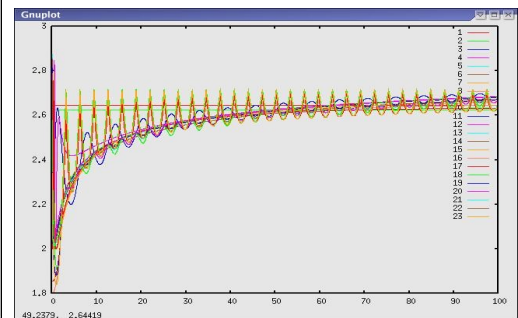
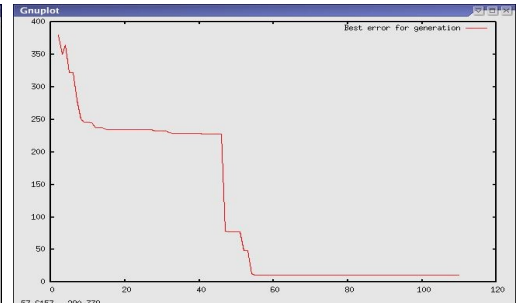
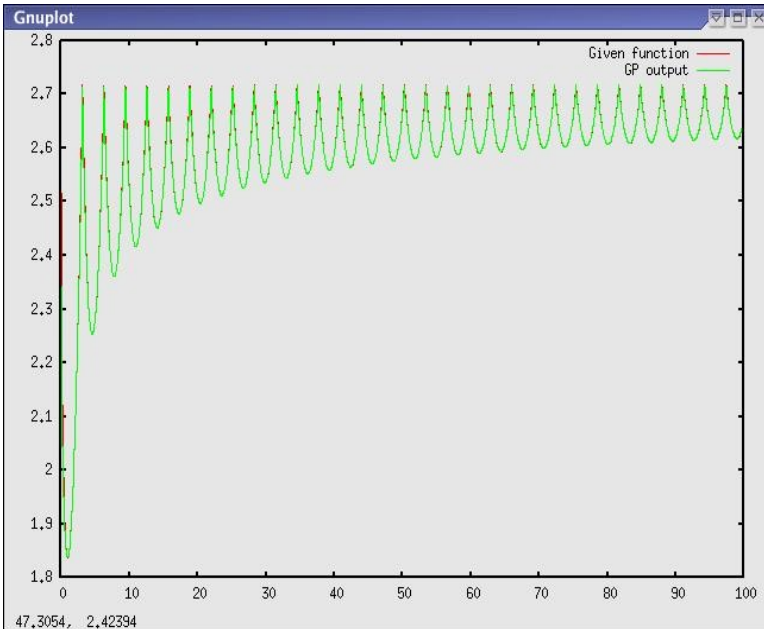


I. The Best Result Computed by my GP

$$f(x) = e - \sqrt{\frac{\sin x}{0.999223} \cdot \sqrt{\frac{\sin x}{0.996777}}}$$



II. Description of Used Techniques

For the given function, the following functions has been selected as nonterminals when creating the random individuals:

$$\sin x, |x|, -x, \sqrt{x}, \log x, x + y, x - y, x * y, x / y, x^y$$

As the terminals, the symbols for x, e and random numbers from 0 to 10 were chosen. The exact probabilities of creation of any terminal or nonterminal could be specified and can be seen in the gp_tools.h file. The algorithm is using the following operators:

Crossover in which two tree nodes are selected randomly, one from parent 1 and the other one from parent 2 and the subnodes are switched. If the depth of the tree seems to exceed the given constant, the trees are terminated by randomly selected terminals.

X Mutation is used to prevent the constants to rule over the population. Because the constants are fairly close to the given function and especially the constant e which is given as a terminal itself, they tend to be the best solutions. Because of their small depth, they can also produce only constants on crossover so they will easily dominate. The x mutation is applied to each individual that is a constant and it simply selects randomly any constant in the tree and replaces it by the x. It is also applied to other subtrees in a small amount.

X Simplified Mutation and *Number Simplified Mutation* reduces the diversity of population by deleting selected subtrees /while their probability varies among the depth, so it is very likely for them to occur in the last row, but they almost never occur in the first one/.

Function mutation only selects any function in the tree and replaces it by randomly selected other function of the same arity.

Constant scaling is used for good individuals. This operator selects randomly the number from the tree and then produces various amounts of identical offsprings that differs slightly only in this constant. This operator is applied with probability of 10% to all individuals and with the 100% probability to the fittest individual.

Simplifying operator is not applied to the population, but is applied with the probability of 20% to the individuals with error less than 20. This operator is also applied to the best found individual to

make it more readable. It recognizes the following things and replaces them with proper adequates:

- subtrees without x value are replaced by their value as a constant
- $-(-x)$ is replaced by x
- $||x||$ is replaced by $|x|$
- $x+0$ and $0+x$ are replaced by x as well as $x-0$
- $x/1$ is replaced by x and $0/x$ is replaced by 0
- subtree divided by the same subtree is replaced by 1

In addition, the size of population is not specified exactly, but as a lower bound. As the probability of simplifying and constant scaling operators increases with the individual fitness the size of population is increased.

III. Performance Analysis

The operations on trees and with float numbers are very time consuming ones and so a lot of effort has been done to speed up the whole algorithm. The function trees are represented as arrays and so all the operations are very efficient and are done in constant time. This representation also allows to easily select randomly any node of the tree. Second performance improvement is the evaluation of the given individual. Because this requires at least 5000 floating point operations, it is the bottle neck of the whole algorithm. Thus the progressive evaluation is used. At first a randomly selected 50 values are evaluated. If the error is small, then another 250 values are checked. And if the error is still small then the whole data are checked. Compared to the run with same conditions but full evaluation of all functions this delivers 10x speedup /because crossover produces easily very bad individuals/.

IV. Fitness evaluation and Roulette Wheel Implementation and Population Size

The sum of all errors is replaced by the fitness which is defined as:

$$f(x) = 100 \frac{thr^2}{err}$$

where the thr is the best individual's error found in previous generation. This scaled fitness mapping also improves the chances of better individuals. The size of population is roughly 500 individuals, but starting on 3000 for better diversity and decreasing to 500 in the first 5 generations.

V. Summary of generated outputs and program usage

For every best individual found, a separate file best_#.txt is created in which is the full description of the individual. Also the file stats.txt is created that contains the statistics of basic variables per generations. The application is highly customizable by the macros all to be found in the gp_tools.h file. Their names and values are self explanatory. Application is now in the state of producing the greedy crossover algorithm. Please note that the gp_data.txt file must be in the same directory as the executable. Those files are in format ready to be printed by gnuplot.

Application is the KDevelop project, but could be run on any computer with the GNU autotools by using this commands in the application root folder:

```
./configure
make all
```

VI. Conclusion

The process of creating the regression function was highly dependent on the probabilities of the generation of the terminal and nonterminal nodes as well as the rates for other operators. With the equal chances for all functions, the algorithm was not so successful, but after looking at the given function and updating the probabilities of different functions, the monotonic approximation has been usually found before the 5th generation and tuned to optimal performance before 7th generation. However to model the whole shape of the function was more difficult and even now, only 50% of runs achieves this goal before 50th generation, of which about 30% delivers very good results before then generation 10. The graphs in the first section represents the most common behavior of the application, where the very good solution is obtained relatively fast, then the constants are scaled immediately and then there is a long time gap before another improvement.