A number of example problems are currently provided with PonyGE2:

1. [[String-match|Example-Problems#string-match]]
2. [[Regression|Example-Problems#regression]]
3. [[Classification|Example-Problems#classification]]
4. [[Pymax|Example-Problems#pymax]]
5. [[Program synthesis|Example-Problems#program-synthesis]]
6. [[Genetic Improvement of Regex Runtime Performance|Example-Problems#genetic-improvement-of-regex-runtime-performance]]
7. [[Multiple Objective Optimisation|Example-Problems#multiple-objective-optimisation]]

A brief description is given below of each problem, along with the command-line arguments necessary to call each problem. The developers of PonyGE2 encourage users to test out the various different operators and options available within PonyGE2 using these example problems in order to gain an appreciation of how they work.

String-match

The grammar specifies words as lists of vowels and consonants along with special characters. The aim is to match a target string.

To use it, specify the following command-line argument:

```
--parameters string_match.txt
```

The default string match target is `Hello world!`, but this can be changed with the `--target` argument.

Regression

The grammar generates a symbolic function composed of standard mathematical operations and a set of variables. This function is then evaluated using a pre-defined set of inputs, given in the datasets folder. Each problem suite has a unique set of inputs. The aim is to minimise some error between the expected output of the function and the desired output specified in the datasets. This is the default problem for PonyGE.

To try this problem, specify the following command-line argument:

```
--parameters regression.txt
```

The default regression problem is `Vladislavleva4`, but this can be changed with the `--grammar_file`, `--dataset_train` and `--dataset_test` arguments.

Optimisation of constants

When running supervised learning problems like regression and classification, PonyGE2 can attempt to optimise constants. These constants will exist in the individual's phenotype string as `c[0]`, `c[1]`, etc., so the grammar should create such terms. Then, if we specify the following argument:

```
--optimize_constants
```

or set the parameter `OPTIMIZE_CONSTANTS` to `True` in either a parameters file or the params dictionary, then PonyGE2 will perform a gradient descent on the values of the constants (given the data) to minimise the error (as measured by the chosen `--error_metric`). It will use the L-BFGS-B method. By default, optimisation of constants is switched off.

Classification

Classification can be considered a special case of symbolic regression but with a different error metric. Like with regression, the grammar generates a symbolic function composed of standard mathematical operations and a set of variables. This function is then evaluated using a pre-defined set of inputs, given in the datasets folder. Each problem suite has a unique set of inputs. The aim is to minimise some classification error between the expected output of the function and the desired output specified in the datasets.

To try this problem, specify the following command-line argument:

```
--parameters classification.txt
```

The default classification problem is `Banknote`, but this can be changed with the `--grammar_file`, `--dataset_train` and `--dataset_test` arguments.

Pymax

One of the strongest aspects of a grammatical mapping approach such as PonyGE2 is the ability to generate executable computer programs in an arbitrary language [[[O'Neill & Ryan, 2003|References]]]. In order to demonstrate this in the simplest way possible, we have included an example Python programming problem.

The `Pymax` problem is a traditional maximisation problem, where the goal is to produce as large a number as possible. However, instead of encoding the grammar in a symbolic manner and evaluating the result, we have encoded the grammar for the `Pymax` problem as a basic Python programming example. The phenotypes generated by this grammar are executable python functions, whose outputs represent the fitness value of the individual. Users are encouraged to examine the `pymax.pybnf` grammar and the resultant individual phenotypes to gain an understanding of how grammars can be used to generate such arbitrary programs.

To try this problem, specify the following command-line argument:

```
--parameters pymax.txt
```

Program synthesis

The General Program Synthesis Benchmark Suite is available in PonyGE2. Grammars and datasets have been provided by HeuristicLab.CFGGP. The individuals produce executable Python code.

To try this problem, specify the following command-line argument:

```
--parameters progsys.txt
```

***NOTE** that [[multicore evaluation|Evaluation#multicore-evaluation]] is **not** currently supported for this type of problem.*

Genetic Improvement of Regex Runtime Performance

An example of Genetic Improvement for software engineering is given for program improvement of Regular Expressions (regexs). The given examples improve existing regexes by seeding them into the population. The fitness function measures runtime and functionality of regexs.

The fitness function for the regex performance improvement problem has a number of sub-modules and programs which are used for generating test cases and for timing the execution of candidate programs. This is an exmaple of how a fitness function can be extended in PonyGE2 with customised modules and classes to perform complex tasks.

To try this problem, specify the following command-line argument:

```
--parameters regex_improvement.txt
```

***NOTE** that [[multicore evaluation|Evaluation#multicore-evaluation]] is **not** currently supported for this type of problem.*

Multiple Objective Optimisation

As detailed in the section on [[multiple objective optimisation|Evaluation#multiple-objective-optimisation]], PonyGE2 comes as standard with an implementation of NSGA2 to optimise multiple fitness functions simultaneously.

Two example problems are provided:

1. Regression with bloat control

   This example runs the standard Vladislavleva-4 [[regression|Example-Problems#regression]] example, but with a second fitness function that aims to minimise the number of nodes in the derivation tree. To try this problem, specify the following command-line argument:

```
--parameters moo/regression_bloat_control.txt
```

***NOTE** that it is advisable to use [[subtree mutation|Variation#subtree-1]] in combination with the [[mutate duplicates|Evaluation#mutate-duplicates]] option with this example problem.*

2. An example benchmark multi-objective optimisation problem from [[[Zitzler, et al.|References]]] is provided. To try this problem, specify the following command-line argument:

```
--parameters moo/zdt1.txt
```

Adding New Problems

It has been made as simple as possible to add new problems to PonyGE. To add in a new problem, you will need:

1. a new [[grammar file|Grammars]] specific to your problem,
2. a new [[fitness function|Evaluation#the-fitness-function-template]] (if you don't want to use a previously existing one), and
3. if you are doing supervised learning then you may also need to add some new [[datasets|Evaluation#datasets]].

A [[template|Evaluation#the-fitness-function-template]] for new fitness functions is provided in `fitness.base_ff_classes.ff_template`. This template allows for new fitness functions to be easily and quickly created in an easily compatible and extensible manner.

***NOTE** that it may be beneficial to create a **new parameters file** for any new problem.*