Benchmark Optimization Functions Using Genetic Algorithms

Description of the selected functions and their plots

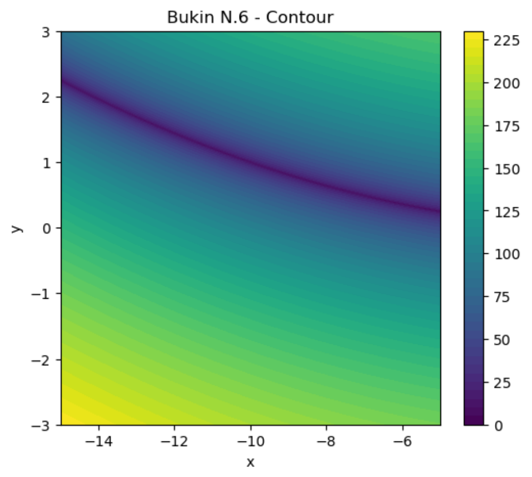
For this assignment I chose the Bukin Function N.6 and the Drop-Wave Function. Here are their definitions and the plots I made using pyplot.

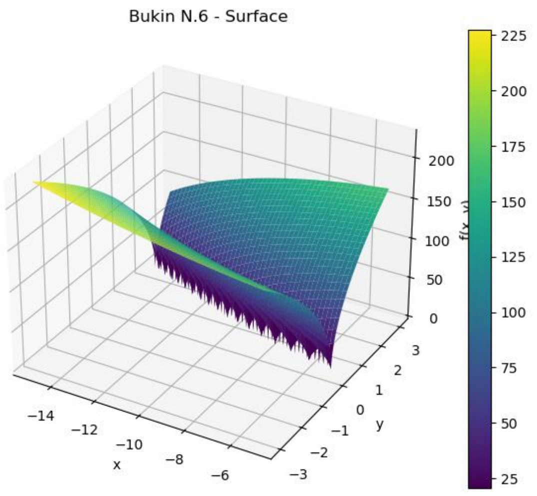
- Bukin Function N.6:

$f : [-15, 5] \times [-3, 3] \rightarrow R$

$f(x) = 100\sqrt{|x_2-0.01x_1^2|}+0.01|x_1+10|$

global minimum: $f(x^*)=0$, at $x^*=(-10,1)$



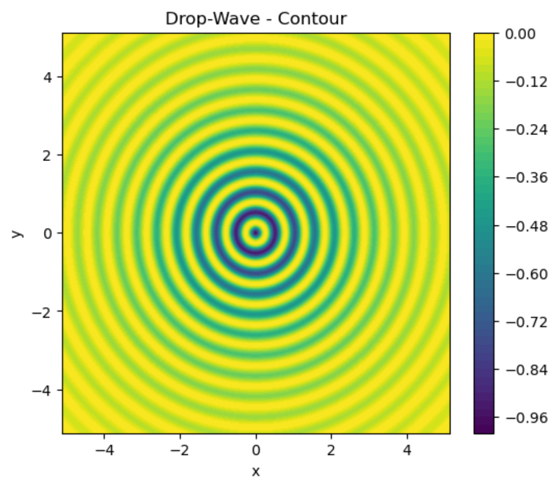


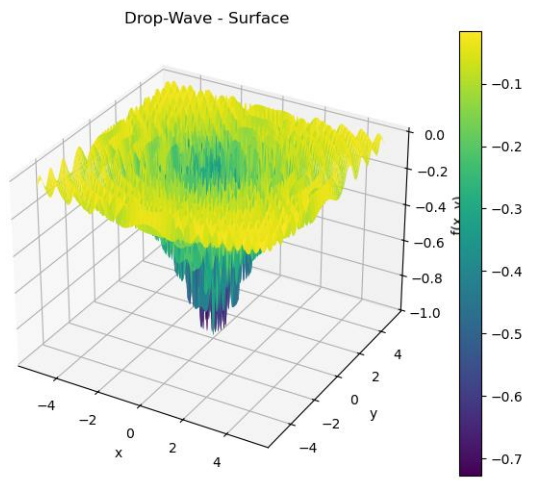
- Drop-Wave Function:

f : [-5.12, 5.12] x [-5.12, 5.12] -> R

$f(x)=-(1+\cos(12\sqrt{x_1^2+x_2^2}))/(0.5(x_1^2+x_2^2)+2)$

global minimum: $f(x^*)=-1$, at $x^*=(0,0)$





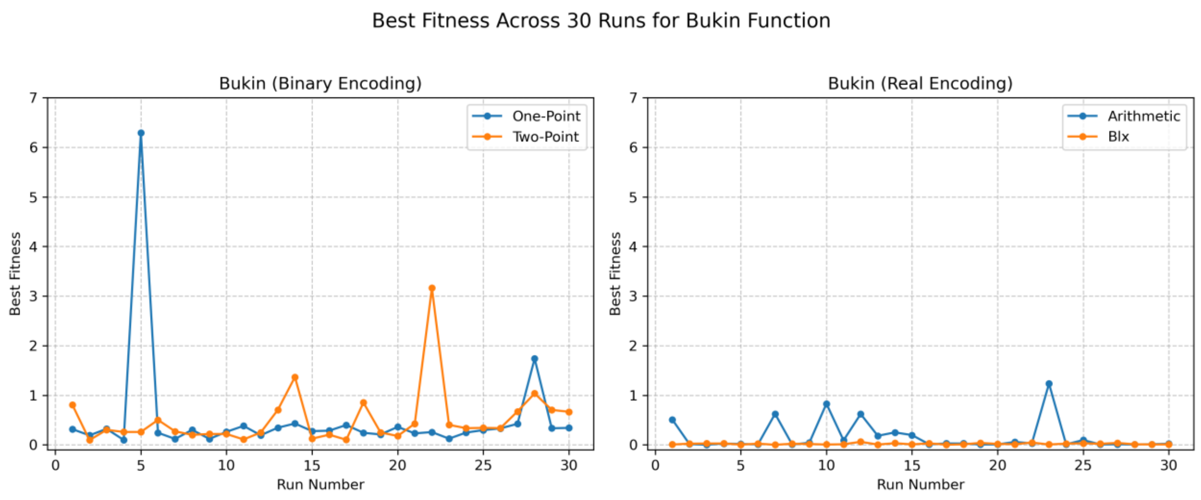
Explanation of GA configurations

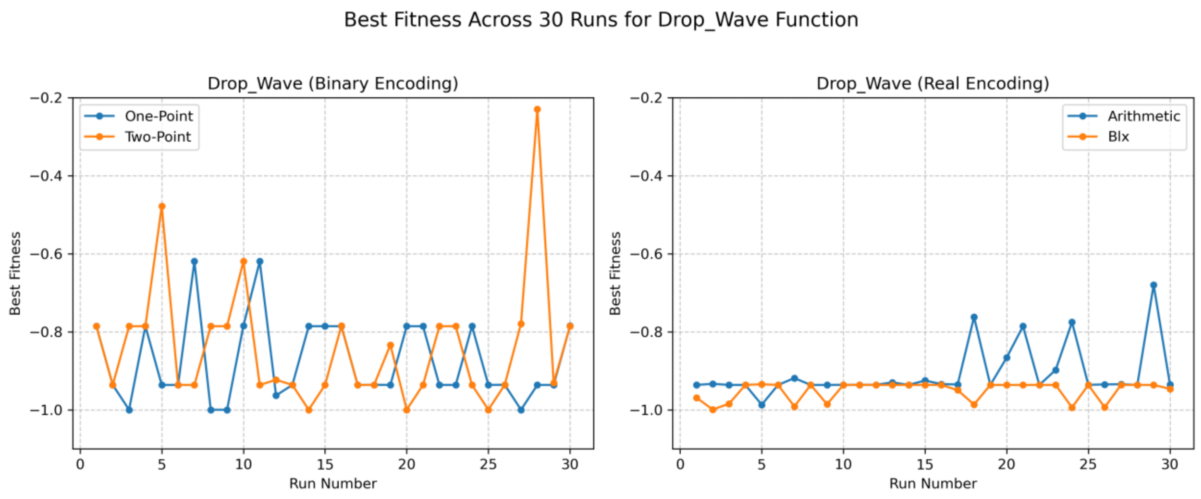
The populations were randomly generated within the bounds of each function. I found that a population size of 30 got closer to the intended result than higher sizes, even at higher generation counts.

For the crossover, a moderately-high rate (0.7) worked best for me. The selection of the individuals was done using a tournament selection. Each parent was chosen based on its fitness from a pool of 3 randomly chosen individuals. This pool sized yielded the most consistent results.

The children replaced the parents after a crossover, thus maintaining a constant population size. At first glance this seemed to be unproductive, since it eliminated the best fit individuals from the population, but, on the flip side, it allowed all individuals to partake in the crossover process, regardless of their fitness, despite the convergence being really quick.

Tables and plots of experimental results





We can observe that the variance in the Drop-Wave using Binary Encoding is very high, while in the other configurations it is relatively low. It can also be seen that the Real Encoding worked much more better than the Binary Encoding in the case of the Bukin function, but not in the Drop Wave, even though it was more stable.

Function	Encoding	Crossover	Generations	Mean	Std	Min
bukin	binary	one_point	30	0.524818	1.125297	0.098691
bukin	binary	two_point	30	0.512981	0.585996	0.096330
bukin	real	arithmetic	30	0.167148	0.298127	0.003607
bukin	real	blx	30	0.020240	0.014318	0.001940
drop_wave	binary	one_point	30	-0.874252	0.106561	-0.999831
drop_wave	binary	two_point	30	-0.838762	0.163611	-0.999831
drop_wave	real	arithmetic	30	-0.908016	0.067017	-0.986934
drop_wave	real	blx	30	-0.950746	0.023453	-0.999250

Statistical comparison and conclusions

I ran t-tests using the scipy library and the data from the total of 240 runs and I compared, two by two, the crossover methods for each representation and function, and the representations for each function.

Results:

T-test for Bukin (Representation): Binary vs. Real

T-statistic: 3.593
P-value: 0.0006
Conclusion: The difference is significant ($p < 0.05$).

T-test for Drop_Wave (Representation): Binary vs. Real
T-statistic: 3.806
P-value: 0.0003
Conclusion: The difference is significant ($p < 0.05$).

T-test for Bukin (Crossover (Binary)): One-Point vs. Two-Point
T-statistic: 0.051
P-value: 0.9595
Conclusion: The difference is not significant ($p > 0.05$).

T-test for Bukin (Crossover (Real)): Arithmetic vs. BLX
T-statistic: 2.696
P-value: 0.0115
Conclusion: The difference is significant ($p < 0.05$).

T-test for Drop_Wave (Crossover (Binary)): One-Point vs. Two-Point
T-statistic: -0.996
P-value: 0.3243
Conclusion: The difference is not significant ($p > 0.05$).

T-test for Drop_Wave (Crossover (Real)): Arithmetic vs. BLX
T-statistic: 3.296
P-value: 0.0022
Conclusion: The difference is significant ($p < 0.05$).

These tests suggest that the BLX crossover seems to be working better for each of the functions and representations. The binary representation got closer to the result for the Drop-Wave function, while the real representation worked better for the Bukin function.