Search Medium Write

Member-only story

How to Solve ODEs in MATLAB Using Runge-Kutta

Learn to code a Runge-Kutta 4th-order algorithm to solve ordinary differential equations

Zack Fizell · Following
Published in Towards Data Science · 6 min read · Mar 28, 2022

70 2

Photo by Joshua Woroniecki on Unsplash

Ordinary differential equations (ODEs) are common in all facets of engineering and science. They are used to model physical phenomena such as turbulent air flow, satellite orbits, heat transfer, vibrations, and countless others. They are at the foundation of how humans understand physics and mathematics. While we can create these equations based on intuition and our fundamental understanding of how the world works, solving them can be even more challenging.

. . .

Ordinary Differential Equations

In this article, I'll be assuming you have a fundamental understanding of calculus and differential equations. The term ordinary differential equations refers to differential equations that contain functions of one independent variable and the derivatives of those functions. For most engineering and science ODEs, the independent variable is time and the equations take the form as follows:

$$\dot{\vec{y}} = f(t, \vec{y}(t))$$

$$\vec{y} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad \dot{\vec{y}} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dots \\ \dot{x_n} \end{bmatrix}$$

Here, the ***y*** vector can be a spatial state, mass, temperature, etc., and ***t*** is time. Unfortunately, most ODEs cannot be solved analytically; this means a function cannot be derived to provide an exact solution to the differential

equation. Another way to solve these differential equations is by utilizing numerical integration methods.

. . .

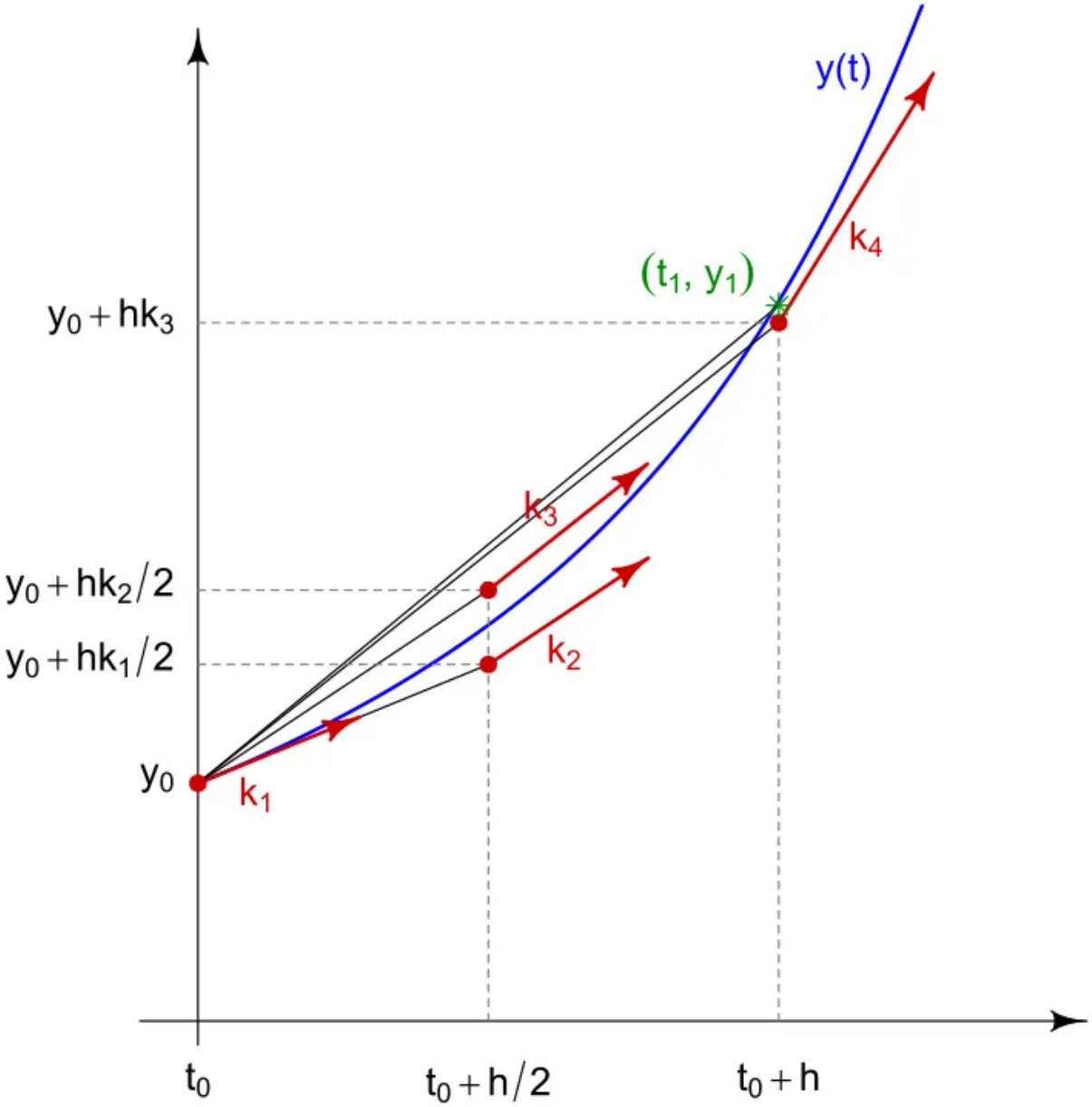
Numerical Integration

The term numerical integration was first coined in 1915, but the benefits of it were not truly seen until modern computers. Numerical integration is a method to approximate the change of a function y throughout time by knowing the differential equations that govern the change of y in time. They are an estimation as stated, so they are only as good as the methods and techniques used. One of the most popular forms of numerical integration is the 4th-order Runge-Kutta method (or RK4).

RK4 can be described by the equations and diagram below. More often than not, you will have a vector form of ODEs to solve, so the vector form of RK4 is shown. In the equations, the k values are slope estimates of y calculated using the differential equations at locations shown in the diagram. Once the k values are determined, they are averaged and multiplied by the time step, h. This is then added to the current value of y to get the next time step approximation of y. This is process repeated until the desired number of time steps is reached. Note that the smaller the h value used, the more accurate the result will be to the true value.

$$\vec{k_1} = f(\vec{y_i}, t_i)$$
$$\vec{k_2} = f\left(\vec{y_i} + \frac{h}{2}\vec{k_1}, t_i + \frac{h}{2}\right)$$
$$\vec{k_3} = f\left(\vec{y_i} + \frac{h}{2}\vec{k_2}, t_i + \frac{h}{2}\right)$$
$$\vec{k_4} = f(\vec{y_i} + h\vec{k_3}, t_i + h)$$

$$\vec{y_{i+1}} = \vec{y_i} + \left(\frac{\vec{k_1}}{6} + \frac{\vec{k_2}}{3} + \frac{\vec{k_3}}{3} + \frac{\vec{k_4}}{6}\right) h$$



Depiction of Slope Estimates [Created by HilberTraum]

. . .

MATLAB RK4 Script

For this example, let's say we have a particle whose motion is described by the following ordinary differential equations:

$$\frac{dv}{dt} = w$$
$$\frac{dw}{dt} = 6v - w$$

This set of ODEs is convenient because it can be solved analytically (solution shown below). These were chosen, so that our RK4 algorithm could be compared to the actual solution. Keep in mind when using your own ODEs, they may not be solvable analytically, so you might not have anything to compare your results to.

$$v(t) = 2e^{2t} + e^{-3t}$$
$$w(t) = 4e^{2t} - 3e^{-3t}$$

We can start the code by defining a function, ***model,*** which will return the value of the ODEs when values ***t***, ***v***, and ***w*** are passed to it. Make sure you put this function definition at the bottom of your script.

```
function dydt = model(t,y)
    v = y(1);
    w = y(2);
    dv_dt = w;
    dw_dt = 6*v-w;
```

```
    dydt = [dv_dt,dw_dt];
end
```

Let's start the main script now. I always start it by clearing the command window, clearing variables, and closing figures.

```
clc
clear variables
close all
```

In order to utilize the RK4 method, we need to define initial conditions, *y0*, and a time array, *t*. Here, *t* is defined to go from 0 to 1 in 1,000 increments and the initial condition for v is 3 and w is 1. The time array is used for creating h and later, the exact values for v and w.

```
% Initial Conditions and Simulation Time
y0 = [3, 1];  % y0 = [v0, w0]
t = linspace(0,1,1000)';
```

Next, we can create an array, y_{kutta}, to store the RK4 approximations at each time step. Using the initial conditions for v and w, we can define the first index ($i = 1$, $t = 0$) of the RK4 array. We can also define the time step variable, h. We then introduce a *for* loop to iterate through the length of the time array. Each iteration we will be adding a new value to y_{kutta} and then using this iteration to create another value for y_{kutta}. This process will repeat until we reach the last time step ($t = 1$).

```
% Runge-Kutta 4th-Order Algorithm
y_Kutta = zeros(length(t), 2);
```

```
y_Kutta(1, :) = y0;
h = t(2)-t(1);  % Constant time step
for i = 2:length(t)
    k1 = model(t(i-1), y_Kutta(i-1, :));
    k2 = model(t(i-1)+h/2, y_Kutta(i-1, :)+k1*h/2);
    k3 = model(t(i-1)+h/2, y_Kutta(i-1, :)+k2*h/2);
    k4 = model(t(i-1)+h, y_Kutta(i-1, :)+k3*h);
    y_Kutta(i, :) = y_Kutta(i-1, :)+(k1/6+k2/3+k3/3+k4/6)*h;
end
```

As mentioned before, this set of ODEs has an analytical solution, so let's create an array of those values and compare it to our result from RK4. To compare, we can simply calculate the difference between the exact solution and our approximated solution.

```
% Exact
y_Exact = 2*exp(2*t).*[1, 2]-exp(-3.*t)*[-1, 3];

% Calculating the Difference Between Exact and RK4 Solutions
diff_Kutta = y_Exact-y_Kutta;
```

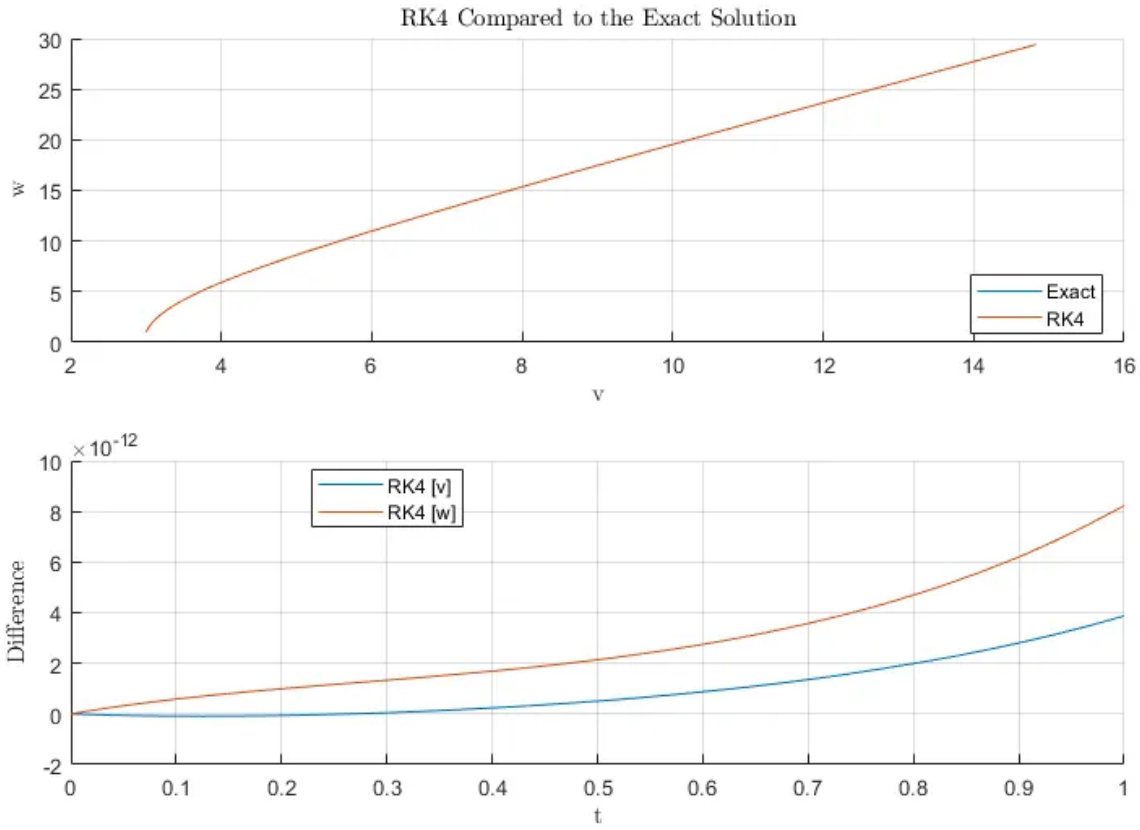
Finally, we have all the data we need to compare the Runge-Kutta 4th-order method to the exact solution. The last step is plotting the results.

```
% Comparison Subplot
figure; subplot(2,1,1);  hold on;
plot(y_Exact(:, 1), y_Exact(:, 2))
plot(y_Kutta(:, 1), y_Kutta(:, 2))
title('RK4 Compared to the Exact Solution', 'Interpreter', 'Latex')
xlabel('v', 'Interpreter', 'Latex')
ylabel('w', 'Interpreter', 'Latex')
legend('Exact', 'RK4', 'Location', 'Best')
grid on; hold off

% Difference Subplot
subplot(2,1,2); hold on;
plot(t, diff_Kutta(:, 1))
plot(t, diff_Kutta(:, 2))
xlabel('t', 'Interpreter', 'Latex')
```

```
ylabel('Difference', 'Interpreter', 'Latex')
legend('RK4 [v]', 'RK4 [w]', 'Location', 'Best')
grid on; hold off
```

The results:



Runge-Kutta 4th-Order Results [Created by Author]

As you can see, the results of the Runge-Kutta 4th-order method are very accurate for this example. The RK4 results are indistinguishable from the exact solution in the top plot, and the difference between the results for v and w and the exact solution are practically negligible. Runge-Kutta can be extended beyond 4th-order to reduce error, but that's probably not necessary for most applications.

. . .

Thank you for reading the article! Hopefully you learned a little about RK4! Give me a follow and check out my other articles on Python, mathematics, and orbital mechanics! If you have any comments or concerns, let me know!

Matlab Programming Machine Learning Engineering Mathematics

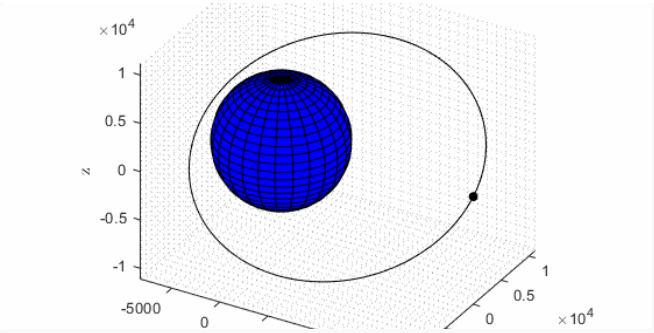
Written by Zack Fizell

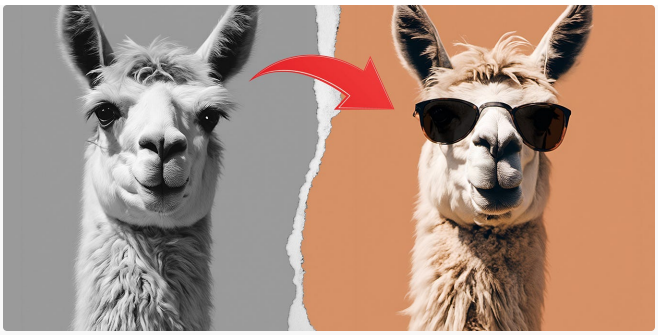
Following

1.4K Followers · Writer for Towards Data Science

M.S. in Aeronautics and Astronautics — Articles on Orbital Mechanics| Machine Learning| Coding — https://medium.com/@zackfizell10/membership

More from Zack Fizell and Towards Data Science





Zack Fizell in Towards Data Science

How to Use MATLAB to Create Two-Body Orbits

Step by step walkthrough on using MATLAB to determine how a spacecraft moves under...

· 6 min read · Jul 21, 2022

254

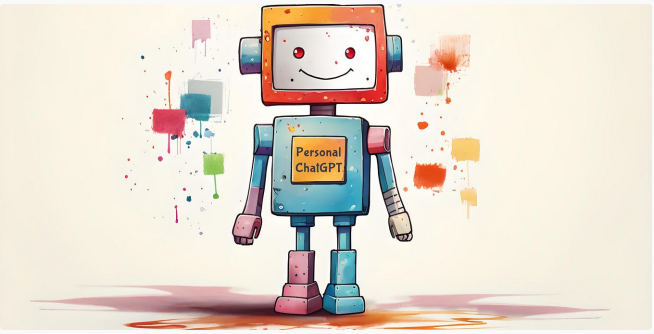
Maxime Labonne in Towards Data Science

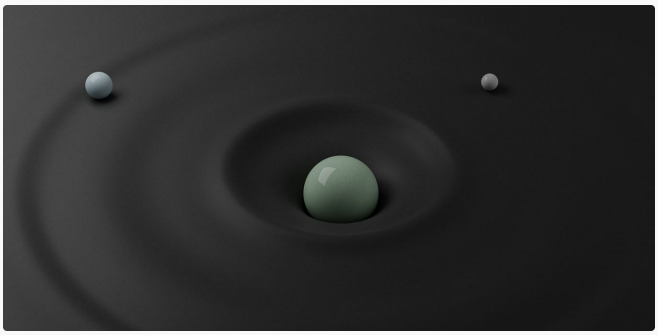
A Beginner's Guide to LLM Fine-Tuning

How to fine-tune Llama and other LLMs with one tool

· 8 min read · Aug 30

411 5





Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

· 15 min read · Sep 7

564 7

Zack Fizell in Intuition

What Are Gravity Wells?

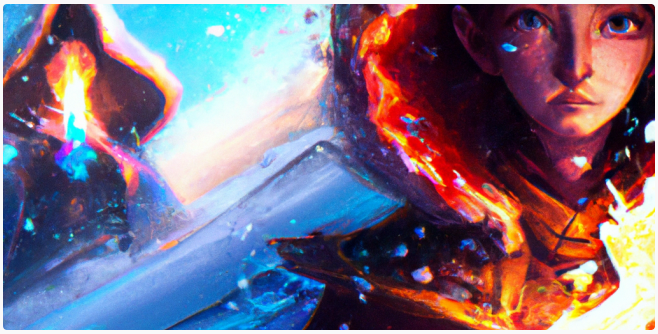
How do we escape the gravitational pull of a planet?

· 5 min read · May 2, 2022

99 4

See all from Zack Fizell | See all from Towards Data Science

Recommended from Medium



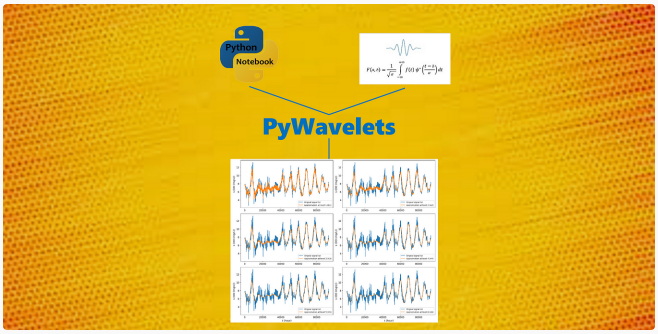
Hennie de Harder in Towards Data Science

An Introduction to a Powerful Optimization Technique: Simulate...

Explanation, parameters, strengths, weaknesses and use cases

✦ · 9 min read · Mar 15

416 7



Dr. Shouke Wei

Multilevel Discrete Wavelet Transform and Noise Reduction o...

How to make multilevel Discrete Wavelet Transform and noise reduction of 1D Time...

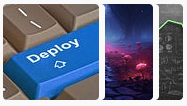
✦ · 8 min read · Mar 21

110 3

Lists

It's never too late or early to start something

15 stories · 128 saves

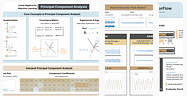


Predictive Modeling w/ Python

20 stories · 397 saves

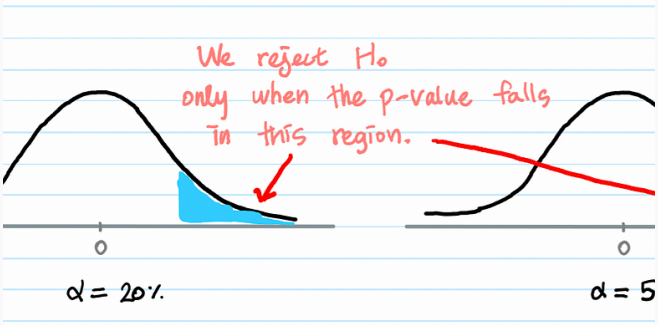
General Coding Knowledge

20 stories · 353 saves



Practical Guides to Machine Learning

10 stories · 457 saves



Ms Aerin in IntuitionMath

Chi Square Test—Intuition, Examples, and Step-by-Step...

The best way to see if two variables are related.

✦ · 15 min read · Feb 12

377 3

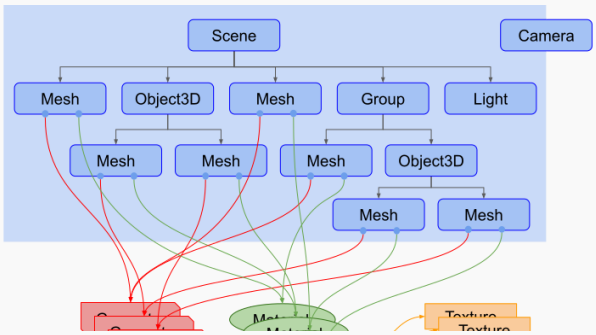
Franciszek Szewczyk

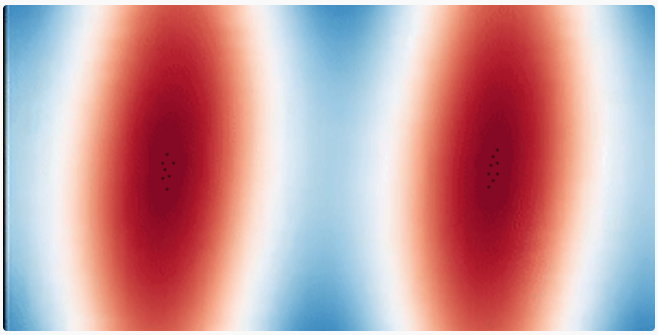
Rope Simulator in C++

For starters, we're going to implement a simple rope simulator. We will be using C++,...

7 min read · Sep 11

26 1





Rabin Lamichhane

Three.js

Three.js is a popular open-source JavaScript library used for creating and displaying 3D...

4 min read · Apr 27

15

Philip Mocz in Level Up Coding

Create Your Own Navier-Stokes Spectral Method Fluid Simulation...

For today's recreational coding exercise, we solve the Navier-Stokes equations for an...

✦ · 7 min read · Aug 3

634 3

See more recommendations