Google Summer of Code ’23 SymPy

Improving Parsing Submodule

Tirthankar Mazumder

Table of Contents

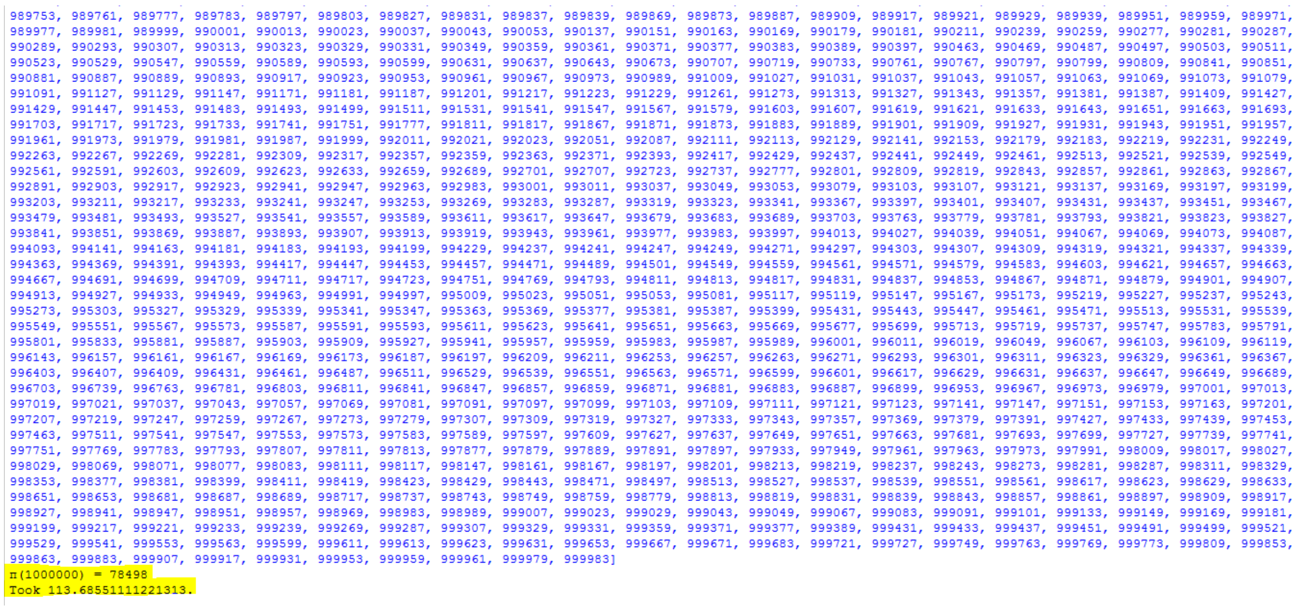
Personal Details

- Name: Tirthankar Mazumder
- Github Profile: [wermos]
- Email: mztirthankar@gmail.com
- University: Indian Institute of Technology Bombay
- Country of Residence: India
- Contact No: 9123802761
- Timezone: Indian Standard Time (UTC + 5:30)
- Primary Language: English

Programming Background

My first experience with programming was when I was in eighth grade: My father brought home a Programming in Python book from the library for me to read. Even though, at that age, I was unable to grasp the intricacies of core concepts like classes and strings, I got the hang of numerical programs, and made a calculator and a prime number generator.

I will never forget the joy I felt after I managed to make the program calculate how many primes there are between 1 and 1 000 000. The program ran for 113 seconds on the laptop we had at the time:



```
989753, 989761, 989777, 989783, 989797, 989803, 989827, 989831, 989837, 989839, 989869, 989873, 989887, 989909, 989917, 989921, 989929, 989939, 989951, 989959, 989971,
989977, 989981, 989999, 990001, 990013, 990023, 990037, 990043, 990053, 990137, 990151, 990163, 990169, 990179, 990181, 990211, 990239, 990259, 990277, 990281, 990287,
990289, 990293, 990307, 990313, 990323, 990329, 990331, 990349, 990359, 990361, 990371, 990377, 990383, 990389, 990397, 990463, 990469, 990487, 990497, 990503, 990511,
990523, 990529, 990547, 990559, 990589, 990593, 990599, 990631, 990637, 990643, 990673, 990707, 990719, 990733, 990761, 990767, 990797, 990799, 990809, 990841, 990851,
990881, 990887, 990889, 990893, 990917, 990923, 990953, 990961, 990967, 990973, 990989, 991009, 991027, 991031, 991037, 991043, 991057, 991063, 991069, 991073, 991079,
991091, 991127, 991129, 991147, 991171, 991181, 991187, 991201, 991217, 991223, 991229, 991261, 991273, 991313, 991327, 991343, 991357, 991381, 991387, 991409, 991427,
991429, 991447, 991453, 991483, 991493, 991499, 991511, 991531, 991541, 991547, 991567, 991579, 991603, 991607, 991619, 991621, 991633, 991643, 991651, 991663, 991693,
991703, 991717, 991723, 991733, 991741, 991751, 991777, 991811, 991817, 991867, 991871, 991873, 991883, 991889, 991901, 991909, 991927, 991931, 991943, 991951, 991957,
991961, 991973, 991979, 991981, 991987, 991999, 992011, 992021, 992023, 992051, 992087, 992111, 992113, 992129, 992141, 992153, 992179, 992183, 992219, 992231, 992249,
992263, 992267, 992269, 992281, 992309, 992317, 992357, 992359, 992363, 992371, 992393, 992417, 992429, 992437, 992441, 992449, 992461, 992513, 992521, 992539, 992549,
992561, 992591, 992603, 992609, 992623, 992633, 992659, 992689, 992701, 992707, 992723, 992737, 992777, 992801, 992809, 992819, 992843, 992857, 992861, 992863, 992867,
992891, 992903, 992917, 992923, 992941, 992947, 992963, 992983, 993001, 993011, 993037, 993049, 993053, 993079, 993103, 993107, 993121, 993137, 993169, 993197, 993199,
993203, 993211, 993217, 993233, 993241, 993247, 993253, 993269, 993283, 993287, 993319, 993323, 993341, 993367, 993397, 993401, 993407, 993431, 993437, 993451, 993467,
993479, 993481, 993493, 993527, 993541, 993557, 993589, 993611, 993617, 993647, 993679, 993683, 993689, 993703, 993763, 993779, 993781, 993793, 993821, 993823, 993827,
993841, 993851, 993869, 993887, 993893, 993907, 993913, 993919, 993943, 993961, 993977, 993983, 993997, 994013, 994027, 994039, 994051, 994067, 994069, 994073, 994087,
994093, 994141, 994163, 994181, 994183, 994193, 994199, 994229, 994237, 994241, 994247, 994249, 994271, 994297, 994303, 994307, 994309, 994319, 994321, 994337, 994339,
994363, 994369, 994391, 994393, 994417, 994447, 994453, 994457, 994471, 994489, 994501, 994549, 994559, 994561, 994571, 994579, 994583, 994603, 994621, 994657, 994663,
994667, 994691, 994699, 994709, 994711, 994717, 994723, 994751, 994769, 994793, 994811, 994813, 994817, 994831, 994837, 994853, 994867, 994871, 994879, 994901, 994907,
994913, 994927, 994933, 994949, 994963, 994991, 994997, 995009, 995023, 995051, 995053, 995081, 995117, 995119, 995147, 995167, 995173, 995219, 995227, 995237, 995243,
995273, 995303, 995327, 995329, 995339, 995341, 995347, 995363, 995369, 995377, 995381, 995387, 995399, 995431, 995443, 995447, 995461, 995471, 995513, 995531, 995539,
995549, 995551, 995567, 995573, 995587, 995591, 995593, 995611, 995623, 995641, 995651, 995663, 995669, 995677, 995699, 995713, 995719, 995737, 995747, 995783, 995791,
995801, 995833, 995881, 995887, 995903, 995909, 995927, 995941, 995957, 995959, 995983, 995987, 995989, 996001, 996011, 996019, 996049, 996067, 996103, 996109, 996119,
996143, 996157, 996161, 996167, 996169, 996173, 996187, 996197, 996209, 996211, 996253, 996257, 996263, 996271, 996293, 996301, 996311, 996323, 996329, 996361, 996367,
996403, 996407, 996409, 996431, 996461, 996487, 996511, 996529, 996539, 996551, 996563, 996571, 996599, 996601, 996617, 996629, 996631, 996637, 996647, 996649, 996689,
996703, 996739, 996763, 996781, 996803, 996811, 996841, 996847, 996857, 996859, 996871, 996881, 996883, 996887, 996899, 996953, 996967, 996973, 996979, 997001, 997013,
997019, 997021, 997037, 997043, 997057, 997069, 997081, 997091, 997097, 997099, 997103, 997109, 997111, 997121, 997123, 997141, 997147, 997151, 997153, 997163, 997201,
997207, 997219, 997247, 997259, 997267, 997273, 997279, 997307, 997309, 997319, 997327, 997333, 997343, 997357, 997369, 997379, 997391, 997427, 997433, 997439, 997453,
997463, 997511, 997541, 997547, 997553, 997573, 997583, 997589, 997597, 997609, 997627, 997637, 997649, 997651, 997663, 997681, 997693, 997699, 997727, 997739, 997741,
997751, 997769, 997783, 997793, 997807, 997811, 997813, 997877, 997879, 997889, 997891, 997897, 997933, 997949, 997961, 997963, 997973, 997991, 998009, 998017, 998027,
998029, 998069, 998071, 998077, 998083, 998111, 998117, 998147, 998161, 998167, 998197, 998201, 998213, 998219, 998237, 998243, 998273, 998281, 998287, 998311, 998329,
998353, 998377, 998381, 998399, 998411, 998419, 998423, 998429, 998443, 998471, 998497, 998513, 998527, 998537, 998539, 998551, 998561, 998617, 998623, 998629, 998633,
998651, 998653, 998681, 998687, 998689, 998717, 998737, 998743, 998749, 998759, 998779, 998813, 998819, 998831, 998839, 998843, 998857, 998861, 998897, 998909, 998917,
998927, 998941, 998947, 998951, 998957, 998969, 998983, 998989, 999007, 999023, 999029, 999043, 999049, 999067, 999083, 999091, 999101, 999133, 999149, 999169, 999181,
999199, 999217, 999221, 999233, 999239, 999269, 999287, 999307, 999329, 999331, 999359, 999371, 999377, 999389, 999431, 999433, 999437, 999451, 999491, 999499, 999521,
999529, 999541, 999553, 999563, 999599, 999611, 999613, 999623, 999631, 999653, 999667, 999671, 999683, 999721, 999727, 999749, 999763, 999769, 999773, 999809, 999853,
999863, 999883, 999907, 999917, 999931, 999953, 999959, 999961, 999979, 999983]
π(1000000) = 78498
Took 113.68551111221313.
```

Ever since, I have slowly been learning about more performant languages. In high school (11th and 12th grade), we learned Java, which is much faster than Python but still runs on a Virtual Machine. Afterwards, in college, I learned C++ through our introductory programming course, CS 101. C++ is even faster than Java and Python, and is compiled down to machine code.

After learning about C++ features, I did a project where we created a Physics Engine called Physicc. By doing the project, I got to learn many things about how projects are planned and executed, and also got to familiarize myself with Git and GitHub.

After this, I started watching CppCon videos, where I started learning about some of the more advanced C++ features like templates and compile-time code execution. During this time, I also learned about data-oriented design, which is a set of programming guidelines aimed towards getting the best performance out of the hardware that one has.

Technical Background

Platform and Development Environment

My OS of choice is Windows 11. However, I don't use Windows to develop in Python. I use WSL (Windows Subsystem for Linux), which is functionally equivalent to dual-booting. In particular, WSL2 uses a real Linux kernel under the hood. My editor of choice is usually Vim or Visual Studio Code. It highly depends on the exact task at hand. I usually use Vim for a more low-level look (at specific files), but use Visual Studio Code for a more bird's eye view of how multiple files and classes interact with one another. I find the "find definition" and "find declaration" features very useful because it allows me to see the definition/declaration without changing files.

Programming Experience

I have lots of previous experience with programming and contributing to open source projects in general. I have successfully completed GSoC once with the CERN-HSF organization. In that project, I had to do a survey of existing C++ math libraries and then integrate the fastest one into the algebra-plugins repository, which provides the math functionalities to a couple of their research repositories, detray and traccc. During that GSoC project, I made the following PRs:

- #69 **(Merged)** — Added Google Benchmark to the project.
- #77 **(Merged)** — Added Fastor to the project.
- #78 **(Merged)** — Fully integrated Fastor into the repository as a first-class backend ready to be used, and added tests.
- #84 **(Merged)** — Fixed an inconsistency in the CMake logic.
- #93 **(Merged)** — Improved `const`-correctness in the repository.
- #94 **(Merged)** — Modernized the tests by replacing casts to void with the `[[maybe_unused]]` attribute.

I also had to write blog posts about my work and my GSoC experience working with CERN-HSF. My CERN-HSF blog post on their website can be found here. The blog posts I made on my own personal blog are here, here, here, and here.

A comprehensive list of my other programming projects and experience can be found in the Past Projects section at the end.

Experience with Python

I have an extensive amount of experience with Python. Like I mentioned in the Programming Background section, my very first programming language was Python. Apart from that, I have used Python for many scripting tasks (like copying music from a hard drive onto my laptop in a specific order), Project Euler problems, and more. Moreover, I used it in my previous GSoC project to plot the results I got in my survey of math C++ libraries.

Overall, I would say that I'm quite familiar with Python, both in terms of syntax and its extensive standard library.

Favorite SymPy Feature

My favorite SymPy feature would have to be its printing feature. While it is a very popular feature to mention as one's favorite, the reason why it is my favorite is slightly different.

One of the first serious bugs I solved in SymPy was a curious case where the printing of a SymPy vector object was incorrect. It took me a lot of time and debugging to find the root cause of the issue. Part of the problem was that I was unfamiliar with how SymPy works under the hood.

While solving the bug, I was forced to learn how the printing code works under the hood. Once I got over the frustration of spending so long on one small bug, I was able to appreciate the cleverness of the method using for printing string representations of SymPy objects: the code essentially does a lot of configuration and then dispatches to a function specific to that object, which then does the actual printing. For example, if I want to print a `sympy.vector` object, the `_print_BasisDependent` function is ultimately called which deals with all the printing logic for `sympy.vector` objects, and `sympy.vector` objects only. Similarly, calling `print` on a `sympy.float` object invokes the `_print_Float` function under the hood.

Experience with Git

I am very familiar with Git. All of my projects (listed in the Past Projects section), along with my GSoC project, used Git, so I am quite familiar with its functionality, including things like force pushes, rebases, and merges, which tend to trip up beginners.

Motivation

As a math student who is also interested in Computer Science, symbolic math libraries were always amazing to me because they represented a perfect combination of both subjects.

There are many open source symbolic math libraries, such as Maxima, Fricas, and more. However, these CASes are in languages like Lisp, which make them less accessible to contributors such as myself.

SymPy is in a unique position because there are submodules where I can contribute and make the library more powerful, but that does not mean that the library itself is not powerful.

Contributing to the parsing submodule allows me to fulfill two objectives: exercise my programming muscles, but also help other people achieve their research and other goals faster. Since the LaTeX parser is the most used parser in SymPy, improving it would mean improving the user experience for a lot of people, which brings me satisfaction.

Contributions to SymPy

Here is a list of all my contributions to SymPy so far:

- #23096 **(Merged)** and #23122 **(Merged)** — Updated the SymPy AST parser to remove the use of a deprecated CPython class.
- #23191 **(Merged)** — Fixed a bug in the terminal pretty printing code.
- #23194 **(Merged)** — Improved the parenthesis code in the LaTeX printer.
- #23200 **(Merged)** — Modernized the symbol choice for the Laplacian in SymPy.
- #24954 **(Merged)** — Added the C parser code to the CI and made it pass some tests.

Overview of the Project

The parsing submodule in SymPy exists for the purpose of converting code in other languages (such as C, Fortran, LaTeX, etc.) into SymPy code. By far, the most popular parser in SymPy currently is the LaTeX parser.

However, the current implementation is written in Antlr, which is a pretty heavy dependency for a project like SymPy. Moreover, having a parser depend on something like Antlr prevents end users from modifying the parser at runtime, something which we can allow with an alternative implementation.

Additionally, there are a number of issues with the existing LaTeX parser. At the time of writing, there are 26 open issues under the `parsing.latex` in the SymPy issues list.

There has already been talk about moving away from Antlr in previous SymPy issues as early as `#14004`, and some work has already been done in that direction in `#19825`, which means that there is community interest in seeing this change through.

Implementation Details

The LaTeX parser will be rewritten in Lark, which is a parsing toolkit written entirely in Python. Many SymPy users face difficulty with installing Antlr and using it to work on the existing LaTeX parser and how it badly interacts with other popular Python libraries like `matplotlib` and `scipy`.

Since there is already an existing PR for this, much of my work will be in breaking down the PR into smaller chunks for easier code review, and also ironing out incompatibilities so that the new parser can be a drop-in replacement.

As a first approximation, the existing PR (`#19825`) can be broken into a base PR (which has a bare bones Lark LaTeX parser), then the next few PRs can have more and more features, and at the very end, we can remove the Antlr parser entirely and leave a SymPyDeprecationWarning for users who try to use it.

Timeline And Milestones

My summer vacation starts on May 1, 2022. Apart from GSoC, I am also taking part in the XROS Fellowship, whose coding period is from April to the end of June. Other than this, I have no other commitments, so I can easily put 20-30 hours per week during June, and up it to 40 hours per week once the XROS Fellowship period is over.

My third academic year starts from the first week of August, so to make sure that I can devote enough time to academics during the semester, I will try to finish as much of the project during summer break as I can. Hence, I intend to do the GSoC project during the period 29th May to 4th September, which is the standard 12-week coding period.

I also plan to write a blog post every week to track my GSoC progress, containing links to my contributions, a brief overview, and the plan for next week. This will ensure that these are not lost in the future. During my previous GSoC project, I put a lot of effort into making my own blog, so I already have a blog up, and adding content to it will be trivial.

- May 4 - May 28 (Community Bonding Period)
    - Get familiar with Lark
    - Discuss the LaTeX grammar and figure out exactly what subset we wish to parse
        - The grammar is not context-free, so we will need to figure out which set of words we accept and which ones we reject
- May 29 - June 11: Work on #19825
    - Understand the existing work done in that PR
    - Get familiar with the parsing submodule and the existing LaTeX parser code
    - Break down the PR into smaller, more review-friendly chunks (ideally 3 to 5 separate PRs)
- June 12 - June 26: Finish the work in #19825
    - Finish the PR and rewrite the parser in Lark

- June 27 - July 11: Get the Lark parser to pass all tests
    - Merge the new Lark parser and get it to pass all the existing tests in the `sympy/parsing/tests` directory
- July 4 - July 26: Solving existing issues about the lack of features in the LaTeX parser in the new parser
    - Solve issue #22305 and prevent the parsing from evaluating expressions
    - Ensure the fraction parsing logic is as simple as possible and fix #22392
    - Add tests for #22494 and ensure that the parenthesis parsing code is correct
    - Add support for both the text and mathrm forms of the differential operator, as #23551 suggests
    - Teach the parser about the dot notation for differentiation and fix #23617
- July 22 - August 7: Documentation of the new parser
    - Document the new parser fully, including all the design decisions that went into it
    - Also document quirks of the existing Antlr parser which were kept and/or fixed (example)
- August 8 - August 20: Removing the Antlr parser
    - Ensure that the Antlr parser throws a SymPyDeprecationWarning upon usage

    - Set up everything so that the Antlr parser can eventually be removed.

- August 21 - September 4: Buffer week
    - This time will be kept as a buffer to allow for unexpected setbacks
- Stretch Goals (If Time Permits)
    - Solve as many issues in the `parsing.latex` category as possible (example 1, example 2)
    - Solve some other parsing submodule bugs, like addressing the underlying issues in `#24813` (the C parser)

Past Projects

The following are a few of the things I did while working on a C++ game/physics engine called Physicc:

- I was put in charge of handling all the CI, which I did using GitHub Actions
- I wrote the bounding volume code, for which I used template metaprogramming to ensure a low runtime overhead
- I co-wrote the BVH and broadphase portions with one other person
- Helped write the narrowphase code with another person
- In addition to this, I also successfully used Doxygen and GitHub Actions to extract the documentation from our project and host it online using GitHub Pages

Apart from all this, I have also done a couple of self projects:

- I have written a ray tracer in C++20
    - I learned how to properly benchmark code using the `chrono` header
    - I learned how to profile code using the built-in profiler in Visual Studio 2022 and pinpoint the sections/functions where most of the running time is spent.
    - I learned how to use the Intel VTune Profiler to determine how well my multithreading is working, and how close to the theoretical peak performance I am.
    - I learned how to use the C++20 header `syncstream` to output progress to the terminal while the program runs.

Additionally, I also was a mentor for a ray-tracing engine called Rendera, in which I led and managed a group of 4 people of varying skill sets to create a C++17 CPU-side ray-tracer.

Contributions to Open Source

Here is a list of all my non-SymPy open source contributions so far:

- **ACTS**: Experiment-independent toolkit for (charged) particle track reconstruction in (high energy) physics experiments implemented in modern C++.
    - #1384 **(Merged)** — Cleaned up the CI workflow files.
- **xsimd**: C++ wrappers for SIMD intrinsics and parallelized, optimized mathematical functions (SSE, AVX, AVX512, NEON, SVE)).
    - #755 **(Merged)** — Fixed a rendering error in the docs.
    - #760 **(Merged)** — Fixed a bug in the preprocessor check for Clang.

    - #761 **(Merged)** — Incorporated the `[[nodiscard]]` attribute into the codebase.
    - #762 **(Merged)** — Fixed a spelling mistake.
    - #767 **(Merged)** — Modernized the codebase with `<type_traits>` functions.
    - #787 **(Merged)** — Changed many `if` clauses to `if constexpr`.
- **Fastor**: A lightweight high performance tensor algebra framework for modern C++.
    - #167 **(Merged)** — Fixed a bug in the determinant function.
- **SageMath**: A free open-source mathematics software system licensed under the GPL.
    - #35142 **(Merged)** — Fixed the SageMath logo in the README.
- Due to the extensive number of bug fixes and the work I have done on open source repositories, I am quite experienced in navigating and working with large codebases.