GSoC'25 Proposal for SciML - NumFOCUS

Building algorithms for symbolically solving ODEs

Contents

Note : I've used LLM in a few places in this document to correct my grammar so that I can convey more accurately.

Project Details:

- Organization : SciML - NumFOCUS
- Project Title : Building algorithms for symbolically solving ODEs
- Mentors : Dr. Chris Rackauckas
- Project Length : 350 hours
- Difficulty : Hard

Personal Information:

- Name : Karthik Rishinarada
- Email : karthikrk11135@gmail.com
- LinkedIn: Karthik Rishinarada
- Github ID : karthik11135

- Brief Summary : I'm a senior-year college student from NIT Trichy, India. I interned at Capital One in 2024. Over the past few months, I have developed an interest in open source and started exploring open-source codebases. I enjoy listening to music, window shopping, and planning vacations.
- Resume : my resume

Julia:

Julia is a programming language that is intuitive to understand and that is as fast as Fortran or C. It is used for scientific computing and numerical analysis. It was developed to solve problems that python and R could not. There are various packages in Julia that facilitate a wide range of use cases.

About the Organisation:

SciML - SciML (Scientific Machine Learning) is an open-source non profit organization that supports scientific computing projects. It brings together different packages to create a unified and efficient ecosystem for solving complex computational problems.

The following are the various packages that are required for the current project:

1. DifferentialEquations.jl – A library designed to solve a wide range of differential equations, including ordinary differential equations (ODEs), stochastic differential equations (SDEs), delay differential equations (DDEs), differential-algebraic equations (DAEs), and hybrid systems. It supports complex, multi-scale models and can handle agent-based simulations as part of the equation solving process.

2. Symbolics.jl – A high-performance symbolic algebra system built for Julia, designed to perform fast symbolic computations. Its goal is to offer efficient algebraic operations that can be extended directly within the Julia programming environment. It's a very useful tool for people whose work involves symbolic computation.

3. ModelingToolkit.jl – A symbolic modeling framework that simplifies the creation and simulation of complex mathematical models. This package not only allows users to generate code automatically but also simplifies the process of solving and optimizing equations.

Some of the other packages under the umbrella of SciML are NeuralPDE.jl, DiffEqFlux.jl, Catalyst.jl, ParameterizedFunctions.jl etc.

Project Summary:

Ordinary Differential Equations (ODEs) are crucial in Scientific Machine Learning (SciML) because they allow models to encode and learn physical laws, enabling them to make accurate predictions with less data, especially in domains like climate modeling where data is often sparse or difficult to obtain.

This project seeks to create a range of algorithms for symbolically solving ordinary differential equations (ODEs). Two main approaches being explored are differential equation transformations and the use of the Buckingham Pi Theorem. The Buckingham Pi Theorem, based on dimensional analysis, enables one to simplify complex physical systems, typically with multiple variables like length, time, velocity, and mass into fewer dimensionless parameters. By projecting the ODE into these dimensionless combinations, the equation will be with fewer dimensions, exposing hidden structures or symmetries that are not evident in the dimensional version of the ODE. This procedure can ease the symbolic solution difficulty, acting as a valuable pre-processing step prior to using more conventional solution methods.

Tasks :

1. Differential equations function transformations : The goal of this task is to convert a given ODEsystem and transform the variables of the system to a new set of variables.

The PR for this task can be found here : #3391

The solution is a function transform_eqs that takes an ODESystem, variable transformations and their inverses and returns the new set of differential equations and the updated list of variables after substitution and simplification. This is useful for redefining an ODE system in terms of new dependent variables while preserving the dynamics.

For example :

```
@parameters t a
@variables x(t) z(t)
eqs = [D(x) ~ a*x]
sys = ODESystem(eqs, t, [x], [a])
transformations = [z => exp(x)]
back_transformations = [x => log(z)]

new_eqs = transform_eqs(sys, transformations, back_transformations)
# new_eqs: [D(z) ~ a*z*log(z)], new_vars: [z]
```

Procedure :

1. Retrieve the dependent variables and the equations from the ODESystem.
2. Collect the new variables and the mutable variables in a list.
3. Differentiate the LHS and RHS parts of the transformation equations
   From the above example :
   transformations = [z ~ e^x]. Then,
   dz = D(z); dzdt = e^x * D(x)
4. Substitute the values of old variables from the inverse transformations and the values of differential terms from the original equations.
   a. dzdt = e^x * D(x). Value of x is log(z) from the back transformations and the value of D(x) is a*x
5. The final equation of D(z) would be equal to a*z*log(z)
6. Substitute the values of default values if there are any.
7. Form a new ODESystem using the new equations and the new variables.

2. <u>Buckingham PI transformations :</u>

The Buckingham Pi Theorem states that a physical equation involving n variables, expressible in k independent fundamental dimensions, can be

transformed into an equivalent equation with n-k dimensionless parameters (Pi terms).

The PR for this task can be found here : #3443

The solution uses the DynamicQuantities.jl package and the ModelingToolkit.jl package. The function buckinghumFun is responsible for taking in the variables and providing a list of PI terms. And the function transform_eqs is responsible to take the PI terms and the equations and provide a new ODESystem.

Example :

```
@variables f, d, v, ρ, μ
vars1_arr = [ρ, d, v, μ, f]

vars1_quants = [DynamicQuantities.Quantity(0, mass=1, length=-1, time=-1),
DynamicQuantities.Quantity(0, length=1) , DynamicQuantities.Quantity(0,
length=1, time=-1),  DynamicQuantities.Quantity(0, mass=1, length=1, time=-2),
DynamicQuantities.Quantity(0, mass=1,  length=-1, time=-1)]

buckinghumFun(vars1_quants, vars1_arr)[1] would be equal to μ/(d*v*ρ)

pi_terms = buckinghumFun(vars1_quants, vars1_arr)
new_sys = transform_sys(pi_terms, old_sys, [π1])
```

The function buckinghumFun takes two arguments. One is the list of variable names and the other is the DynamicQuantities of them. The procedure to find the PI terms and the new system are as follows :

1. List down all the variables and take the count (n)
2. Count the number of fundamental dimensions (k)
3. Number of PI terms would be n - k
4. Inorder to find the PI terms follow the procedure in the following example
   Variables = A, B, C, D; Number of fundamental dimensions = 3
   Number of PI terms = 1
   Then, π = A * B^a * C^b * D^c (B, C, D are considered as repeating variables)
5. Dimensions are substituted in the variables and the exponents are solved to find the values of a, b and c.
6. Finally the PI terms are used to form the new ODESystem.

Milestones and Deliverables:

Week	Focus Area	Deliverables
Week 1 - 3	• Communicate my ideas with the mentor and understand the requirements. • Continue to work on the transformation algorithms. • Provide examples and write unit tests.	• Working Code for implementations of Buckingham PI and differential equation transformations.

Week 3-7	• Docstrings for all major functions
	• Jupyter notebooks or Julia scripts showing:
	  ○ How to transform an ODE system using transform_eqs
	  ○ How to perform dimensional analysis using buckinghumFun
	• Handle special cases: nonlinear transformations, singularities, and non-invertible substitutions
	• Regularly ask for feedback and improve accordingly
	• Work on my midterm review/presentation.
		• Mid term presentation
		• Test suite for the code written.
		• Documentation for the code written.

Week 7-10	• Optimize the symbolic transformations for speed and memory usage, especially for large systems.	• Deployable code that is properly tested and documented.

	• Register transformations as ModelingToolkit extensions or plugins (if feasible). • Improve error messages and validation for user inputs. • Work on my final review presentation • Ask for the overall feedback and improve accordingly.	• Final review presentaiton.

Why did I choose SciML - NumFocus?

Over the past few months I wanted to explore open-source contributions. So I started looking for repositories on the GSoC website. That's when I came across SciML.

I'm really fascinated by the Julia programming language. It's intuitive and really fast. I wanted to take a break from the traditional programming languages like Python and Javascript and delve into something new and exciting. I reached out to Dr. Chris Rackauckas who guided me and gave me tasks to work on. I was really interested in the problem statements and have enjoyed the process of writing Julia code to solve the issues. Also, one thing that I really admire is that the documentation was written in a very clear manner. The clarity of explanations in the documentation made it easier for me to understand Julia.

Beyond the technical aspects, I also appreciate SciML's active and welcoming community. These are the reasons why I want to contribute to SciML.

Why am I the best fit?

My first internship offer was from a fintech company — Capital One, where I worked as a Software Developer Intern for two months. During this time, I improved the runtime of the codebase by 40%. After my internship, I discovered SciML and began exploring by studying the documentation and working through the tutorials.

I strongly believe that I am a good fit for this project because I have worked on the tasks that are assigned to me. I've always responded quickly whenever I'm asked to do some changes in my open PRs. I am confident

that I can hit the ground running from day zero. Apart from that I'm flexible to work according to any timezone.

I view GSOC as a great opportunity to learn from experienced mentors and contribute to a real world project. However my motivation extends beyond GSOC itself. I plan to continue contributing to SciML and staying involved even if my application isn't selected, because my primary focus is on learning and making good contributions.

Conclusion

In conclusion, my deep interest in Julia and experience with open-source contributions, makes me a great fit for this project. While I have my final exams from May 6 to May 13, I will be entirely free afterward and fully dedicated to contributing to this project. I look forward to contributing my best to the SciML - NumFocus.

Thank you
