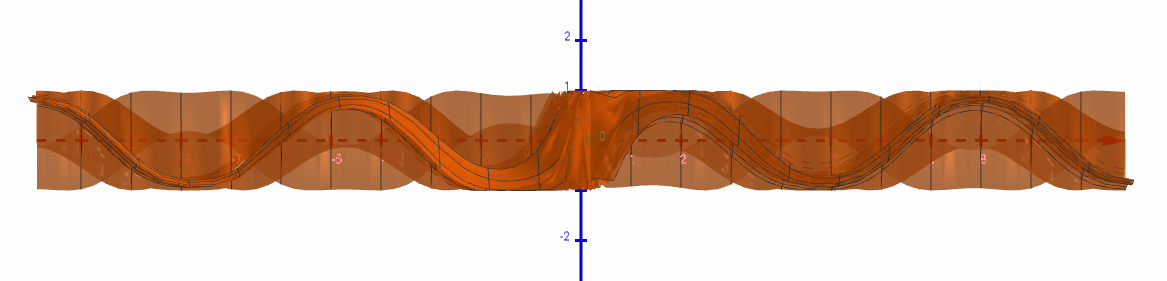
We ran into the same problem as BeamA problem where in some cases we had d different outputs for the same data with same settings.

1. **Fungp Function**
   1. **Function 🡪** sin (((((sin(sin(sin(sin (17)))))/y)+sin(x))+(sin(17)/x))+(sin(y)/x)
   2. **Local Minimum 🡪** -1.0000
   3. **Selected Range 🡪** 0 to 20
   4. **Error 🡪 0.3474**
   5. **3D graph:**

**3D view 🡪**



**2D view 🡪**



Comparison:

1. Minima: As we can observe in the 2d view the fungp function’s local minima is -1.0000 but for the actual function it was always ~-1.0000 but not exactly -1.
2. Graph: Purple represents actual function and orange represents the fungp produced function.

As we can observe below some of the portions of the graph were mimicked by the fungp perfectly but was not able to mimic the entire function correctly. We think the reason for this could be that the fungp required more amount of sample points from all over the graph and not just concentrated on a single portion this could be one of the reason our graph isn’t 100 % accurate.

But as we can observe below our fungp function (sin (((((sin(sin(sin(sin (17)))))/y)+sin(x))+(sin(17)/x)) +(sin(y)/x)) came pretty close to our actual function (sin(x)\*sin(y/x)) for certain portions of the graph.

