Classical Mechanics: Implement Wrapping Geometry and Pathways for Musculoskeletal Modeling

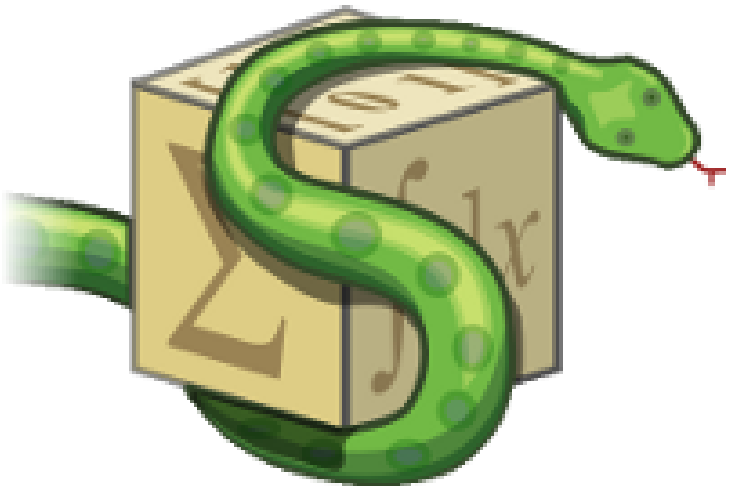

Table of Contents

Me the person

Personal Details

- Full name: Rushabh Mehta
- Primary Email: mehtarushabh2005@gmail.com
- College Email: rbmehta_b23@ce.vjti.ac.in
- Github: RushabhMehta2005
- LinkedIn: mehtarushabh2005
- University: Veermata Jijabai Technological Institute (VJTI), Mumbai
- Degree: Computer Engineering
- Minor: Signal and Image Processing
- Country: India
- Location: Mumbai
- Timezone: Indian Standard Time (IST) (UTC + 5:30)
- Language of Communication: English

My Introduction

Hello, myself Rushabh Mehta, a second year computer engineering student at Veermata Jijabai Technological Institute (VJTI), Mumbai. Regarded as one of the finest engineering institutions in the country, to get enrolled in its programs, you must qualify entrance exams like JEE and MHT-CET, in which millions of students appear and are tested heavily on math, physics and chemistry.

In my spare time I enjoy reading non-fiction stories, solving puzzles of any kind and keeping up to speed with the latest advancements in STEM.

During my 2 years in university, I have participated in various coding competitions and organised various events, mentored juniors in their journey and helped my peers grow along with me. I have good communication and written communication skills. I speak English fluently.

Me the Programmer

Math and CS background

Mathematics has always been my forte, and it forms the basis for the study of core computer science subjects like computational theory, algorithms and much more.

My first programming language was C. I appreciated the low level control over the CPU that the language gave but I preferred faster development and dynamic typing more, thus I learnt Python and have been programming in it for 2.5+ years in it at the moment of writing.

I use Visual Studio Code as my IDE. It has all the features I need built into it making my experience seamless, and Windows 11 as my operating system which serves my needs. I have experience with Git and Github as I use both frequently for my contributions to Sympy and for my personal projects.

A small project that I wish to share here is my (in-progress) attempt at creating a python package for the subject of Theory of Computation. What attracted me towards the idea was the simplistic nature of the models of computation and the algorithmic aspects of the course. It felt intuitive and fun to implement automata in an object oriented fashion. Link here.

Coming back to why I am suited for this project, I have a strong background in physics due to my JEE preparation. Newton's Laws of Motion, Forces, Rotation and Torques were parts of the core curriculum and I am proficient in these concepts. I am also well versed in geometry, paths and curves from my engineering math courses, the concepts of which will be useful to us.

The relevant college courses I have taken are:

- **Engineering Mathematics 1 and 2**
- **Engineering Mechanics**
- Discrete Mathematics
- Data Structures
- Design and Analysis of Algorithms
- Computer Programming and Problem Solving
- **Python Programming Lab**

Now let's talk about Sympy, a tool which I liked after mere minutes of discovering it!

Sympy combines my love for math and computer science and turns it into a Python package. My favourite feature by far is the "integrate()" function. The integration operation is the most daunting topic in math. Whether you're good at it or not is a matter of practicing problems and developing an intuition for its methods. But Sympy distills that down into a single function using powerful and carefully implemented algorithms.

Example problem from JEE exam math section:

$$\int \frac{x^2 - 1 + \tan^{-1}\left(\frac{x^2+1}{x}\right)}{(x^4 + 3x^2 + 1)\tan^{-1}\left(\frac{x^2+1}{x}\right)}\, dx$$

Figure 1: (Try doing this by hand)

But in Sympy,

```
[1]:  %pip install sympy==1.13.3

[2]:  from sympy import *
      init_printing()
      x, y, z, t = symbols('x y z t')
      k, m, n = symbols('k m n', integer=True)
      f, g, h = symbols('f g h', cls=Function)

[3]:  integrate((x ** 2 - 1 + atan((x ** 2 + 1) / x) / (atan((x ** 2 + 1) / x) * (x ** 4 + 3
      * x ** 2 + 1))))
```

[3]:
$$\frac{x^3}{3} - x + 2\left(\frac{\sqrt{5}}{20} + \frac{1}{4}\right)\operatorname{atan}\left(\frac{2x}{-1+\sqrt{5}}\right) - 2\left(\frac{1}{4} - \frac{\sqrt{5}}{20}\right)\operatorname{atan}\left(\frac{2x}{1+\sqrt{5}}\right)$$

Figure 2: The integrate API

It is merely a matter of writing out your integrand function now. This shows the power of symbolic computation and the advantage of using a library like Sympy.

Why I chose to go for Sympy

As expressed already, I am fascinated by Sympy and wish to contribute to it, and improve it for its users. It is a well maintained and respected open source project. It will give me real world software development and engineering experience. The community was extremely welcoming for a new contributor like me, which drew me more towards it. Sympy is widely used by researchers, students and educators all around the globe.

My Contributions to Sympy

I have been contributing to Sympy for about 4-5 months now. I have learnt the open source workflow for Sympy and have gained a general understanding of the codebase.

- (**Merged**) #27228 Solved the issue where the `wigner_3j` function in `sympy.physics` was violating behavioral subtyping with numpy floats.
- (**Merged**) #27288 Solved a similar issue which arose in both the `wigner_6j` and `wigner_9j` functions.
- (**Merged**) #27347 Implemented the `expectation` method of the `PoissonDistribution` class to quickly return the correct expected value of the distribution, thus solving the issue.
- (**Merged**) #27414 Added tests for the `Quaternion.integrate()` method, removed the TODO.
- (**Merged**) #27789 Fixed an incorrect type annotation in `sympy.mechanics` docs.
- (**Closed**) #27286 Implemented `is_kth_power_free` and `kth_power_free_range` functions which utilized `factorint` function to check and obtain k-th power free numbers easily.
- (**Closed**) #27574 Tried to extend the double factorial function for complex arguments.
- (**Open**) #27384 Added support for symbolic multiplication of scalar multiplication of waves.
- (**Open**) #27831 Added a neccessary check in `geodesic_end_vectors` method in `sympy.mechanics`.
- Raised issue #27610 regarding the `ask` function returning unexpected output on the input `I*oo`. It was then resolved.

Me and my project

Now I will describe the project. **This is intended to be a 175 hour project.**

Overview

A wrapping geometry specifies the path along which a force acts. For example, if the wrapping geometry is a sphere, it can model the behavior of a force that follows a spherical path between two points. This is particularly useful in biomechanics, where muscles and tendons often wrap around bones or joints. The geometric representation of these paths directly influences the magnitude and direction of the force, making accurate modeling essential for realistic simulations of musculoskeletal interactions.

According to the principle of least action, when a force travels between two points on a surface, it always follows the **shortest possible path**. These shortest paths are known as **geodesics**. For simple geometries such as spheres or cylinders, closed-form solutions for geodesics exist. However, for more complex surfaces like ellipsoids, cones, and toroids, closed form solutions are either challenging or not available at all. This represents the major problem I will try to solve with this project.

One of the most popular tools currently in use for biomechanical modeling is OpenSim, which is a C++ library. It relies on numerical methods such as discretizing complex geometries and computing force pathways by means of numerical integration. We, on the other hand, in Sympy will aim to solve these problems symbolically as much as possible. An approximate but symbolic solution can offer analytical insights as well; this can be immensely useful for biomechanics researchers. This project will add new wrapping geometry objects to Sympy, enabling users to model more involved musculoskeletal interactions for biomechanics research and educational/teaching purposes. I have reviewed contact geometry objects available in OpenSim and what Sympy currently lacks.

I will implement 3 new wrapping geometries in Sympy, namely the **Spheroid**, **Cone** and **Torus**.

Project Phases

Phase 1 – Conical Wrapping Geometry (WrappingCone)

Mathematical Foundations

A right circular cone is a **developable surface** – it can be flattened onto a plane without distorting distances. This property allows us to derive closed–form expressions for geodesics on the cone by "unfolding" it into a circular sector. Specifically, a point on the cone is described by two coordinates:

- the *slant distance* r from the apex, and
- the angular coordinate u (measuring rotation about the cone's axis).

When unfolded, the cone maps isometrically onto a circular sector with:

$$\text{Radius} = r, \quad \text{Central angle } \gamma = 2\pi \sin\theta,$$

where θ is the cone's half-angle. In the unfolded (plane) setting, we introduce polar coordinates (r, ϕ) with the relation:

$$\phi = u \sin\theta.$$

Given two points P_1 and P_2 on the cone with coordinates (r_1, u_1) and (r_2, u_2), their unfolded coordinates become (r_1, ϕ_1) and (r_2, ϕ_2), where $\phi_i = u_i \sin\theta$. In the plane the geodesic is the straight line (chord) connecting these points. By the law of cosines, the geodesic length L is

$$L = \sqrt{r_1^2 + r_2^2 - 2\, r_1\, r_2 \cos\bigl(\Delta\phi\bigr)},$$

with

$$\Delta\phi = \phi_2 - \phi_1.$$

For the geodesic end vectors, the procedure is as follows:

1. **Unfolded Unit Tangent Calculation:** In the plane, compute the difference vector

   $$\Delta\mathbf{v} = \bigl(r_2 - r_1,\ \phi_2 - \phi_1\bigr)$$

   and normalize it to obtain the unit tangent vector $\mathbf{t}_{\text{plane}} = (t_r,\, t_\phi)$.

2. **Refolding via the Differential:** The unfolding map

   $$F : (r, u) \mapsto (r,\, u \sin\theta)$$

   has differential

   $$dF = \begin{pmatrix} 1 & 0 \\ 0 & \sin\theta \end{pmatrix}.$$

   Its inverse,

   $$dF^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{\sin\theta} \end{pmatrix},$$

   converts the angular component back so that the tangent components in the cone's coordinates become

   $$\bigl(t_r,\, t_u\bigr) \quad \text{with} \quad t_u = \frac{t_\phi}{\sin\theta}.$$

3. **Push–Forward onto the Embedded Cone:** The cone is embedded in $\mathbb{R}^3$ by the mapping
$$f(r,u) = \Big(r\sin\theta\cos u,\ r\sin\theta\sin u,\ r\cos\theta\Big).$$
Its partial derivatives are
$$\frac{\partial f}{\partial r} = \Big(\sin\theta\cos u,\ \sin\theta\sin u,\ \cos\theta\Big)$$
and
$$\frac{\partial f}{\partial u} = \Big(-r\sin\theta\sin u,\ r\sin\theta\cos u,\ 0\Big).$$
At each endpoint, the tangent vector is assembled as
$$\mathbf{T} = t_r\,\frac{\partial f}{\partial r} + t_u\,\frac{\partial f}{\partial u},$$
and then normalized.

Implementation Strategy

- **Geodesic Length:** Compute the geodesic length between two points $p1 = (r_1, u_1)$ and $p2 = (r_2, u_2)$ on the cone by:
  1. Converting the angular coordinates with $\phi_i = u_i \sin\theta$.
  2. Calculating the angular difference $\Delta\phi = \phi_2 - \phi_1$.
  3. Evaluating the length via
$$L = \sqrt{r_1^2 + r_2^2 - 2\,r_1\,r_2\cos\Big(\Delta\phi\Big)}.$$
- **Tangent Vector Computation:** Determine the geodesic's end tangent vectors as follows:
  1. Unfold the points $p1$ and $p2$ to the plane and compute the unit tangent vector along the chord, $\mathbf{t}_{\text{plane}} = (t_r, t_\phi)$.
  2. Map the tangent component back to the cone's (r, u) coordinates using
$$t_u = \frac{t_\phi}{\sin\theta}.$$
  3. Embed the tangent vector on the cone by computing the partial derivatives from
$$f(r,u) = \Big(r\sin\theta\cos u,\ r\sin\theta\sin u,\ r\cos\theta\Big),$$
then forming
$$\mathbf{T} = t_r\,\frac{\partial f}{\partial r} + t_u\,\frac{\partial f}{\partial u},$$
and normalizing $\mathbf{T}$ to obtain the unit tangent.

Phase 2 – Spheroidal Wrapping Geometry (WrappingEllipsoid)

Mathematical Foundations

An ellipsoid of revolution (or spheroid) is given by

$$\frac{x^2 + y^2}{a^2} + \frac{z^2}{b^2} = 1,$$

where a is the equatorial radius and b is the polar radius. Two key derived parameters are the *flattening*

$$f = \frac{a - b}{a},$$

and the *eccentricity*

$$e = \sqrt{1 - \left(\frac{b}{a}\right)^2}.$$

Since there is no simple closed-form for the geodesics on a spheroid, a common strategy is to map the geodetic coordinates to the *auxiliary sphere*. This is done as follows:

1. For each point with geodetic latitude ϕ, compute the **reduced latitude**

$$\beta = \arctan\Big((1 - f)\tan\phi\Big).$$

2. For two points with reduced latitudes β_1 and β_2 and longitudes λ_1 and λ_2, the spherical law of cosines on the auxiliary sphere gives the spherical arc length σ:

$$\cos\sigma = \sin\beta_1 \sin\beta_2 + \cos\beta_1 \cos\beta_2 \cos(\Delta\lambda),$$

   where $\Delta\lambda = \lambda_2 - \lambda_1$.

3. The geodesic distance s on the spheroid is then expressed as a series in the flattening f:

$$s \approx a\Big[A\,\sigma - B\,\sin(2\sigma) + C\,\sin(4\sigma) - \cdots\Big],$$

   with coefficients A, B, C, ... easily determined via a series expansion in f.

For the tangent (or endpoint) vectors, the local directions at each point are computed from the ellipsoidal coordinate transformation. Specifically, the local north and east unit vectors, $\mathbf{N}$ and $\mathbf{E}$, are derived by differentiating the coordinate mapping. Once the azimuth α (the angle between the geodesic and the north direction) is determined, the endpoint tangent vector is given by

$$\mathbf{T} = \cos\alpha\,\mathbf{N} + \sin\alpha\,\mathbf{E}.$$

Implementation Strategy

- **Class Structure:** Develop a class `WrappingEllipsoid` as a subclass of `WrappingGeometryBase` that stores the ellipsoidal parameters a and b (or equivalently a and f).
- **Series Expansion for Geodesic Length:** Implement helper functions using Sympy's symbolic tools to:

1. Compute the reduced latitude β for any given geodetic latitude ϕ.
2. Calculate the spherical arc length σ on the auxiliary sphere via the spherical law of cosines.
3. Expand the geodesic distance s as a series in f, i.e.,

$$s \approx a\,[A\,\sigma - B\,\sin(2\sigma) + C\,\sin(4\sigma) - \cdots],$$

where the coefficients A, B, C, ... are determined symbolically.

- **Computation of Tangent Vectors:** Derive the local north (**N**) and east (**E**) vectors by differentiating the ellipsoidal coordinate transformation. Then:
  1. Compute the azimuth α from the geodetic data.
  2. Form the endpoint tangent vector using the relation

$$\mathbf{T} = \cos\alpha\,\mathbf{N} + \sin\alpha\,\mathbf{E}.$$

Phase 3 – Toroidal Wrapping Geometry (WrappingToroid)

Mathematical Foundations

A torus is parametrized by

$$\begin{aligned} x(u,v) &= \Big(R + r\cos v\Big)\cos u, \\ y(u,v) &= \Big(R + r\cos v\Big)\sin u, \\ z(u,v) &= r\sin v, \end{aligned}$$

with $u \in [0, 2\pi)$ and $v \in [0, 2\pi)$. Here, R is the major radius (the distance from the center of the tube to the torus center), and r is the minor radius (the tube radius).

The first fundamental form, which encodes the metric on the torus, is given by

$$ds^2 = \Big(R + r\cos v\Big)^2\,du^2 + r^2\,dv^2.$$

Although a torus is not developable over its entire surface, for a sufficiently small patch (e.g., where the muscle wraps) it is acceptable to locally "unfold" the torus into a planar domain. In this local domain, the geodesic is approximated by a straight line and its length is computed by the Euclidean distance. This computed length is then appropriately scaled to account for the toroidal curvature when mapped back to the surface.

For the tangent (or endpoint) vectors, we derive a local basis from the partial derivatives of the torus parametrization. Analogous to the north–east basis in the ellipsoidal case, these local tangent directions allow us to combine with the computed azimuth to yield the final endpoint unit vectors:

$$\mathbf{T} = \cos\alpha\,\mathbf{N} + \sin\alpha\,\mathbf{E},$$

where α is the azimuth of the geodesic at the endpoint.

Implementation Strategy

- **Class Structure:** Develop a class `WrappingToroid` that extends `WrappingGeometryBase` and stores the torus parameters R and r.
- **Point Verification:** Implement a method `point_on_surface` that checks if a given point (x, y, z) satisfies the torus equation. For example, by verifying that

$$\left(\frac{\sqrt{x^2+y^2}-R}{r}\right)^2+\left(\frac{z}{r}\right)^2\approx 1.$$

- **Geodesic Length Calculation:** Develop `geodesic_length` as follows:
  1. *Local Unfolding:* Map the two points from their toroidal coordinates (u, v) to a planar domain using the standard mapping. For instance, the u-coordinate is scaled by the local factor $R + r\cos v$ while the v-coordinate is retained.
  2. *Chord Computation:* Compute the Euclidean chord length in the plane:

$$L_{\text{planar}}=\sqrt{\Delta u^2\,(R+r\cos v)^2+\Delta v^2\,(r)^2},$$

     where Δu and Δv are the differences in the toroidal coordinates (with an appropriate handling of periodicity).
  3. *Scaling Adjustment:* Adjust L_{planar} using scaling factors derived from the first fundamental form to obtain the geodesic length s on the torus.
- **Endpoint Tangent Vectors:** For `geodesic_end_vectors`:
  1. Compute the partial derivatives of the torus parametrization:

$$\frac{\partial f}{\partial u}=\Big(-\big(R+r\cos v\big)\sin u,\ \big(R+r\cos v\big)\cos u,\ 0\Big),$$

$$\frac{\partial f}{\partial v}=\Big(-r\sin v\cos u,\ -r\sin v\sin u,\ r\cos v\Big).$$

  2. Define the local basis vectors analogous to "north" and "east" using these derivatives.
  3. Determine the geodesic's azimuth α from the unfolded planar mapping.
  4. Form the endpoint tangent vector as

$$\mathbf{T}=\cos\alpha\,\mathbf{N}+\sin\alpha\,\mathbf{E},$$

     and normalize $\mathbf{T}$ to obtain a unit tangent.

Project Timeline

Community Bonding Period (May 8 to June 1)

- Set up my GitHub blog to share my progress publicly.
- Get to know my mentors, establish communication channels, and finalize a regular meeting schedule with them.
- Network with fellow GSoC contributors and review their proposals to better understand diverse projects this year.
- Familiarize myself with the codebase and the overall GSoC ecosystem.
- Engage in preliminary discussions regarding the project scope and initial ideas with my mentors.

Weeks 1, 2, and 3 (June 2 to June 23)

- Focus on Phase 1: `WrappingCone` (Conical Wrapping Geometry).
- Implement closed-form expressions for the geodesic length and endpoint tangent directions for the cone.
- Implement the `WrappingCone` class with methods `point_on_surface`, `geodesic_length`, and `geodesic_end_vectors`.
- Develop unit tests and detailed documentation for `WrappingCone` to validate against known cases.

Weeks 4, 5, and 6 (June 24 to July 14)

- If there is any remaining work from Phase 1, finish it in week 4.
- Transition to Phase 2: `WrappingEllipsoid` (Spheroidal Wrapping Geometry).
- Implement the mathematical derivations for the reduced latitude, spherical arc length σ, and series expansion for the geodesic length s as a function of f.
- Develop local helper functions and integrate them into the `WrappingEllipsoid` class.
- Begin writing initial tests and comprehensive documentation for `WrappingEllipsoid`, ensuring consistency with the overall `WrappingPathway` interface.

Weeks 7, 8, and 9 (July 15 to August 4)

- Finalize tests and documentation for Phase 2.
- Begin Phase 3: `WrappingToroid` (Toroidal Wrapping Geometry).
- Develop methods to compute local basis vectors from the torus parametrization and derive the endpoint tangent vectors.
- Initiate preliminary testing and draft documentation for `WrappingToroid`.

Weeks 10, 11, 12, and 13 (August 5 to September 1)

- Finalize testing and documentation for `WrappingToroid`.
- Integrate and refine all modules, clean up the code, and merge any open pull requests.
- Polish overall documentation to ensure clarity and consistency across all new classes, in compliance with the `WrappingPathway` interface.
- Submit final evaluations and project deliverables.

My Commitment to this project

- I have end semester examinations till 15th May, after which I will begin my work discussed in the community bonding phase.
- My 5th semester starts near July 14th, but there is a significantly lesser amount of work in the first couple of months of the semester, thus this will not create any hindrance.
- I have no other work or internships during this summer; therefore, I will dedicate my entire time during the day to GSoC.
- I will communicate regularly with my mentors, updating them about my progress.
- I will strive to work as independently as possible, adhering to the proposed timeline. If any unforeseen blockers arise, I will communicate them transparently to my mentors.
- If there are any blockers or unforeseen situations arising, I will communicate them to my mentors with full transparency.
- I am committed to owning the work completed during GSoC, addressing any bugs or issues that arise post-project, and continuing to improve the implementation based on user feedback and emerging needs.

References

- OpenSim Documentation.
- Vincenty, T. (1975). *Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations.* Survey Review.
- Karney, C. F. F. (2013). *Algorithms for geodesics.* Journal of Geodesy.
- Danielsen, J. (1989). *The area under the geodesic.* Survey Review.
- Standard texts on differential geometry and geodesy, including works by Bessel and Helmert.
- Sympy Wiki and Documentation.
- GSoC 2024 proposals.
- GSoC 2025 Timeline.