About eGebra

Venkat Arun

January 1, 2015

Abstract

eGebra is a generic front end for symbolic manipulation systems such as symPy, Mathematica^{tm}, Macsyma etc. Designed to bring the best of human creativity and computers' speed and accuracy, it provides a platform and back-end cas system independent interface for all practitioners of mathematics to intuitively study and manipulate symbolic math. eGebra tries to make full use of computing power available on today's machines by automatically scheduling tasks without the user needing to prompt for them, while giving higest priority to user scheduled tasks when they exist. This ensures the user can quickly know a lot about whatever expressions they are manipulating. Further, it has a client server model which can be deployed either as an internet service or which can be used to exploit any powerful computing systems that the user may own while having an intuitive interface on almost any device that has an internet browser.

1 Introduction

The user interface allows for the creation of symbolic objects. It currently supports symPy, mathematica and maxima languages (all of which are supported by symPy). Also, open source graphical editors exist for editing equations which can be integrated in the future. Further, in principle, a parser can be made for a subset of Latex.

After creating objects the system automatically starts performing some common computations. The user can explicitly ask for some computations to be performed in which case they are done with top priority.

In effect, as soon as the object is created, a fan of computed properties such as the graph (for a function), roots, factorization etc. are displayed. The user can then search (in plain english, therefore removing the necessity for memorising function names) for computations and then request for them to be performed.

2 Unique Features

So what is so cool about this project? A lot of things (well, there may be some bias here :)). Below are some capabilities that I envision for eGebra for which (I believe) no satisfactory solution exists in the market:

1. An intuitive interface for mathematical symbolic manipulations
2. A system that greatly reduces computation mistakes made by humans and keeps a diligent track of all assumptions made in a derivation.
3. A CAS system with natural input methods that properly exploit touch-screens of modern devices
4. A CAS interface that does not require any 'coding' (ie. the writing of things in a specific format so that a computer may understand)
5. A CAS system that automatically does some common computations without the user needing to prompt for them explicitly
6. A unified view to all popular CAS systems available in the market such as symPy, Macsyma, Mathematica^{tm}, Maple^{tm}, Mathics etc.
7. A client-server system that allows computations to be done on a powerful computer but yet allows easy access on almost every device
8. A system that allows for proper utilization of extensive computation facilities available most of which remains unutilized
9. Effective 'cloud' based solutions to symbolic mathematics
10. In summary: An intuitive, unified, powerful and robust system that brings the best of human and computer's capabilities and allows traditional 'paper and pen' mathematics to be completely replaced by this 'electronic mathematics'

3 The Software

This is a partly technical description of eGebra's design and capabilities (existing and planned). A user documentation will be made once it reaches some level of maturity. Currently eGebra uses symPy as its CAS (computer algebra system) backend, but it is carefully designed so that other systems can be readily plugged in without changing the most of the code. eGebra is currently written in pure Python. This may change when wrappers are introduced to link to CAS software other than SymPy.

The backend core of eGebra is its server. The server has two main classes of entities of interest: symbolic objects and computations. Symbolic objects represent mathematical objects such as variables, equations, expressions, graphs, list of roots etc. that the user may wish to manipulate. Computations are, as the name suggests, operations that can be performed on these symbolic objects such as differentiation, eigenvalue-decomposition, substitution etc. Generally computations lead to the creation of new symbolic objects from old ones and this relation is maintained as explained below.

3.1 Symbolic Objects

The CAS backend specific parts of eGebra provide an abstraction the rest of the eGebra system which enables it to operate without knowing which backend (such as symPy, Mathematica^{tm}, Macsyma, Maple^{tm} ...) is being used. Further, associated with each symbol is information about which other symbolic objects it is related to. For istance, an expression may be the indefinite integral of an object, it could be the determinant of another, or it may have a list of its roots (which will be another symbolic object). The relationships are labeled by the type of computation (or user input) used to generate one of the symbols from the other(s).

Symbolic objects and their relationships are organised into notebooks/workspaces which further are considered to belong to specific users. This also enables the symbolic objects to be encrypted if security is a concern in an online service (not implemented yet).

3.2 Computations

The available computations obviously depend on the CAS backend being used. Again our CAS wrapper abstracts the computations as a set of string identifiers for each operation, a verbose description of each operation and the number, type and description of operands required and the results generated. Further modules then merely pass the 'Symbolic Object' objects along with the operation identifier string to the wrapper to get the new symbolic object(s).

Now the question of scheduling computations arises. At any point in time the UI informs the server about which symbolic objects the user is focussing on. Then a prioritiser (not implemented yet) ranks operations to be performed on each of them. These are then passed to the scheduler with a low priority. Any computations that the user explicitly asks for are also given to the scheduler with a very high priority.

The scheduler is very simple. It simply assigns tasks to the available threads according to the priority of the tasks and the results are pushed back to the UI as soon as they are available. Note that the prioritizer has not been implemented yet but a basic 'stupid' version of it should not take too much time to implement.

4 Current Status of the Project

This project is still at its infancy. One version has already been implemented where both ther back-end and front-end used to work enough to be demonstrated. A newer version is being constructed that uses the django server (the earlier one was implemented using a BaseHttpServer as I was not aware of how complete Django is). Currently the wrapper for sympy and abstractions to a generic system are fairly complete. Most backend services and the database management system are functional. Those that aren't merely need a function to be written in views.py.

The front end was fairly complete before it started behaving badly. It sports the graph plotting algorithms provided by 'arbor' along with computation search and request systems. That's all for now. Watch this document for further updates.