Computational Project 1

Zae Moore

October 1, 2023

Introduction

The purpose of this project is to create a program in Python capable of solving a 1D integral and ODE that have real world applications in physics. For the integral, I chose to solve for the Electric field inside and outside of a sphere with a charge density dependent on r. For the ODE, I chose to solve Newton's Cooling Law which dictates how an object's temperature changes in response to the temperature of the environment.

One-Dimensional Integral

Physics

For the 1D integral, I solved for the Electric field due to a sphere with charge density $\rho(r) = k * r$ where k is some constant that can be set in the code. For this problem, I set it to 5.

In CGI units, Gauss Law is $\int E * dA = 4\pi Q_{enc}$, where E is the electric field, dA is the area of the Gaussian surface that we integrate over, and Q is the total charge enclosed. Because the charge density is only dependent on r, there is spherical symmetry to this problem, so Gauss Law simplifies to $E = \frac{Q_{enc}}{r^2}$. To solve this, we first need to calculate the enclosed charge Q at any radius r using the charge density.

This will end up being two separate integrals, one for inside the sphere and one for outside the sphere. Inside the sphere, we have:

$$\begin{aligned} Q_{in} &= \int_0^{2\pi} \int_0^{\pi} \int_0^{r} \rho(r) r^2 sin(\theta) dr d\theta d\phi \\ &= 4\pi k \int_0^{r} r^3 dr \\ &= \pi k r^4 \end{aligned}$$

Outside the sphere, the enclosed charge will be the total charge within the sphere, so we have:

$$\begin{aligned} Q_{out} &= \int_0^{2\pi} \int_0^{\pi} \int_0^{R} \rho(r) r^2 sin(\theta) dr d\theta d\phi \\ &= 4\pi k \int_0^{r} r^3 dr \\ &= \pi k R^4 \end{aligned}$$

I allow the user to input the value of the radius of the sphere, but in my examples I chose 1m. So the total enclosed charge of the sphere becomes:

$$Q_{out} = \pi k$$

This is the end of the integral part, but to get the Electric field to plot we need one more step. Using Gauss' law from earlier, we find the inner and outer Electric field to be:

$$\begin{aligned} E_{in} &= \pi k r^2 \\ E_{out} &= \pi k \frac{R^4}{r^2} \end{aligned}$$

Ultimately, the goal is to plot the electric field vs. the radius.

Code

In order to solve for the Electric field, I utilized Gauss' Law. The first step is to then calculate the total charge Q enclosed within the sphere, both for r ¡ R and for r ¿ R (where R is the radius of the sphere, which I set to 1m in my examples).

The methods I utilized in my code are the Riemann sum, trapezoidal rule, Simpson's

rule. I also utilized sci-py's integral solving function with the trapezoidal and Simpson's methods and I plotted the exact analytical solution for comparison.

For the Riemann, trapezoidal, and Simpson methods, I asked the user to input a step size. The smaller the step size, the more accurate the result will be. Then, at each step, the code computes the area of that section utilizing the given method. This is all done through a loop that stops at the maximum radius that the user wants to measure. At each step/loop, this calculated area is added to a total sum which is then appended to a list. This ongoing list is the integral at each step. At the end, this will be plotted against the radius.

Results

For the first run here, shown in Figure 1, I used a sphere radius of 1m, a maximum radius of 100m, and a step size of 0.01.

The top figure shows the total charge enclosed in the sphere as calculated by each method. I found that this was the easiest way to compare the methods numerically. All of the methods except the Riemann sum method are extremely close to the analytical solution. Looking at the y-axis, the Riemann sum method isn't very far off compared to the other methods as well. The difference is about 30 times the step size. It is clear that the Riemann method is the least accurate in this situation, but overall still does a decent job. The bottom figure is the electric field from 0 to 100m. All of the methods line up quite well here, it's very difficult to see a significant difference.

For the next run, shown in Figure 2 I kept the physical properties the same but I changed the step size to 0.001, which should produce a more accurate result.

The smaller step size has made the coded methods even closer to the analytical result, although we can still see that the Riemann method is off from the analytical method by about 30 times the step size in the total charge calculation.

For the third set of results, shown in Figure 3, I changed the radius of the sphere to 10m, but kept the step size to 0.001.

The differences in the total enclosed charge are more pronounced here, you can see how my coded methods are noticeably off from the scipy and analytical solution. In comparison to the scale of the value, however, the accuracy is still quite close. The electricl field plot is also still very accurate, to the point that it's difficult to see any significant deviation.

From these results, it's clear that a smaller step size will get you better results with my coded methods. As you increase the radius of the charged sphere, the differences in my methods will also become more noticeable until you compare them to the scale of the values

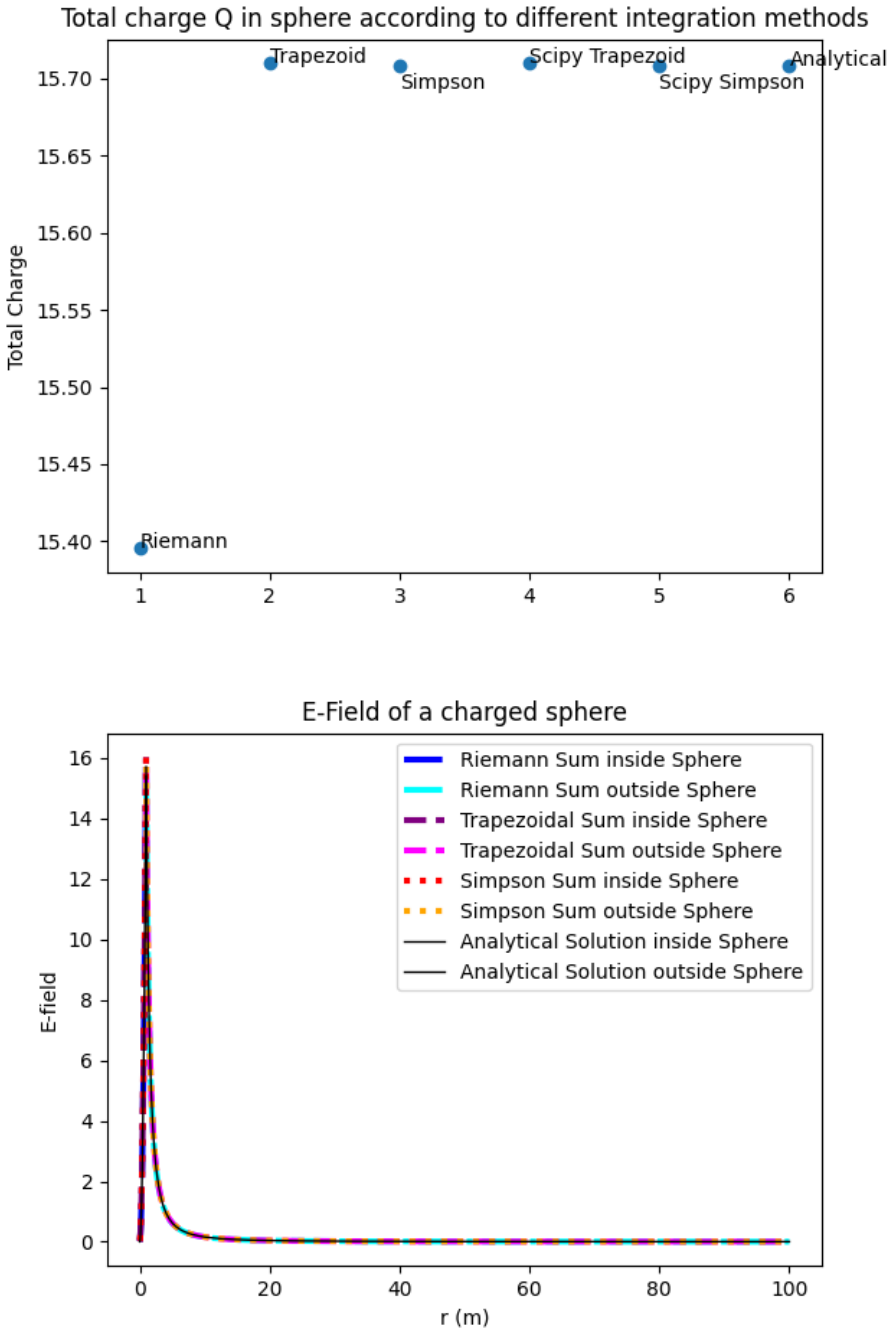

Figure 1: Results for R = 1m and step size = 0.01

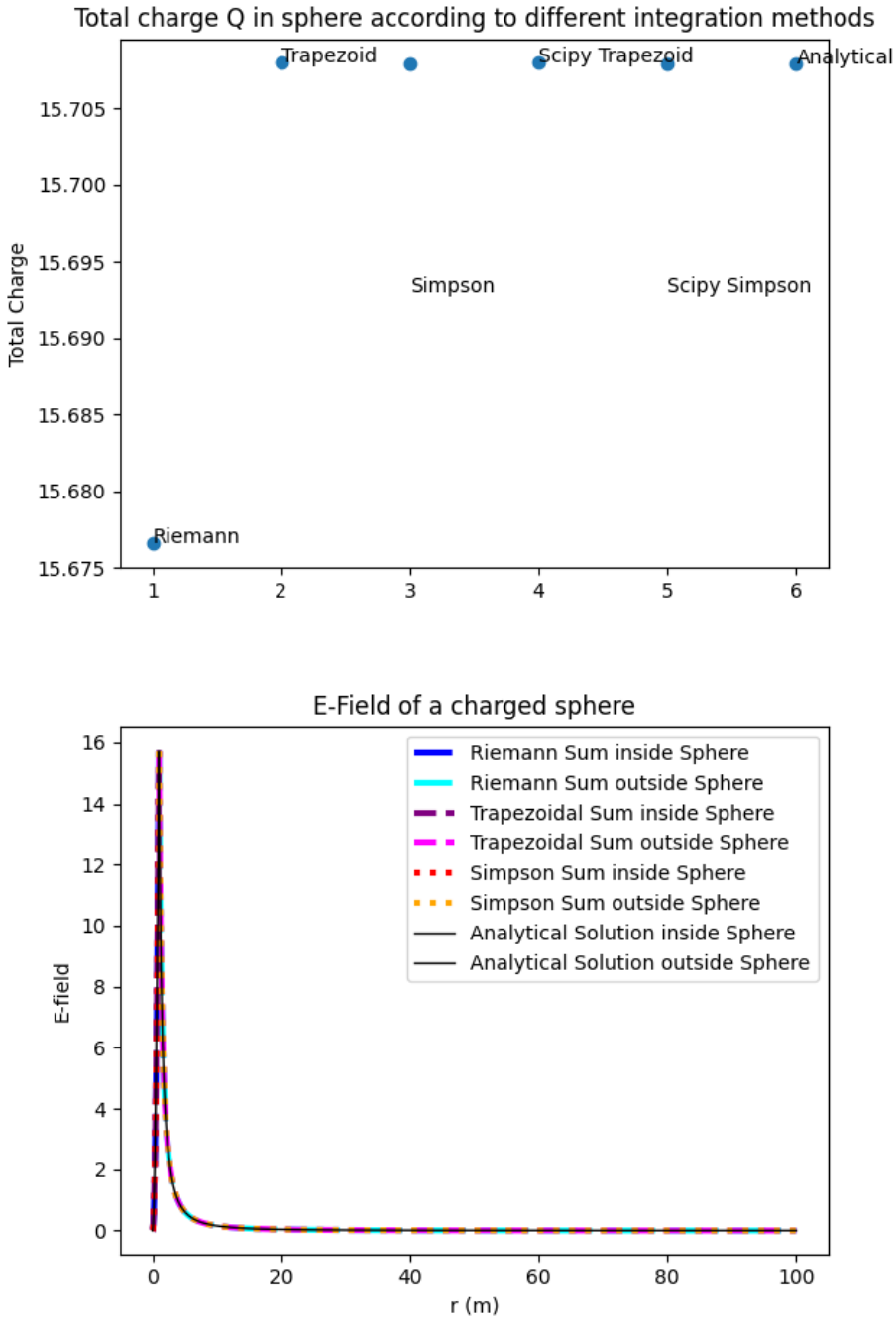

Figure 2: Results for R = 1m and step size = 0.001

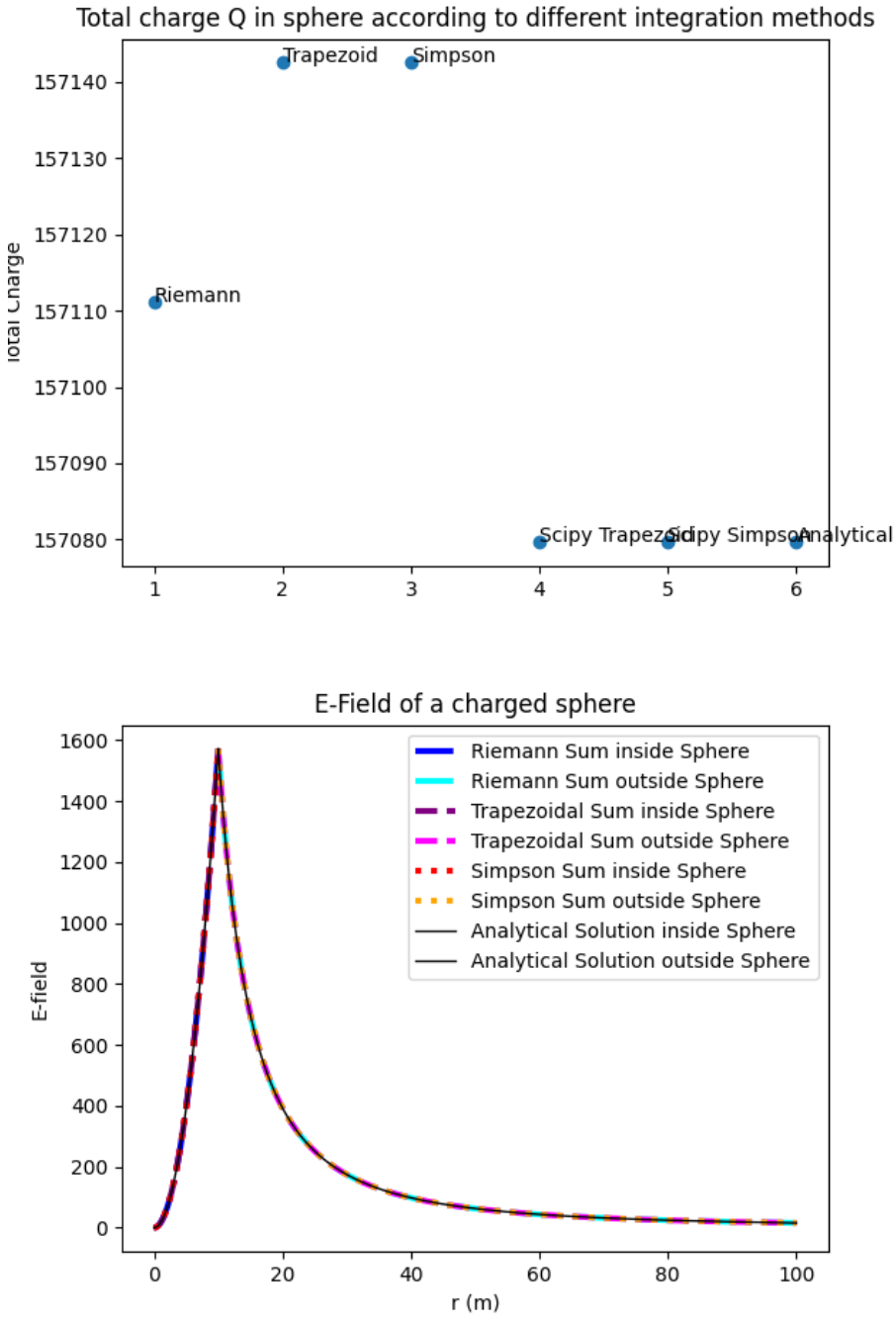

Figure 3: Results for R = 10m and step size = 0.001

produced. This is likely due to the fact that the integral is proportional to r^3, so as you increase r the differences become more noticeable. In order to minimize these differences, it's best to use a small step size.

Validation

In order to validate my solutions, I considered two separate checks. First, I checked that the electric field outside the sphere equals the electric field inside the sphere at r = R. This is checking that the electric field is constant at each point in space (reword this).
Second, I checked that the electric field goes to zero as r goes to infinity. The further from the charge sphere you go, the less strong the electric field should be, until it reaches zero infinitely far away.
Looking at the plots I produced in the results, both of these validation checks pass. All of the methods I made decently match the analytical solution, are constant at the radius of the sphere, and go to 0 as r goes to infinity.

ODE

Physics

For the ODE, I solved for Newton's Law of Cooling $\frac{dT}{dt} = -k(T(t) - T_{env})$ where $T(t)$ is the temperature of the object over time, T_{env} is the temperature of the environment around the object, and k is a positive constant that is related to the heat transfer coefficient (specific to a material and the type of heating occurring) and the surface area of the object. In my code, I set k to be equal to 3.0 and kept it at that value for the tests.
The analytical solution to this ODE is

$$T = Ce^{-kt} + T_{env}$$

Where C is a constant that is determined by the initial conditions such that $C = T_{env} - T(0)$

Code

The methods I utilized in my code are the Runge-Kutta 4th order method and the Euler method. I also utilized sci-py's ode-solving function and I plotted the exact solution for comparison.

For the Runge-Kutta and Euler methods, I asked the user to input step size. The smaller the step size, the more accurate the solution will be. For each method, I then had them loop over the time in which we are measuring the change in temperature. At the end of each loop, we increase the time by the step size.

Within each loop for each method, the code uses the given method to calculate the temperature at the time given and appends this value to a list. At the end of the loop, the method returns this list back to the main script which then plots it against the time list.

Results

For the first test I ran, I used a start object temperature of 100K, an environmental temperature of 200K, and a step size of 0.01. The results are shown in Figure 4

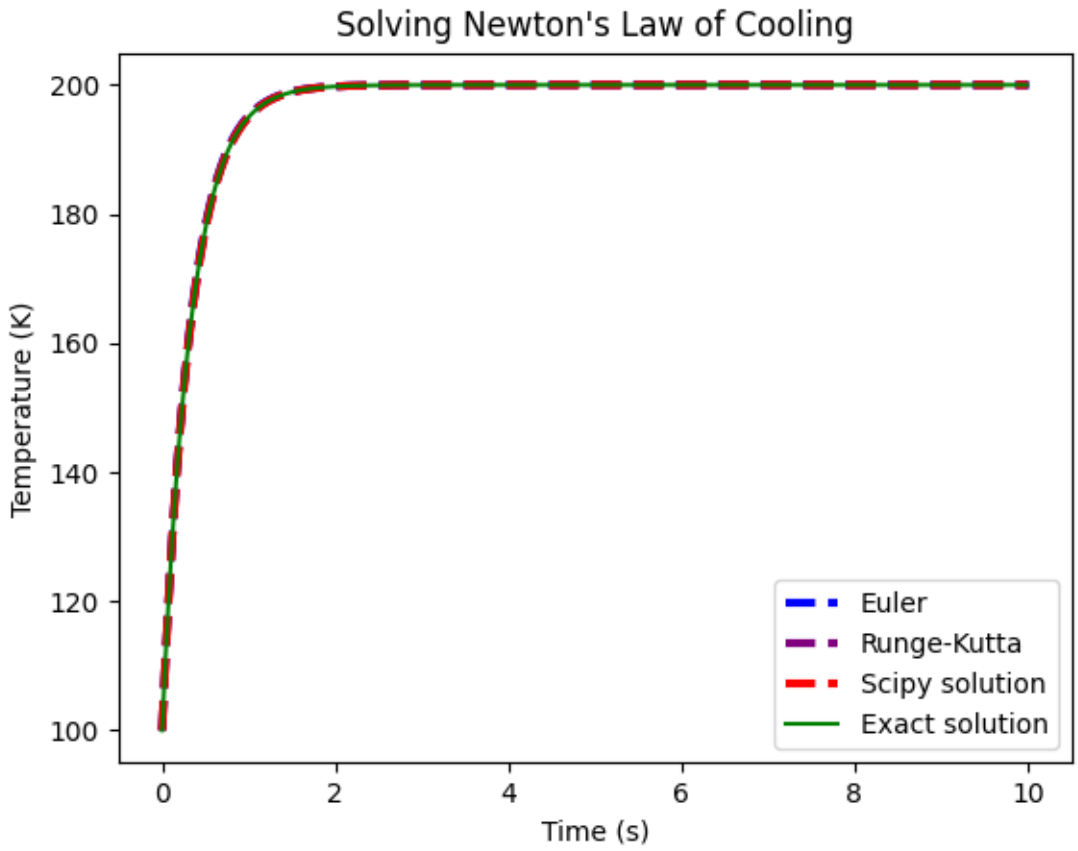

Figure 4: Heating object with step size = 0.01

The results here clearly show that my methods and scipy's method are very close to the analytical solution in the case of an object being heated.

The next test I ran was to see if the results would be as good if we were looking at cooling at object. I kept the same step size, but changed the initial temperature of the object to 300K and the environment temperature to 20K. The results are seen in Figure 5

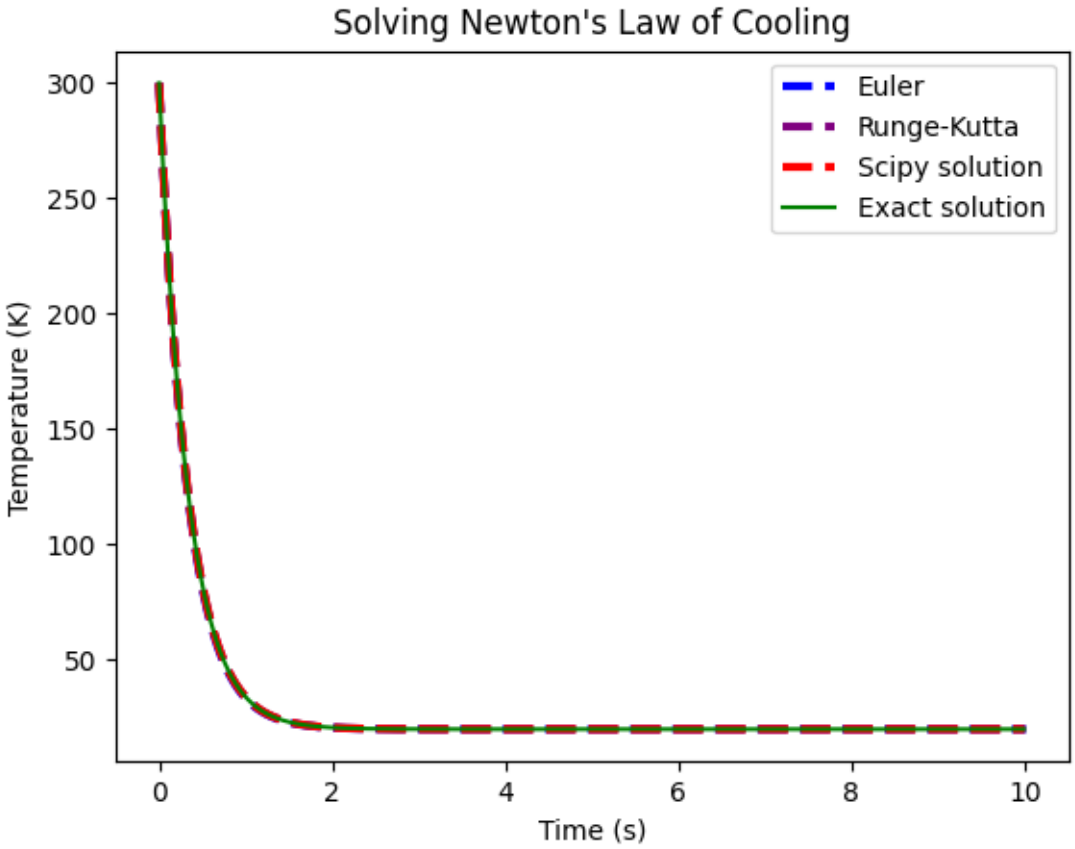

Figure 5: Cooling object with step size = 0.01

These results also show the accuracy of my coded methods and scipy's methods.

In order to find a situation in which my methods are less accurate, I reran the heating test with a step size of 0.1, as seen in Figure 6.

Here, we start to see some variation in the solutions of my methods from the analytical solution. The Euler method is, as expected, the farthest off. The Runge-Kutta and scipy solution are consistently right on top of each other and remain quite accurate to the analytical solution. It is clear that in order to achieve the most accurate solutions, a small step size (¡=0.01) is desired.

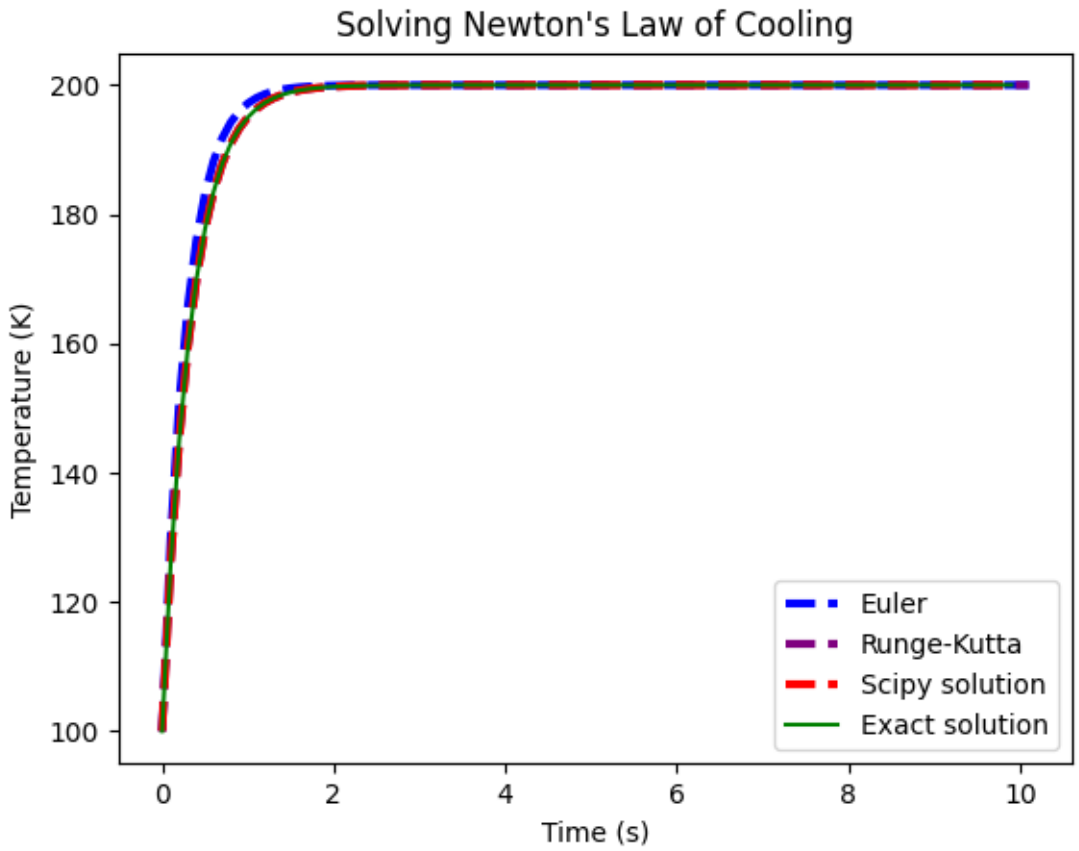

Figure 6: Heating object with step size = 0.1

Validation

In order to validate my solutions, I considered two separate checks. First, I checked that the temperature of the object is a smooth, continuous line that approaches the temperature of the environment without any jumps, discontinuities, etc. The second check is that in the limit of time going to infinity, the temperature of the object should always reach the temperature of the environment. Reasonably, this should happen long before t = infinity, but it should always happen in that limit regardless.

Looking at the resulting plots, all of the methods I constructed pass these checks. The temperature functions are smooth and continuously move towards the temperature of the environment before they level off once they reach that temperature. The methods I coded also match the analytical solution quite well, with the general shape and values being very close.

Conclusion

In conclusion, I had coded multiple methods for solving both a 1D integral and an ODE. My various methods tend to do quite well when compared to the analytical solution as long as a step size ¡= 0.01 is used. For the 1D integral, the trapezoidal method and Simpson's method consistently do better than the Riemann method, regardless of step size. For the ODE, the Runge-Kutta method does better than the Euler method. At a small enough step size, however, the differences are almost unnoticeable.