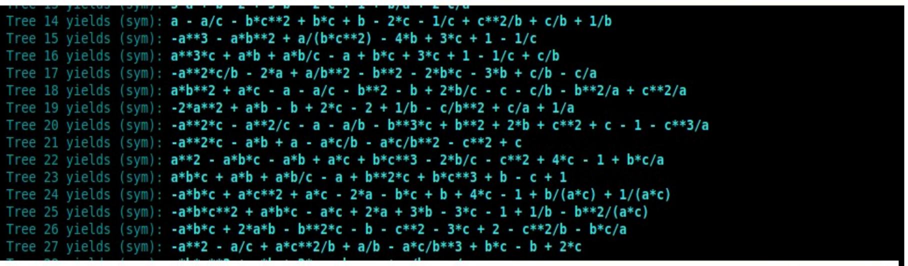


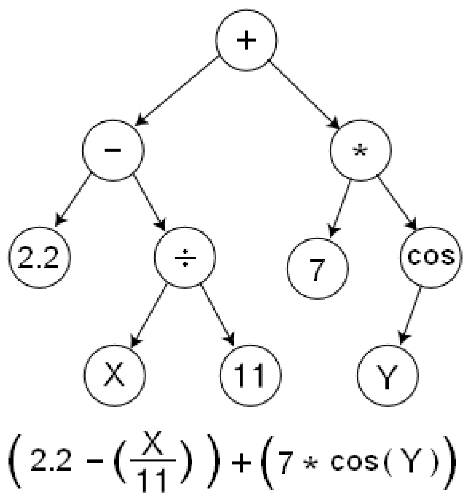
```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

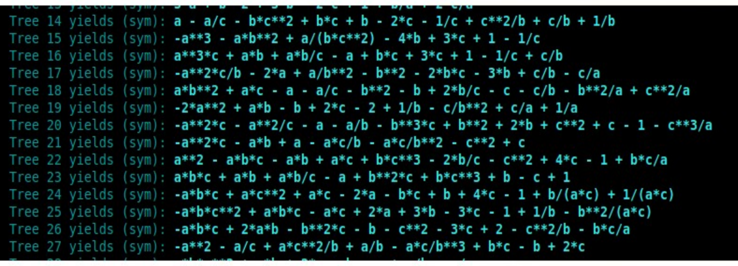
Karoo GP Quick Start Guide

A Genetic Program for Python

ver. 20151026

by Kai Staats





```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

Karoo GP Quick Start Guide

Welcome to Karoo GP!

Karoo GP is an evolutionary algorithm, a *genetic programming* application suite which provides both symbolic regression and classification analysis. Written in the programming language Python, Karoo GP owes its foundation to the "Field Guide to Genetic Programming" by Poli, Langdon, McPhee, and Koza.

Karoo GP provides a transparent interface to the inner workings of genetic programming. As a teaching tool, it enables instructors to showcase, step-by-step, how an evolutionary algorithm arrives to its solution. As a hands-on learning tool, Karoo GP supports rapid, repeatable experimentation with a simple, no-programming-required interface. Included with Karoo GP are two executables: an intuitive Text-based User Interface with built-in, real-world test cases, and a fully scriptable, single-line configuration which provides SciKit Learn-like functionality.

Requirements:

- Python 2.7.6 [GCC 4.8.2]
- libraries: NumPy 1.9.2, SymPy 0.7.6.1, pprocess 0.5.1, sklearn 0.16.1, matplotlib, csv, os, sys, time

Tested systems:

- Ubuntu Linux 14.04 Desktop on Intel i5/i7 (4 core): no known issues
- Ubuntu Linux 14.04.3 LTS on Intel Xeon E5-2650 (40 core): no known issues

Launch Karoo GP:

(from a native shell): $ python karoo_gp.main.py

(from iPython): $ run karoo_gp_main.py

This Quick Start Guide is designed to give you a crash course in this particular flavour of genetic programming and a light introduction to GP in general. Even if you are an expert in evolutionary algorithms, you are encouraged to make time to read this document in order to engage all Karoo GP has to offer.

What is Genetic Programming?

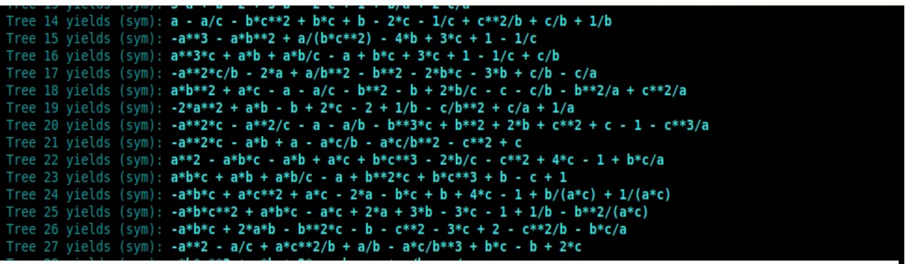
Genetic Programming (GP) is a type of evolutionary algorithm, a subset of machine learning. Inspired by biological evolution, GP is a computer program that uses random mutation, a fitness function, and multiple generations of evolution to resolve a user-defined task. At the foundation level, genetic programming discovers the relationship between features in data, that is, correlations between measurements of the real world.

For example, all of the planets in our solar system hold a nearly identical relationship to each other given their orbital periods and average distance from the Sun. When Johannes Kepler (1571-1630) derived this mathematical relationship based upon the observational data available in his time, he discovered that the orbits of the Earth and Mars could be described as:

[orbital period] ^ 2 / [average distance from the sun] ^ 3

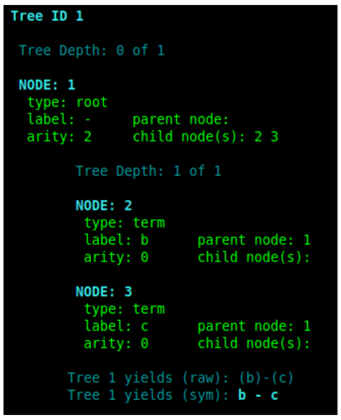
A Genetic Program may be used to rediscover the same relationship for *all* the planets in our solar system, expressed as the mathematical function. In fact, this classic test case is included with Karoo GP as the default Absolute Value fitness function, described in more detail below.

In general, genetic programming can be used to discover a relationship between features in data (symbolic regression) or to group data into categories (classification). Both of these employ a training (learning) and test (validation) phase, after which the resulting function (mathematical expression) may be applied to new data in order to showcase the same relationships between elements in the data, or to sort data into the known classes.



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

First Time Through



```
Tree ID 1

 Tree Depth: 0 of 1

 NODE: 1
  type: root
  label: -      parent node:
  arity: 2      child node(s): 2 3

        Tree Depth: 1 of 1

        NODE: 2
         type: term
         label: b      parent node: 1
         arity: 0      child node(s):

        NODE: 3
         type: term
         label: c      parent node: 1
         arity: 0      child node(s):

       Tree 1 yields (raw): (b)-(c)
       Tree 1 yields (sym): b - c
```

The Karoo GP user interface will prompt you for a series of inputs. Your first time through, enter Play mode, as follows, in order to understand the functions of Karoo G and to learn to visualise a GP tree.

1. Select Play mode.
2. Select a Full tree.
3. Select a depth of 1.

   Depth 1: 1 operand, 2 variables, eg: a+b

   Depth 2: 3 operands, 4 variables, eg: (a+b) * (c-d)

   Depth 3: 7 operands, 8 variables, eg: (a+b) * (c-d) / (a**e) * (b-c)

Take a look the GP tree represented on-screen. There are 3 nodes composed of 2 terminals (variables) and 1 function (operand). Select Play mode a few more times, playing with larger trees and both Full and Grow methods. You will find that the branches of Full trees always reach the bottom, where the branches of Grow trees may terminate early, with a terminal as the final node. With some practice, you can read the tree as printed on screen. Later versions of Karoo GP will provide an option to generate an image (similar to the one presented on the cover page) in addition to this text-based format.

Second Time Through

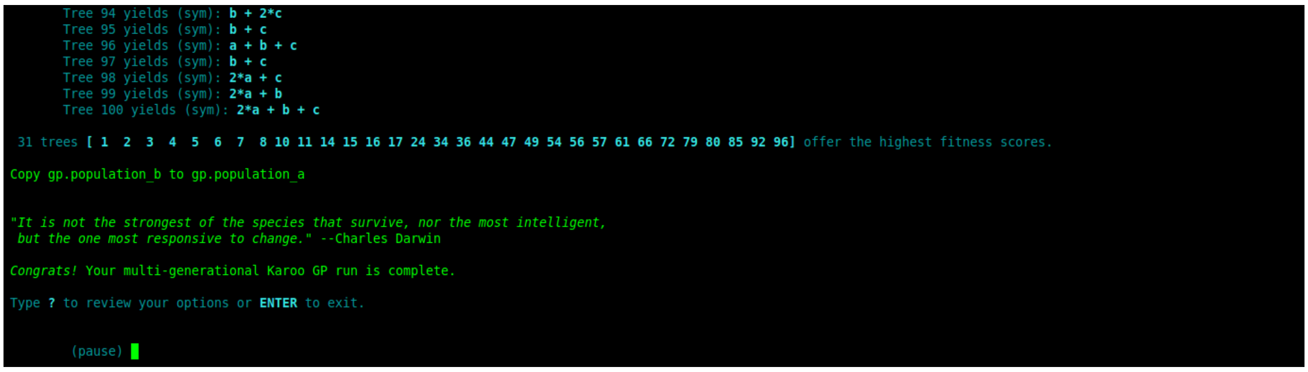
Run Karoo GP again, but this time simply press ENTER at each of the seven queries. Sit back and watch as Karoo GP works to resolve a simple relationship between 3 variables which represent 3 columns of data: a + b + c = s.

We know the answer, but Karoo GP must discover a solution through an evolutionary process. As GP is based on random number generation, there is a chance that in the default ten generations it will not work, or it might find a relationship between the columns of data other than that which was expected.

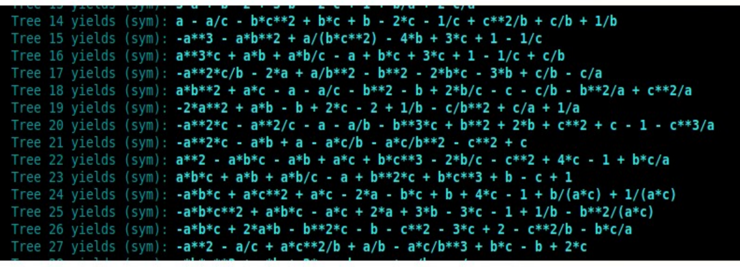
By providing a genetic program with deeper trees, or by increasing the minimum number of nodes required, you can encourage the evolutionary process in genetic programming to discover more complex solutions.

When done, you will be presented with *(pause)*. Type ? and ENTER to review the menu:

1. Type *l* and ENTER to view all trees which provide the correct solution.
2. Type *p*, ENTER, then the unique number of any tree in the list, to print that tree to screen.
3. The solution will be listed in both its raw and Sympyfied format. Remember that multiple trees may carry the same or more than one solution to the same problem.
4. Now you may alter the (b)alance of the operators, (cont)inue the evolutionary process, (test) the trees, or (q)uit.



```
        Tree 94 yields (sym): b + 2*c
        Tree 95 yields (sym): b + c
        Tree 96 yields (sym): a + b + c
        Tree 97 yields (sym): b + c
        Tree 98 yields (sym): 2*a + c
        Tree 99 yields (sym): 2*a + b
        Tree 100 yields (sym): 2*a + b + c

 31 trees [ 1  2  3  4  5  6  7  8 10 11 14 15 16 17 24 34 36 44 47 49 54 56 57 61 66 72 79 80 85 92 96] offer the highest fitness scores.

Copy gp.population_b to gp.population_a


"It is not the strongest of the species that survive, nor the most intelligent,
 but the one most responsive to change." --Charles Darwin

Congrats! Your multi-generational Karoo GP run is complete.

Type ? to review your options or ENTER to exit.


         (pause) █
```



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

Subsequent runs

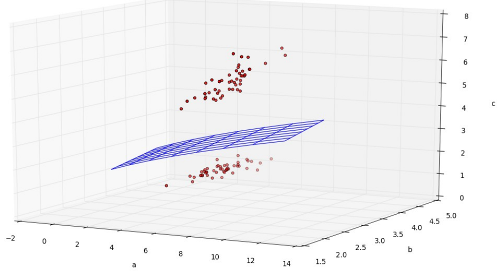
When you are ready to dive in a bit further, take a few minutes to learn about each of the following:

1 - KERNEL SELECT

When prompted with a choice of (a)bs diff, (c)lassify, or (m)atch:

Absolute Value is a minimisation function, meaning it seeks a fitness score with the *lowest* number. By default, this kernel will attempt to solve Kepler's 3rd Law of Planetary Motion. The challenge with this problem is encouraging GP to discover an expression more complex than *t/t*. As the correct expression is *t^2 / r^3*, you can experiment with both deeper trees and increasing the minimum number of nodes. Or, let Karoo GP run for 20, 30, ... 50 or more generations and random mutation might just discover the correct solution.

To learn more, visit: www.physicsclassroom.com/class/circles/Lesson-4/Kepler-s-Three-Laws



Classification is a maximisation function, meaning it seeks a fitness score with the *highest* number. By default, this kernel will attempt to solve the Iris flower problem. This is a machine learning classic built upon data generated before the dawn of DNA classification, when botanists used microscopes and calipers to measure the features of plants. While Kepler's Law has one solution, the Iris problem has many. If you plot one of the final functions over a scatter plot of the data you should find a clean separation a defined by the GP generated function.

To learn more, visit: archive.ics.uci.edu/ml/datasets/Iris

Karoo GP supports multiclass classification from 2 to *n* classes.

Match will attempt to discover the relationship between variables a, b, and c as *a + b + c*, as noted in *Second Time Through* (above). This is also the simplest and most rewarding of the fitness functions to experiment with on your own. Try increasing tree depth and the minimum number of nodes. See *Using Your Own Data* at the bottom of this guide to learn how to replace the default data with your own.

2 - TYPE OF TREE

Full trees have branches that all reach the maximum depth. These trees are less likely to solve problems, as they are not as flexible in their solution space. For example, if the desired solution is a + b + c (5 elements) but a Full tree is confined to 15 elements, some of those elements will be forced to cancel each other in order to arrive to the simpler function. Possible, but more effort and less probable. However, Full trees do contribute to non-linear solutions, a variable multiplied by itself to achieve power or square root functions.

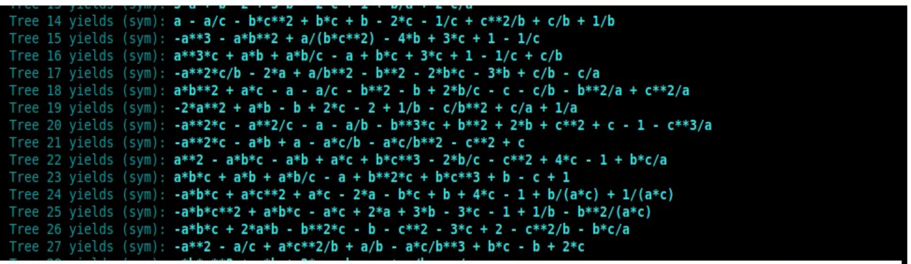
Grow trees have unbalanced branches, meaning some branches reach the maximum depth while others do not. Grow trees are more likely to find simpler solutions.

Ramped Half/Half initialises the first population with 50% Full and 50% Grow, and then applies the Grow method to each population for all subsequent generations. This is the most popular method among GP researchers as it injects a higher degree of evolutionary creativity into the initial population.

3 – MAXIMUM TREE DEPTH

Karoo GP is unique from some flavours of GP in that it sets a user defined maximum tree depth, thereby restricting program bloat. The maximum number of nodes in a tree is defined by: nodes = 2^(depth + 1) – 1

As noted above, deeper trees present greater opportunity for more complex solutions, and they enable the inclusion of a greater number of features from the data in a single polynomial expression.



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

4 - MINIMUM NUMBER OF NODES

The greater the minimum number of nodes (including both variables and operands), the higher degree of complexity to be found in the expressions generated by the trees. The gene pool from which the tournament selection operates is built only from those trees which meet the minimum number of nodes criterion. You may change this number during Karoo GP runs and learn how it affects the evolutionary process.

5 - NUMBER OF TREES

100 trees is a common number for each population, but feel free to experiment with other numbers.

6 - NUMBER OF GENERATIONS

10 Generations will give you a sense of whether or not Karoo GP will quickly find a solution. However, for more complex problems, or if you have not invoked a high minimum number of nodes, sometimes 20, 30, ... 50 or more generations are required for the random process of evolution to generate an improved tree which may then find its way into subsequent generations. *Be careful!* if you set the minimum number of nodes *too* high, you may force a kind of elitism in the population, killing off simpler solutions, or even the entire population.

You may (cont)inue with subsequent generations when all generations have run their course (see bottom).

7 - DISPLAY MODE

The display modes (i)nteractive, (m)inimal, (g)eneration, and (s)ilent are the means by which you interact with and monitor the evolutionary process of Karoo GP. They do not affect the outcome of the process itself.

> **Interactive -** *(pause)* with each step of the Karoo GP process.
>
> **Minimal -** runs through the entire evolution, start to finish without *(pause)*. However, it does display each tree as it is evaluated, which is a lot of fun to watch.
>
> **Generation** – *(pause)* with the end of each generation, allowing you to make adjustments once per generation.
>
> **Silent -** designed to run remotely on a server with limited overhead. This mode presents the results of each generation just to let you know it is still alive and cranking through data while you are engaged in data reductions, Facebook, or a Star Wars role playing game.

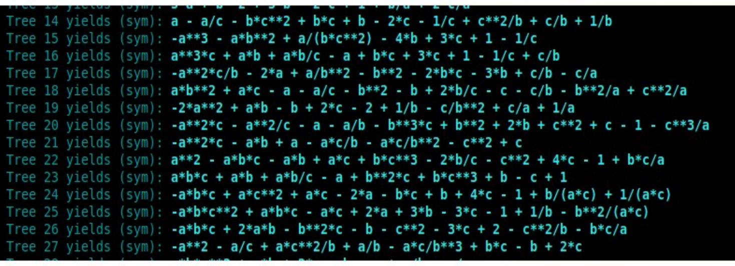
There are 2 additional modes available to you at any time, but listed only with the *(pause)* menu:

> **Debug -** for those of you who want to watch the evolutionary process at the intimate level, this is a fully transparent window to the inner workings of tree-by-tree mutation and crossover reproduction.
>
> **Timer -** allows you to monitor the performance of your current GP run against the current number of CPU cores engaged. At each *(pause)*, you may select (c), the number of cores, and then ENTER to continue. Karoo GP is *multi-core* engaged, meaning it spawns processes across physical CPU cores (not multiple threads on a single core). At the time of this edit, Karoo GP is *not* GPU enabled.

You can switch to any mode at any *(pause)*.

Once you are in *minimal* or *silent* modes, there are no longer opportunities to modify the evolutionary parameters until all generations are complete. However, you may continue the evolutionary process and Karoo GP will pick up where you left off. Simply enter (cont) followed by the number of subsequent generations, and off you go.

If new parameters are desired, remember to set all configurations before you continue.



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

What You See On-Screen

The (pause) Option

Be certain to play around with all the options available to you in *(pause)*. You can change the level of interaction, minimum number of nodes, number of CPU cores; print and test trees, and more.

The Most Fit trees

In Interactive and Minimal display modes, Karoo GP will present to you the outcome of each GP tree as it is generated. In Interactive and Debug modes, you will enjoy a much more detailed, transparent view to what is happening on the inside of the evolutionary process.

At the end of each generation of population evolution, a list of bold numbers is presented on-screen. These are the trees GP has found to have the best overall fitness score in the latest generation. At each *(pause)*, you may *list*, *print*, and/or *test* any given tree, and again at the end of any given run.

What Is Saved Off-Screen

Take a look in files/

population_a.csv includes all foundation population, that is, the original population and all subsequent evolved populations. If you run 10 generations with 100 trees each, you will have a .csv file with 1000 trees.

population_b.csv is filled only if you export a snapshot of the evolutionary process (from the *(pause)* menu) using the (w) command.

population_final.csv is, as its name indicates, the final generation of GP trees and should, if all goes well, offer the most fit trees of all the generations. While this final generation is also included in population_a, this file is isolated in order to make it easier for you to locate and work with these solutions.

The Evolutionary Operators

There are 4 evolutionary operators applied in Karoo GP. The foundation for these evolutionary operators was derived from the "Field Guide to Genetic Programming" by Riccardo Poli, William Langdon, and Nicholas McPhee, with contributions by John Koza. Using the (b)alance option in the *(pause)* menu, you can set the ratio of the operators, such that combined they equal 100%.

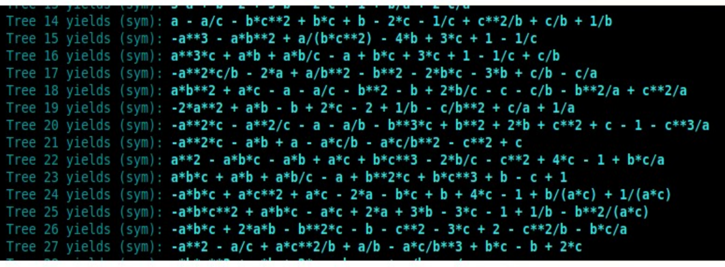
In the following, *tournament selection* refers to the random selection of a number of trees (default 10) whose fitness scores are compared. The tree with the highest score is moved to the next generation through an evolutionary operation. We use tournament selection, not a top-to-bottom evaluation of the entire population, else we risk premature convergence on what may not be the best overall solution, or *elitism*.

Reproduction

Through tournament selection, a single tree from the prior generation is copied without mutation to the next generation. In the biological world, this is analogous to a member of a prior generation directly entering the gene pool of the subsequent (younger) generation.

Point Mutation

Through tournament selection, a copy of a tree from the prior generation mutates before being added to the next generation. In the biological world, this may be analogous to asexual reproduction, that is, a copy of an individual with a minor mutation. In this method, a single point is selected for mutation while maintaining function nodes as functions (operands) and terminal nodes as terminals (variables). The size and shape of the tree will remain identical.



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

Branch Mutation

Through tournament selection, a copy of a tree from the prior generation mutates before being added to the next generation. In the biological world, this may be analogous to asexual reproduction, that is, a copy of an individual but with a potentially *substantial* mutation. Unlike Point Mutation, in this method an entire branch is selected. If the evolutionary run is designated as Full, the size and shape of the tree will remain identical, each node mutated sequentially, where functions remain functions and terminals remain terminals. If the evolutionary run is designated as Grow or Ramped Half/Half, the size and shape of the tree may grow smaller or larger, but it may not exceed the maximum depth defined by the user.

Crossover Reproduction

Through tournament selection, two trees are selected as *parents* to produce a single *offspring* (future versions of Karoo GP will produce 2 offspring per set of parent trees). Within each parent tree a branch is selected. Parent A is copied, with its selected branch deleted. Parent B's branch is then copied to the former location of Parent A's branch, and inserted (grafted). The size and shape of the offspring may be smaller or larger than either of the parents, but may not exceed the maximum depth defined by the user.

This process combines genetic code from two trees, both of which were chosen by the tournament process as having a higher fitness than the average population. Therefore, there is a chance their offspring will provide an improvement in total fitness.

In most GP applications, Crossover Reproduction is the most commonly applied evolutionary operator (60-70%). For those who like to watch, select (d)e(b)ug mode at the launch of Karoo GP or at any *(pause)*.

Working with Data in Karoo GP

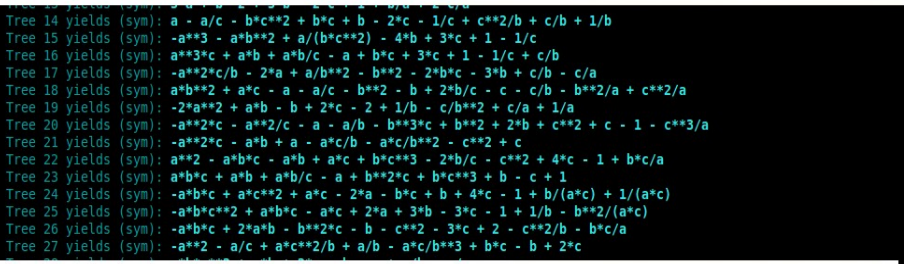
Data and Features

A dataset is the collection of raw data points, one or more values or measurements related to observation of the real world. Features are values extracted from that observational data. In other words, features are *transformed data*. In both cases, each row in an array or file is regarded as a separate sample; each column is the measurement of an observation in the case of a dataset; a feature in the case of a feature set. However, in the world of Machine Learning, a set of features is often referred to as a dataset, even though Machine Learning seldom works with raw, observational data.

Feature selection is the process of selecting a subset of relevant variables which best describe the dataset as a whole, as many features may not be rich in discriminatory power. Feature selection helps reduce the dimensionality of the data in order that the classifier is able to more easily learn. This principle is called the “curse of dimensionality”.

Feature extraction works with observational datasets to build new, derived, informative values or relationships between the selected features for use in Machine Learning applications such as Support Vector Machines, Genetic Programming, or Artificial Neural Networks. The relevance of each feature cannot be expressed with all the classes together. Sometimes one feature might be relevant for one class but not another. This presents a more challenging problem which demands a stronger understanding of the data statistics.

The Iris dataset included with Karoo GP is very simple, the features being the real-world measurements themselves. Yet with something more complex, such as text mining, one might begin with the raw accumulation of thousands of SMS, email, Twitter feeds, etc., remove low-value words (i.e. articles, conjunctions), and build new features through the retention of certain sized blocks of words (i.e. n-grams), or through the application of other rules which generate valuable features.



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

Replacing the Default Data

Karoo GP is designed for easily repeatable experimentation. When you launch Karoo GP, data and function files associated with the user selected kernel are auto-loaded. For example, if you select "c" for classification, then "files/data_CLASSIFY.csv" and "files/functions_CLASSIFY.csv" are loaded accordingly, unless you specify another .csv file at launch.

When you are ready to apply your own dataset, you can replace the data in one of the .csv files in the files/ directory (keeping the file name the same), or at launch, point Karoo GP to a properly prepared .csv which contains your data, as follows:

(from a native shell): $ python karoo_gp.main.py /[directory]/[your_file].csv

(from iPython): $ run karoo_gp_main.py /[directory]/[your_file].csv

A few things to keep in mind:

1. If you generate your own feature set, be certain to save the original spreadsheet as .ods or .xls in addition to the .csv in order to preserve any formula, notes, or formatting for later reference or modification.
2. If you replace the contents of an existing .csv with your own features, pay attention to the following:
   - If editing the .csv using a text editor, do NOT add spaces between the variables and the comma.
   - Label each column with a variable. For example: *a,b,c* or *a1,a2,a3* or give each column a variable which represents the real-world measurement, such as *t* for time or *p* for pressure.
   - The right-most column must be labeled "s" (lower case, no spaces before or after).
3. Be *very* wary of your editor applying hidden characters in the .csv as these will cause Karoo to crash.
4. If sending a .csv file through email, wrap it in an archive (.zip, .tar.gz) else data corruption might occur.

NOTE: *Your .csv must conform to the format described in the above paragraphs or bad things will happen.*

Data Size

In theory, the size of the .csv file is limited only by the amount of RAM in your computer … and your patience. A practical run of GP is often against 10,000 rows of data in a given .csv file. You can imagine that this takes some time, far more than the simple Kepler and Iris examples given.

Train vs Test

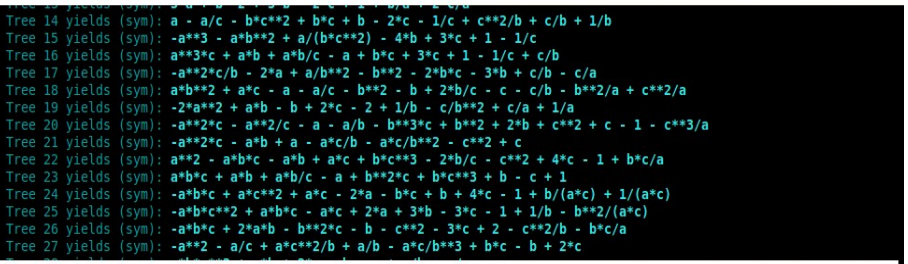
Karoo GP auto-splits all datasets greater than 10 lines into TRAINING and TEST in order to offer data on which to test that is unique from the training. However, if you present Karoo with 10 or less rows in your .csv, it automatically copies the same data into both TRAIN and TEST as it is assumed you are using Karoo GP in an instructional or experimental mode.

Terminals (Variables and Constants)

Karoo GP automatically extracts the terminals from the top of the data .csv file. This enables you to name each column of your data such that when you read a GP derived polynomial, you understand the correlation to the data.

You can also inject constants into the mix. In the current implementation of Karoo GP, you simply add the desired constants at the top of new columns in the data .csv. The entire column associated with each constant is then filled with zeros. These zeros are space fillers which have no impact on the processing of the data or function.

For example, you would replace what would be a variable with a constant value "0.5" or "1" or "5".



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

Functions (Operands)

The functions (operands) applied to Karoo GP are designated by you. The most often used operands are:

+, -, *, /, cos

Note that with all files/functions_[name].csv all operands are followed by an *arity*, that is, the number of variables or constants the operand expects to work with.

For example, * has an arity of 2, meaning it expects something before and after itself, as in the expression:

a * b

Nearly all operands have an arity of 2, with the exception of the boolean operators *not* which has an arity of 1 and *if* which has an arity of 3, as in *if a then b, else c*. There is a list of many operands in the files/templates/ directory. Not all have been fully tested as of the writing of this document. The limitation is the ability for the Python library SymPy to process them. To learn more about SymPy and its capabilities, visit www.sympy.org

Please note that with Karoo GP, all trig functions must be preceded by another operand, as in *+ cos* such that if you introduce *cos* to your list of operands, you will likely want to include all 4 of the arithmetic operands:

```
+ cos,2
- cos,2
* cos,2
/ cos,2
```

The power function (^ or **) is not necessary, as this is generated by any variable multiplied by itself once or twice. In the real world, most non-linear laws of nature are based on powers of 2 or 3, not 5, 12, or … 132. As such, GP finds these quite easily without the explicit operand. What's more, the introduction of ** often results in a stalled program, as the random nature of genetic programming will quickly generate a multi-nested exponent which totally stalls the CPU.

Square root too can be discovered through * and /, but you can attempt to introduce this to Karoo GP and see what happens.

Finally, keep in mind that the quantity of each operand in your list affects its chance of being selected. If you desire for *cos* to have the same chance at being selected as the basic arithmetic operands, then you should duplicate the arithmetic operands at least 3 times, followed by one instance of the *cos* set in functions_[name].csv

Loading Prior or Seed Populations

With Karoo GP it is possible to load a former population, or a hand-crafted seed population at any *(pause)*. This enables you to load prior, archived populations and apply a new series of tests, or to carry forward with an evolutionary run with a new set of parameters to test a new hypothesis. What's more, it allows you to start with the same population and test any number of evolutionary parameters over varying generations in order to determine which enables the most rapid convergence on a desired solution.

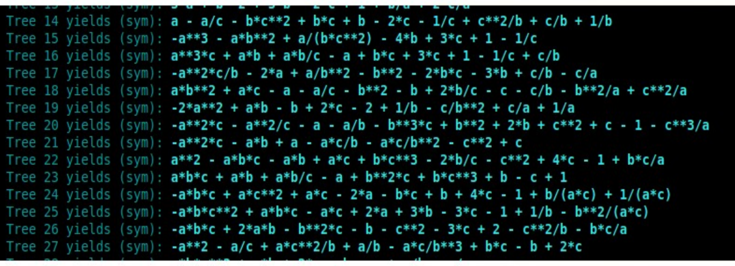
Archive & Restore

To archive a population, simple copy files/population_f.csv to a safe place.

When you desire to restore a population, rename the prior as population_s.csv and move it back into the files/ directory. From the *(pause)* menu, select (load) and immediately, the former population_s will be copied to population_a which is the foundation population for the next generation.

If you desire to load an archived population at the very start, choose (g)eneration, (i)interactive, or (s)ilent mode. At the first *(pause)*, conduct the load and then press ENTER to go.

NOTE: *The Karoo GP population files are very specific in their format. Do not alter the file structure, in any way.*



```
Tree 14 yields (sym): a - a/c - b*c**2 + b*c + b - 2*c - 1/c + c**2/b + c/b + 1/b
Tree 15 yields (sym): -a**3 - a*b**2 + a/(b*c**2) - 4*b + 3*c + 1 - 1/c
Tree 16 yields (sym): a**3*c + a*b + a*b/c - a + b*c + 3*c + 1 - 1/c + c/b
Tree 17 yields (sym): -a**2*c/b - 2*a + a/b**2 - b**2 - 2*b*c - 3*b + c/b - c/a
Tree 18 yields (sym): a*b**2 + a*c - a - a/c - b**2 - b + 2*b/c - c - c/b - b**2/a + c**2/a
Tree 19 yields (sym): -2*a**2 + a*b - b + 2*c - 2 + 1/b - c/b**2 + c/a + 1/a
Tree 20 yields (sym): -a**2*c - a**2/c - a - a/b - b**3*c + b**2 + 2*b + c**2 + c - 1 - c**3/a
Tree 21 yields (sym): -a**2*c - a*b + a - a*c/b - a*c/b**2 - c**2 + c
Tree 22 yields (sym): a**2 - a*b*c - a*b + a*c + b*c**3 - 2*b/c - c**2 + 4*c - 1 + b*c/a
Tree 23 yields (sym): a*b*c + a*b + a*b/c - a + b**2*c + b*c**3 + b - c + 1
Tree 24 yields (sym): -a*b*c + a*c**2 + a*c - 2*a - b*c + b + 4*c - 1 + b/(a*c) + 1/(a*c)
Tree 25 yields (sym): -a*b*c**2 + a*b*c - a*c + 2*a + 3*b - 3*c - 1 + 1/b - b**2/(a*c)
Tree 26 yields (sym): -a*b*c + 2*a*b - b**2*c - b - c**2 - 3*c + 2 - c**2/b - b*c/a
Tree 27 yields (sym): -a**2 - a/c + a*c**2/b + a/b - a*c/b**3 + b*c - b + 2*c
```

Write Your Own Fitness Function

Advanced users may modify existing fitness function kernels or create their own. This Quick Start Tutorial does not provide a detailed guide, rather a pointer to the places in Karoo GP which require modification, as follows:

- In karoo_gp_main.py, add the new kernel to the TUI query for gp.kernel
- In karoo_gp_base_class.py:
  1. In the method fx_karoo_data_load, modify the dictionaries: data_dict, func_dict, and fitt_dict
  2. In the method fx_fitness_gym, search for "[other]" and modify.
  3. In the method fx_fitness_eval, search for "[other]" and modify.
  4. Now build your new fitness function as a Python method. Search for "def fx_fitness_function_[other]" to find the place-holder. Keep in mind that due to the multi-core functionality of this particular section, it is imperative that your sum be passed back through the pprocess portal, not saved as a global variable. Refer to the other fitness functions for examples.

Additional Scripts

Karoo GP Server

While this Quick Start Tutorial discusses karoo_gp_main.py, the server version karoo_gp_server.py performs identical functions but bypasses the Text User Interface at launch, instead calling upon the hard coded values in the configuration file. Using this format, it is relatively simple to automate multiple runs of Karoo GP with variations on the parameters at each run, including the test of various depths of trees, minimum number of nodes, or even back-propagation from a GA or other, external analysis application.

A future version of Karoo GP will include a command line, argument-driven version for fully enabled scripting.

Karoo Features Sort

Inside the tools/ directory, you will find karoo_features_sort.py. This script prepares a clean, 50/50 dataset from an otherwise imbalanced set of binary classifications. In other words, if you have 1.3M rows of data, with only 5% being class 1, this script prepares 5,000 rows of class 0, and 5,000 rows of class 1 data points, no 2 reused. It can be adjusted to fewer or more than 10,000 rows, and with some modification, made to work with more than 2 classes.

Karoo Iris Plot

Inside the tools/ directory, you will find karoo_iris_plot.py. If you don't mind fiddling with the algebraic expressions generated for the Iris dataset, you can produce a 3D overlay of an expression against a scatter plot.

Karoo Multiclass Classifier

Inside the tools/ directory, you will find karoo_multiclassifier.py. This script was built to test the auto-scaling multiclass classification algorithm. It is just so much fun to play with, I included it for your shared pleasure.

Experiment! Explore! Evolve!

You cannot damage Karoo GP by experimenting. If you screw up a data file, either Karoo GP or Python will complain (or just come to a screeching halt). Keep backups of your data and functions. Save templates of those that work well. But most of all, have fun! And if you find any errors in the code (there may be a few), do not hesitate to contact me. I will do my best to respond quickly --kai