SymPy optimizations and demostrations for the generation of N body equations of motion

Juan Sebastián Barbosa

April 9, 2019

Abstract

Kane's and Lagrange's methods generate equations of motion, nevertheless there is space to make optimizations to reduce the time required to obtain them. To do so, trigonometric and physical simplifications must be studied, but speed can also be increased with the use of SymEngine and C/C++ code with wrappers for Python. Developing a set of interesting examples that not only obtain the equations of motion for many body systems, but also evolve the system in time, is also an objective of this project

1 The person

- **Full name:** Juan Sebastian Barbosa
- **University:** Universidad de los Andes, Colombia
- **Email:** juansebastian.coy@gmail.com
- **GitHub:** jsbarbosa
- Bicycle user :)

I am a last year physics and chemistry undergraduate student, with experience on C and Python, as open-source tools for the development of knowledge. Among my passions are: physical computing (code that physically interacts with surroundings e.g: microcontrollers, Raspberry), popular science in which I love to use computer simulations to show that there is plenty

of phenomena that can be studied within a computer, software development to simplify everyday tasks and physics itself.

2 The programmer

I have been programming for quite a while now. I first started with a Lego Mindstorm, then I moved to Microsoft Excel Macros (VB) at the end of my high school. Later, at college, I used Java during my first semester, and then I met Python throughout my second semester and it was love at first sight. At that very moment, I moved from Windows to Ubuntu and I have not stopped using it ever since; now it's been more than 4 years since then. Afterwards, during my 6th semester in college, I was introduced to C, and although it is not as easy to use or straight forward as Python, it's beauty is found in its power and speed. After that, I became Junior Teacher Assistant for the Computational Methods class for two years. Computational Methods is a course where we study numerical methods to solve ODEs, PDEs, integrals, derivatives and how to find solutions to equation systems. Most (if not all) of my coding is freely available in GitHub. Using this cool gadget I have found that I code in Python 5 times more than in C.

I think Python is the dream language, it is easy to use, and it is as readable as if it was pseudocode. Python is supported by a huge community -that I have found to be really friendly and ready to help-. Furthermore, it has literally hundred of thousands of modules and libraries that make it the perfect definition of a General Purpose Language. Python is as powerful for statistics as R, it only takes an `import pandas as pd`. With numerical calculations NumPy, and SciPy make it hard for MatLab to take any advantage. And off course symbolic calculations are not unique to Mathematica, thanks to the highly advanced **SymPy** module, from which many people around the world are thankful for. And best of all, Python does all of these for free, opening a universe of endless possibilities for which anyone, can code anywhere.

When it comes to scripting, my favorite IDE is Atom, as it is highly customizable, supports almost every programming language, and has a nice GUI (I have also used Spyder and Geany). However, for exploration and results presentation, Jupyter notebooks are the best.

I have used **SymPy** to find solution to equation systems, process mathematical expressions (e.g. integrate, gradients, simplify) and export results with `sympy.latex()`. **SymPy** has become one of my favorite tools, as it helped me on multiple classes, such as Solid State Physics, Quantum Mechanics and Classical Mechanics throughout my career. As a result of my Classical Mechanics and Computational Methods classes, I implemented a demonstration project, for which I used **SymPy** to obtain the equations of motion for a double pendulum with Lagrangian formalism, which were then solved numerically and resulted in an animation of the position in time, which in turn was generated using the Matplotlibs animation module. Each step to obtain the equations of movement was detailed and explained in the code, but mathematical expressions were generated inline by **SymPy**. Also, analogous demonstrations were made for single springs pendulums, projectiles (with force drag) and a model of the solar system (ten bodies, with information from Horizons). Besides, a $N \log(N)$ algorithm (Barnes and Hut) for gravitational systems with its visualization module was written by me on 2017 by the name of astrohut in C, with Python binders for the heavy duty functions. Lastly, for PDEs, a module called rippleTank was written in the same year in order to simplify the studies of wave interactions in ripple tanks.

3 You and your project

In physics the trajectories that a body follows are ruled by equations of motion. Depending on the system finding these equations can become very difficult. The Euler-Lagrange equation is useful to find the equation of motion associated to each of the system coordinates based on the potential and kinetic energy. Using energy has a great advantage over Newton's Second Law, as it requires scalars only. Even better is the Kane method, as it does not require to differentiate energy functions, which is handy (not to mention faster) for many body systems.

With this in mind, I wish to increase the speed at which the Kane's and Lagrange's methods generate equations of motion. To do so, trigonometric and physical simplifications must be studied, but speed can also be increased with the use of SymEngine and C/C++ code with wrappers for Python. These would be very helpful as many problems, depending on the coordinates -even for few bodies-, have a really long set of equations that can take a while

to generate. I would also like to develop a set of interesting examples that not only obtain the equations of motion for many body systems, but also evolve the system in time by numerically solving the differential equations. By doing this a cool set of demonstrations for the 'mechanics' submodule, can be used to accompany the documentation, to increase the understanding of the code, and to motivate the study of computational tools to easily represent the physical world. I also believe that a code is only as good as its documentation. The project might extend beyond the Google Summer of Code program, for which I have no problem to continue developing after this time e.g. to add new features. Right now, for the months spanning the GSoC, I have no other high priority duties, thus you can expect full time dedication to **SymPy**.

As an undergraduate student in physics, I had two classical mechanics related classes, where I learned the Lagrangian method (including Lagrange multipliers for non conservative forces). I also took 3 computational courses all of which use Python. Finally, I am working in my Thesis project were I simulate black hole's trajectories in non analytical, fully triaxial galactic potentials.

As far as the motivation goes, off course, the possibility of contributing to **SymPy**, being part of the selected students that get accepted to the Google Summer of Code, brings a great deal of happiness to me. As a physicist, any contribution I give to my field of study is something I find highly rewarding. But thinking of the possibility that the lines of code I write will be useful for someone, even if is only one person, regardless of his level of scholarity, is what excites me the most. As I said, I love popular science, and to me, coding is the best tool to encourage people to study sciences.

3.1 Community Bounding (May 6 - 27)

3.2 Week 1 - 3 (May 27 - June 14)

- Writing code optimizations for the Kane's and Lagrange methods in the `mechanics` submodule. To do so, both, mathematical and physical simplifications are expected.

- Improvements are to be quantified by benchmarking the new code with the old one.

3.3 Week 4 (June 17 - 21)

- Code documentation for the monthly submission. Pull request.

3.4 Week 5 - 9 (June 24 - July 26)

- Phase 1 evaluations (June 24 - 28).
- Speed improvements are to be expected with the use of C/C++ code. SymEngine opens the possibility to decrease computational times by writing the same Python algorithms in C/C++ with tiny wrappers for Python.
- Code documentation for the monthly submission. Pull request.
- Phase 2 evaluations (July 22 - 26).

3.5 Week 10 - 12 (July 29 - August 16)

- Examples and demonstrations development, numerically solving a set of generated equations.

3.6 Week 13 (August 19 - 26)

- Code documentation for the **final submission**. Pull request.

4 Others

I would like to make at least 3 pull requests, one at the end of each coding month.

4.1 Patch requirement

I have uploaded a pull request to fix a symbol related issue on the `geometry` submodule `Parabola` #14461.

5 References for the project

- Classical Mechanics - John R. Taylor
- Computational Physics: Problem Solving with Python - Rubin H. Landau, Cristian C. Bordeianu, Manuel J Pez