Phys 607 - Computational Physics, Comparison of Numerical Integration Methods

Haoyang Zhou

September 20 2024

Instructed by Alex Nitz

1 Application Topics

In this project, there are two physical problem chosen for comparisons using python modules that calculate the integral of that given problem. This section briefly explain the problems.

1.1 Problem 1 (ODE): Radioactive Compound-Decay

We know that radioactive decay of substance A-B follows the simple ODE:

$$\frac{dN_A}{dt} = -\lambda_A N_A(t)$$

Solving for analytical solution:

$$N_A(t) = N_A(0)e^{-\lambda_A t}$$

Then, substance A can decay into substance B, which can also decay into substance C. For B, the ODE becomes:

$$\frac{dN_B}{dt} = \lambda_A N_A(t) - \lambda_B N_B(t)$$

Solving for it yields:

$$N_B(t) = \frac{\lambda_1}{\lambda_2 - \lambda_1} N_1(0)(e^{-\lambda_1 t} - e^{\lambda_2 t}) + \lambda_2 N_2(0)e^{-\lambda_2 t}$$

Above is the main ODE we would like to emphasize. Suppose C doesn't decay, C would follow:

$$\frac{dN_C}{dt} = \lambda_B N_B(t)$$

However, I didn't get the analytical solution to C, leaves space for improvement.

1.2 Problem 2 (numerical): Moment of Inertia for a non-trivial, not so symmetrical object (tetrahedron)

The moment of inertia has its axis defined going through one of its vertex, exiting at the center of the equilateral triangle on the other side. One might define the pointing direction of vertex with the spinning axis going through as positive Z axis. The I solution follows:

$$I_{ij} = \int \rho(q_i, q_j, q_k)(r^2\delta_{ij} - q_i q_j)dV$$

In the case if Z is spinning axis:

$$I_z z = \int\int\int \rho(x,y,z)(x^2+y^2+z^2-z^2)dxdydz = \int\int\int \rho(x,y,z)(x^2+y^2)dxdydz$$

In case if ρ is constant, it can be pulled out and be multiplied in the end. Here's the vertex setup for the tetrahedron I imagined:

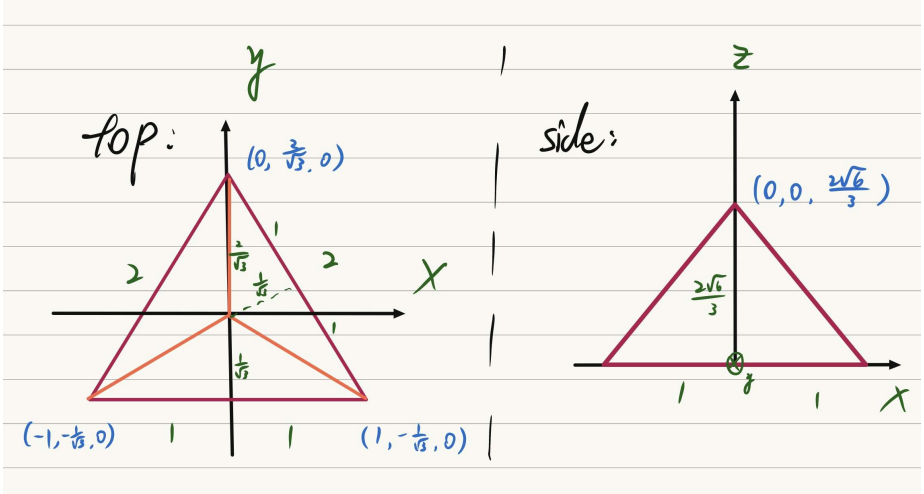

Figure 1: Vertex of Tetrahedron, useful for coding

2 Validation of Methods

This section will compare the result graphing solutions and validate their significance.

2.0.1 Compound Decay Comparison

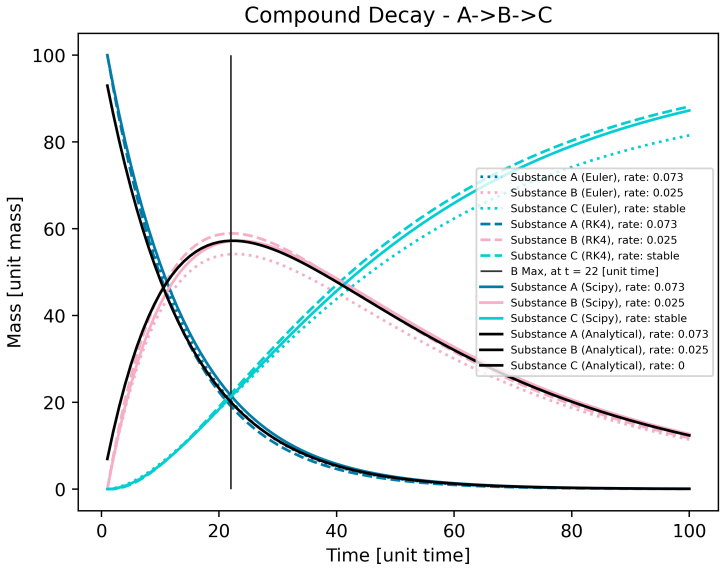

Figure 2: Decay Comparison

2.0.2 Decay Integral Error

Above is the direct comparison of four methods implemented (explicit Euler, Runge-Kutta 4th order, Scipy BDF, and Analytical) for each substance. Again, exception is analytical solution for substance C, which I wasn't able to mathematically solve for it. Leaves space for improvements.

In the graph, each substance has all of its numerical methods coded in same color, where analytical solution is in black solid. Within each substance, one can see the Euler method has the largest deviation from the analytical solution, with a local truncation error on roughly order of $O(h^2)$. The RK-4 having $O(h^5)$ locally, the BDF having even more precision. One would observe the black line to be very close to the scipy solution of substance C. Clearly, the explicit Euler method has the most error, followed by RK-4, and the scipy BDF method.

The Euler method predicts the next instance of mass with the current mass, and decay rate. In such case, one single linear line is drawn between the future point and the current, that is only depended on one point. Locally each time, this would introduce the most error. The RK4 takes average of six intermediate points at the middle of current- future time, introducing less error.

A vertical line I thought be fun to include marks point were A is adding mass to B with the same rate as B is decaying, at maximum of B mass.

2.0.3 Inertia Difference Comparison

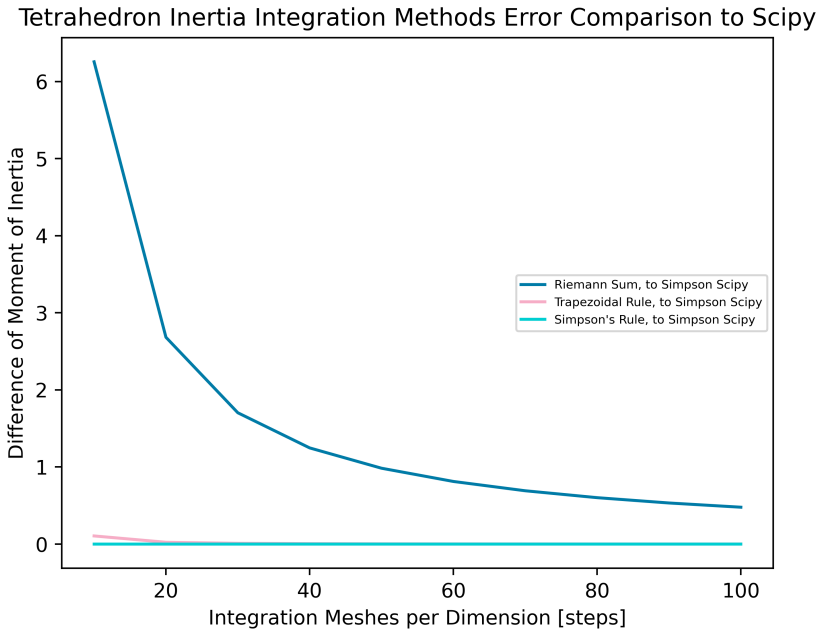

Figure 3: Inertia Difference across 10-100 Mesh Sizes in each Dimension

The above graph plots the difference in inertia value between the methods, relative to the scipy's Simpson's method. As one can see, the less the steps, the larger the error in all methods. Clearly, some has more than others.

The Riemann sum method sums up rectangles, in this case of 3D, small cubes. On the edge of the tetrahedron, this method becomes very inaccurate as it always over-calculate the inertia by overestimating the volume integral where it is already exceeding the boundary of our shape. As mesh becomes more fine, the error decreases as the overestimation becomes smaller and smaller. On the other hand, the trapezoidal rule poses a similar fashion, but having way smaller error than the Riemann sum as it allows tilted surfaces when it comes to volume integral. As one can see, the difference reaches near-zero between trapezoidal and scipy's Simpson's rule as soon as mesh size increases to 20.

2.0.4 Inertia Time Difference

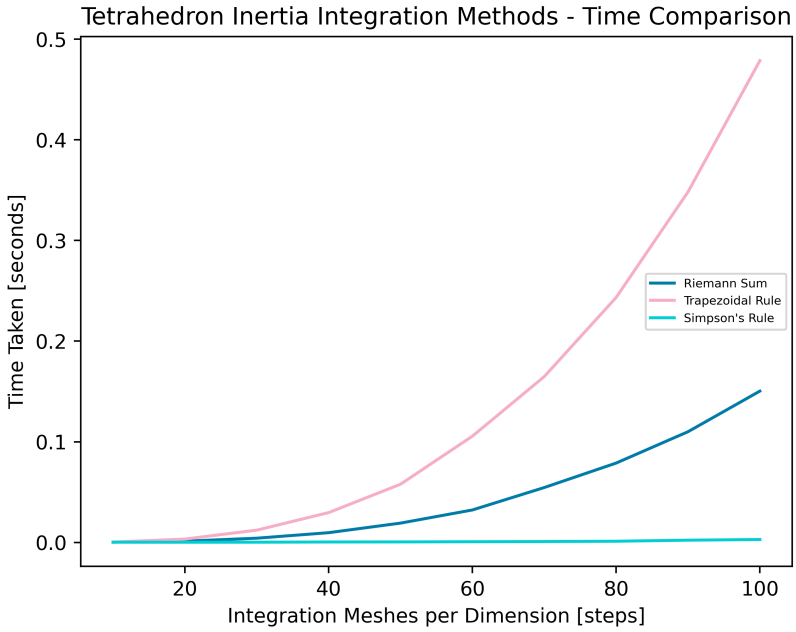

Figure 4: Time Comparison between Methods of Inertial Integration

As one can see, three methods used about the same time at mesh size of 10 in each dimension. AS mesh finesse increases, so does the computational time. Simpson's rule here utilizes scipy's library, which is already a very optimized method, rather than the other two of my personal numerical integration methods. It is clear, that the numerical Riemann sum has the medium run-time, but the largest error. The numerical trapezoidal method has the longest run-time, and medium error. The scipy's Simpson's rule at last, have the quickest run-time, and the smallest error, converging to nearly the correct result even when meshing is as coarse as 10 grids per dimension.