Zachary Griffin (zg85), Gina Hsu (gh334), Liane Longpre (lfl42)

Final Project Design Document

System Description

We plan on implementing a regression analysis and graphing tool. Users will be able to input a set of data points or function. Our system will perform regression analysis and produce a best fit function, a graphical representation, and statistical information about the function.

Key features:

- Given a dataset input
  - Create a regression function
  - Create a graphical representation of the regression function
  - Provide statistical measures of the regression function (e.g. R^2 value)
- Given a function input
  - Create a graphical representation
  - Create randomized datasets with user-input parameters

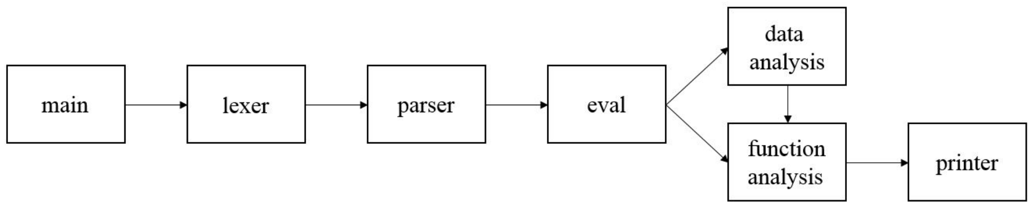
Description of System:
The user will interact directly using the terminal. We plan on using a REPL similar to the one implemented in the A2 Adventure Game. The user will be given a list of possible commands as well as a help menu with example inputs. Dataset inputs will be given as a CSV file. Graphical representations will be saved as a .jpg file in the current directory. We plan on using third-party libraries to parse CSV files and to create the graphical representation.

We plan on first implementing simple linear regression. Our plan gives rise to a number of extensions of which we plan on exploring as many as time permits:

- Multivariate linear regression
- Polynomial regression
- Logistic regression
- Probability distributions (i.e. binomial, cumulative, Poisson, etc)

Architecture

Our system will resemble a pipe and filter architecture.
C&C Diagram:

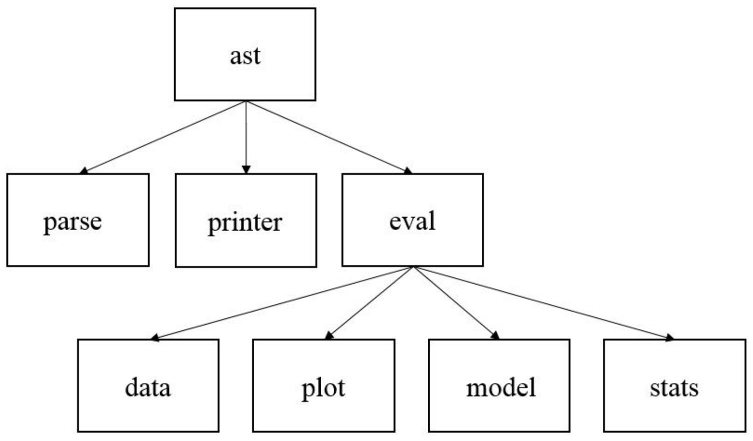


System Design

Modules:

- Main: REPL and help menu for user
- Parse: Parses user input commands
- Ast: Declares types of expressions
- Eval: Takes a parsed command and calls the appropriate modules and functions accordingly
- Data: Performs regression on inputs of datasets
- Stats: Performs statistical analysis on inputs of functions
- Plot: Creates graphical representations of inputs of functions or datasets
- Model: Creates datasets from inputs of functions and datasets
- Printer: Prints result of command to user

Dependency Diagram:



Module Design

Our AST module defines type command, which differentiates different types of commands from our user. These will be used to pattern match which modules to call when eval is called on the command. Our parse module parses string inputs into commands, and depends on a lexer and parser. Our eval module declares a type result that is a variant distinguishing different types of results based on user commands (e.g. graph, function, etc). Our data, plot, stats, and model modules will contain functions.

Data

Structures:

- Linear functions are maintained as float tuples as so: (slope, intersect)
- Datasets are maintained as float tuple lists as so: [(x, y) ; …]
- Statistics are maintained as float tuples as so: (R^2, standard error)
- Commands are maintained as variants as so:
    - LinReg: take dataset, produce function
    - GraphFunct: take function, produce graph
    - GraphData: take dataset, produce graph
    - Stats: take function and dataset, produce statistics
    - Model: take function, produce dataset
- Results are maintained as variants as so:
    - Funct: function result
    - Dataset: string to be outputted to user (e.g. "dataset has been saved as 00.csv")
    - Graph: string to be outputted to user (e.g. "graph has been saved as 01.jpg")
    - Stats: statistics results

For communication with the user, we utilize a read-evaluate-print loop, which calls parse, eval, and printer respectively. The user will input string commands which we will later parse. For all commands, we will return a string in the REPL containing the result of the desired operation. In the case of a graphing command, we will additionally save the graph as a .jpg in the user's current directory. In the case of a modeling command, we will save the resulting dataset as a .csv file in the current directory as well. We will utilize imperative programming to create file names, so as not to overwrite previously used names (e.g. graph0.jpg -> graph1.jpg) The outputs will not be stored, so in the case where a user wants to graph a function or dataset that has been returned, they must then pass in the value or file to the next command.

External Dependencies

We will make use of the following third-party libraries:

- OCaml-csv: CSV parser for dataset inputs
- PL Plot: To create graphical representation of given function

Testing Plan

We plan on first creating a general test harness together as a team before beginning to implement any of the project. This will help with understanding our modules and types. It will also clarify what steps will need to be taken to begin implementation.

Once we begin implementation, we plan on continuing to add to the test harness in the manner:
If a member implements the code for a certain function, then that member will create tests for each possible path the implementation might take. This will handle glass box testing. A different member that did not implement the code for that function will then create tests for different input types and corner cases, which will handle black box testing. This plan ensures that members hold each other accountable for writing correct code and for thorough testing.