The PolyWiz Programming Language

Tamjeed Azad, Rose Chrin, Max Helman, Aditya Kankariya, Anthony Pitts

You're a wizard, Stephen!

The PolyWiz Team

Rose Chrin
Group Leader
World's Best Boss

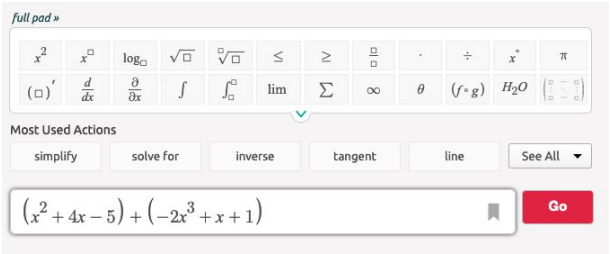
Aditya Kankariya
Language Guru
Professional Artist

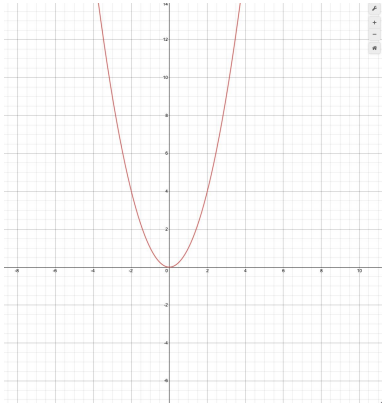
Tamjeed Azad
Systems Architect
Grammar Stickler

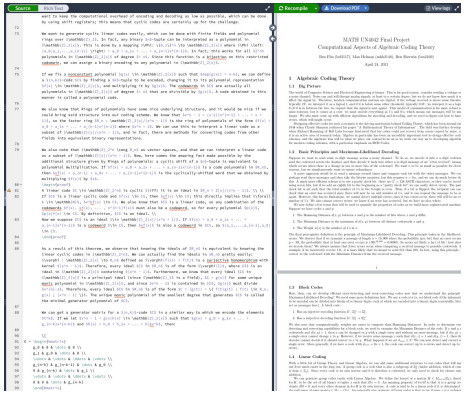
Max Helman
Systems Architect
Night Owl

Anthony Pitts
Tester
Early Bird

The PolyWiz Programming Language

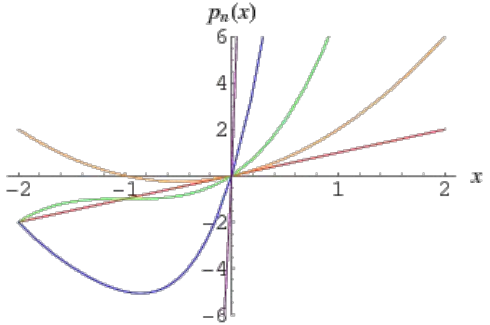






The PolyWiz Programming Language

- A language for symbolic mathematics, focused on a Polynomial datatype for easy symbolic manipulation
- Polynomial datatype supports easy plotting and LaTeX integration
- Strongly and Statically Typed, Statically Scoped, Imperative Language
- File ending: .wiz



Poly, The Heart of PolyWiz

- Robust support for polynomial operations
  - Addition (+), Subtraction (-), Multiplication (*), Division (/)
  - Polynomial Composition & Evaluation
  - Constants Retriever (poly1 #) & much more...
- Simplified syntax to work with polynomials
  - Instead of evaluating a polynomial via loops, powers, several local variables, etc...
    - poly1 @ 8
  - Instead of composing two polynomials via convoluted powers, summation, etc…
    - poly1 : poly2

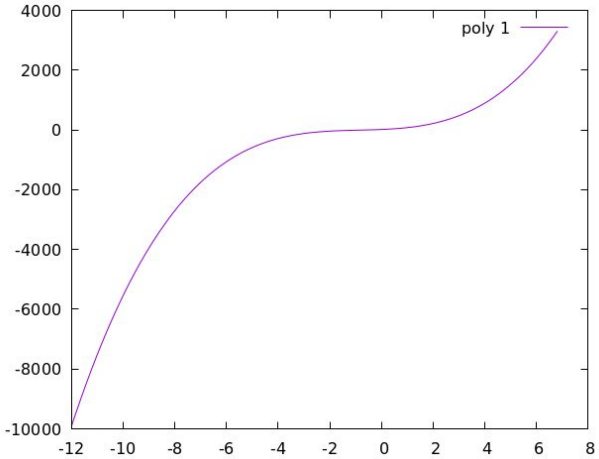
Poly Data Type Implementation

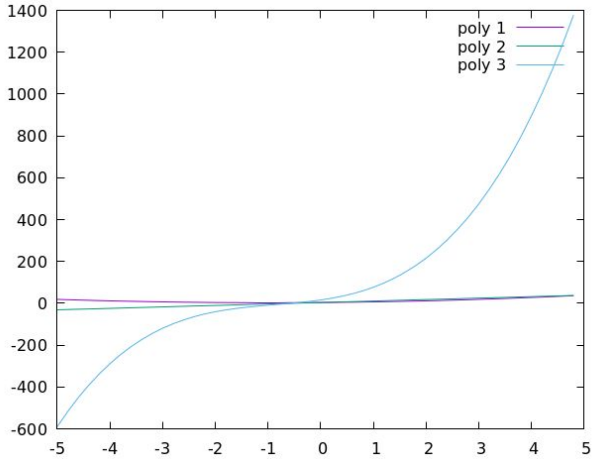
- poly is represented as a float array
  - The ith element represents the constant multiplied by x^i
  - length of poly array = order of the polynomial
  - example: $3x^2 + 5x - 4.2$ is represented as [-4.2, 5, 3]
- Pros & Cons of this implementation
  - **Efficient**. All coefficient lookup is O(1) because indexing specification.
    - This speed allowed for optimal implementations of all poly operations
  - In rare cases, this can waste space
    - $3x^{25}$ is represented with twenty five 0's, followed by a 3.
    - Then again, any array with an order > 10 without other terms is quite impractical.

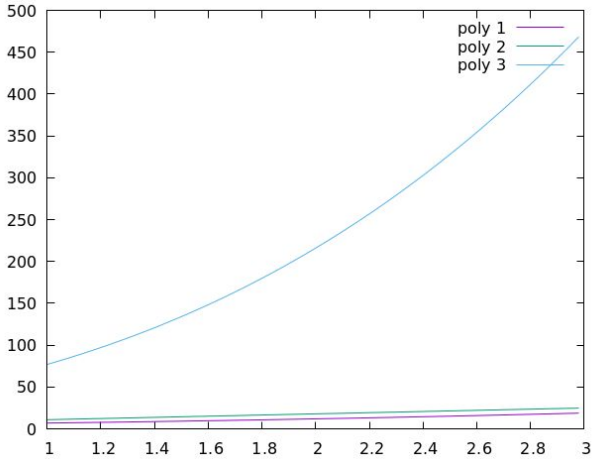
Plotting Implementation

- Two functions: plot and range_plot.
  - Plot takes in array of polys, filepath to desired output location, and plots at default x-value range of -5.0, 5.0.
  - Range_plot has same inputs and output style, but also requires definition of an x-value range.
- Uses gnuplot command line program to implement plotting. System calls using linked C function library.
- Evaluates polynomial(s) at many points to generate a temporary text file, then generates a temporary companion plotting script based on how many polynomial functions are present, then feeds this plotting script to gnuplot to make a png plot. Both temporary files removed after execution.

Plot Examples







LaTeX Implementation :D

- It’s in LaTeX so it must be true.
- The print_tex function takes in a Poly and outputs it in format suitable for LaTeX math mode, including wrapping exponents in curly brackets and wrapping the entire expression in dollar signs.
- The tex_document function takes in a list of strings and a list of indices for filepaths. It wraps the entire list in a LaTeX document, placing each element on a new line and wrapping the filepaths in LaTeX code to display images.
- This was mostly implemented with string manipulation in C… pain

A Sample PolyWiz Program

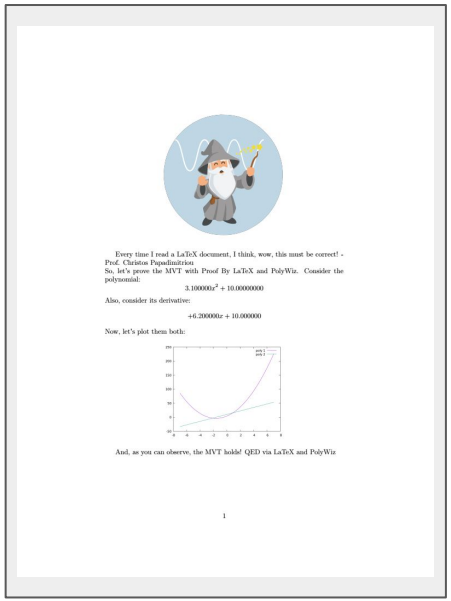
- Let’s use PolyWiz to verify the Mean Value Theorem and output this verification in LaTeX!
- Program can be viewed on GitHub: tests/test-complex_program.wiz

A Closer Look at the Output...

PolyWiz Output


```
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\begin{figure}[h]
\centering
\includegraphics[width=2.5in]{polywizard.png}
\label{fig_sim}
\end{figure}
Every time I read a LaTeX document, I think, wow, this must be correct! - Prof. Christos Papadimitriou\\
So, let's prove the MVT with Proof By LaTeX and PolyWiz. Consider the polynomial:\\
$$3.100000x^{2}+10.000000x+4.000000$$\\
Also, consider its derivative:\\
$$+6.200000x+10.000000$$\\
Now, let's plot them both:\\
\begin{figure}[h]
\centering
\includegraphics[width=2.5in]{complexprogram_plot.png}
\label{fig_sim}
\end{figure}
And, as you can observe, the MVT holds! QED via LaTeX and PolyWiz\\
\end{document}
```

LaTeX Compiler



Every time I read a LaTeX document, I think, wow, this must be correct! - Prof. Christos Papadimitriou
So, let's prove the MVT with Proof By LaTeX and PolyWiz. Consider the polynomial:

$$3.100000x^2 + 10.000000$$

Also, consider its derivative:

$$+6.200000x + 10.000000$$

Now, let's plot them both:

And, as you can observe, the MVT holds! QED via LaTeX and PolyWiz

1

LaTeX Output

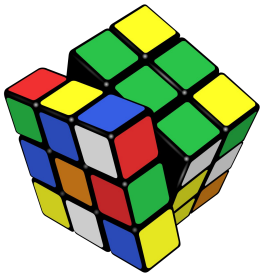
Future Additions

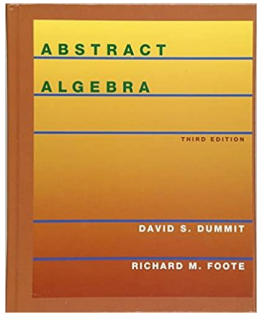
“Hey Prof, I just have to finish my compiler and then I can get to grading.”

“Finish your compiler? That’s a bit open ended, no?”

Future Additions

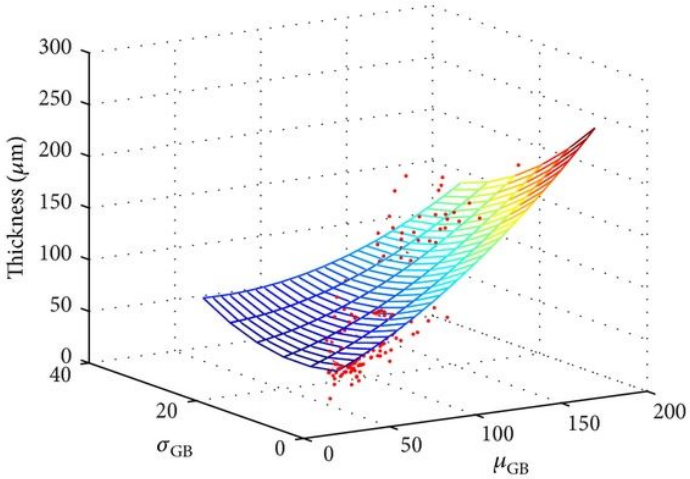
- While doing a project on Algebraic Coding Theory for another class, I (Max) realized that PolyWiz would be perfect for implementing coding schemes if it had a numeric type for elements in a Finite Field.
- We could also give the option to specify that the coefficients in a polynomial must come from this numeric type.
- Essentially, we could add some more sophisticated math behind the polynomials; this would also make our language suitable for cryptographic applications.





Future Additions

- It would be awesome to add support for Multivariate polynomials; gnuplot has a built-in 3D graphing library that we would take advantage of. However, we would need to change some underlying implementations of the polynomial data type.
- We could also add functionality to allow for plot outputs in formats beyond .PNG files. Gnuplot natively supports an incredibly large number of formats beyond this, which could be exploited.
- We would like to build native support for working with data. PolyWiz would be great for things like regression coefficient estimation. This would likely require matrices as well...



Takeaways

On the first day of class, a wise man gave us some advice...

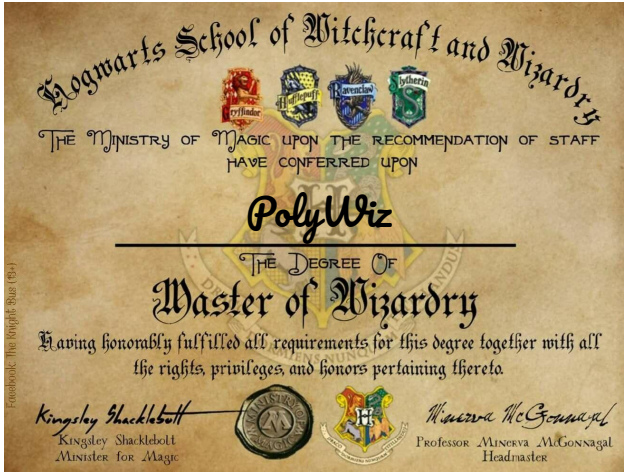
“Implement each part separately. It will work when you put it together.”	We tried this. It did not work well. We will likely never do this again in the future on any project.
“Write software communally. That way, nobody is at fault.”	Pair programming was actually very helpful. However, it helped to have a “point person” for each bug (feature).
“Ignore other members’ opinions: you’re always right; they’re always wrong.”	Some of our best features and design choices came from having a second set of eyes on things. As a corollary, some absolutely terrible features and design choices were avoided this way.

Takeaways

Some other takeaways that we were not warned about on the first day:

-Learn from the greats. Some past projects implemented things in really awesome ways; studying these saved us a lot of time when we wrote our own implementations. And of course, give credit where credit is due!

-Detailed and direct feedback, even if negative, is much more useful than brief and unspecific praise, which leaves you without a clear direction.



Acknowledgements

-Some previous languages that inspired us: Coral (2018), Shoo (2018), SickBeets (2017), Mathematica, C, MicroC

-Al Aho for winning the Turing Award :)