Member-only story

Develop Your Own Newton-Raphson Algorithm in Python

Solve for optimum solutions, equilibrium points, etc. using the NR method

Zack Fizell · Following

Published in Towards Data Science · 7 min read · Apr 14, 2022

Photo by David Clode on Unsplash

The Newton-Raphson method is an iterative method used to approximate the roots or zeros of a function. Determining roots can be important for many reasons; they can be used to optimize financial problems, to solve for equilibrium points in physics, to model computational fluid dynamics, etc. As you can see the uses extend well beyond any one subject. Generally, in complex equations the roots cannot be solved for explicitly, so they must be approximated; that's where the Newton-Raphson method comes into play.

. . .

The Newton-Raphson method (or algorithm) is one of the most popular methods for calculating roots due to its simplicity and speed. Combined with a computer, the algorithm can solve for roots in less than a second. The method requires a function to be fit into the following form. This can be done in most cases by simple addition or subtraction.

$$f(x) = 0$$

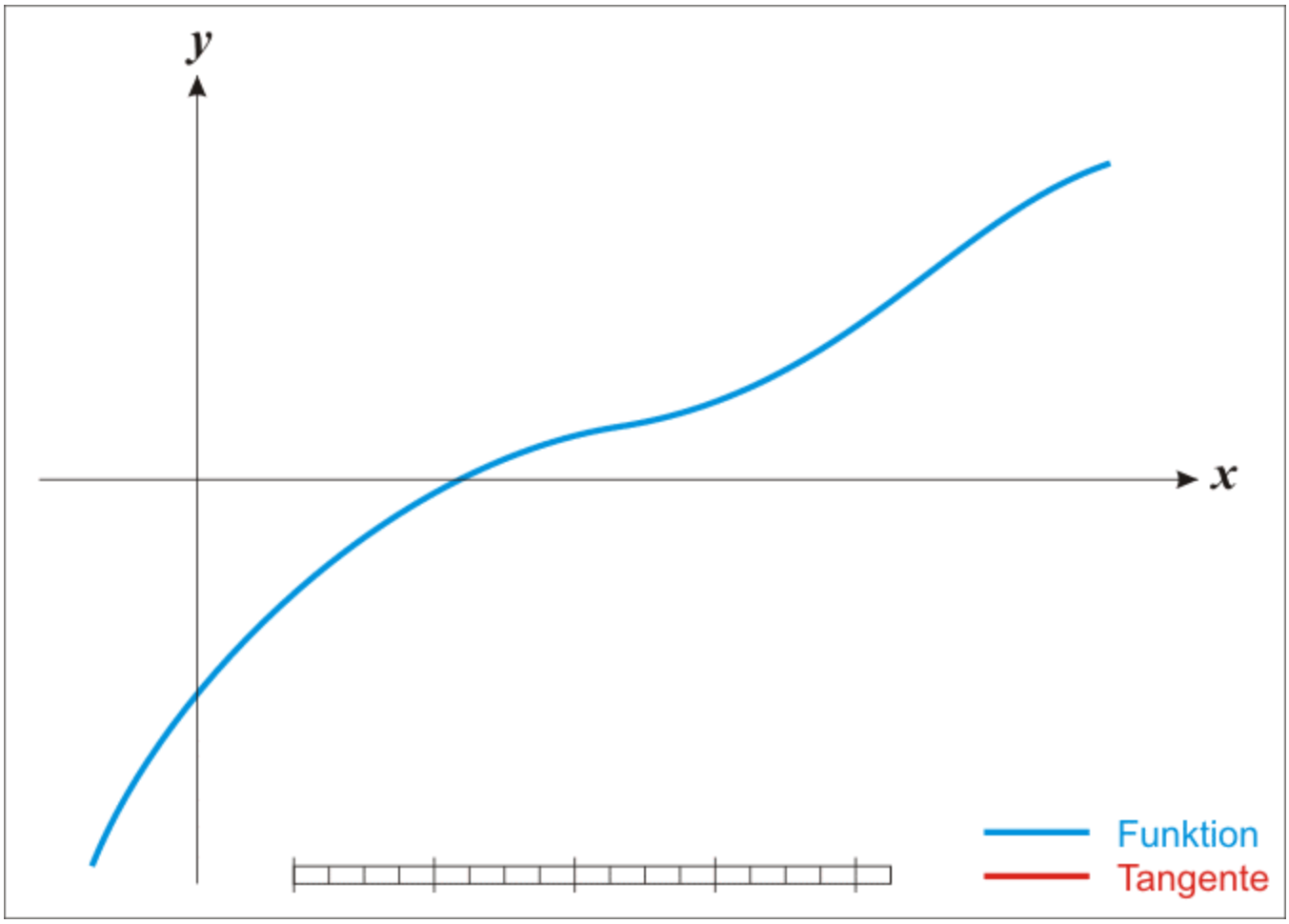
Take the example below. Simple subtraction is all that is needed to transform the equation into the form above. The left side of the equation would then be the function, *f(x)*. We will be using this equation later on, so keep it in mind.

$$\cos x = 2x^3$$

to

$$\cos x - 2x^3 = 0$$

The first step of the method takes an initial guess and uses the function and function derivative to calculate a next guess. Then, this guess is used in a similar fashion to calculate the next guess, and so on, until a tolerance or iteration limit is met. As seen in the animation below, the derivative, *f'(x)*, (the red line) is used as a slope to assist in calculating the next guess for *x*.



Newton-Raphson Method Visualized [Created by Ralf Pfeifer]

Let's write out what is happening in equations, so that it makes a little more sense. The first iteration would look like this:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

The second iteration:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

The subsequent iterations until some tolerance on the value of $f(x_i)$ (or an iteration limit) is hit:

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}$$

If you (or a code) can calculate the derivative of $f(x)$, then you can use this algorithm to iterate and calculate an equation's root(s). Your initial guess can be very important. Depending on the problem, if you start with a terrible guess, it can make your convergence take a while or not at all. However, with a good guess, the Newton-Raphson algorithm will converge on a solution relatively quickly within a few iterations. Your initial guess is also important if you have an equation with multiple roots. Depending on your first guess, you could converge on any of the roots. These problems can be mitigated by using an educated initial guess.

The Newton-Raphson algorithm can be implemented using Python or any coding language relatively easily. A computer can arrive at the desired solution very quickly when the initial guess is decent as you will see in the example. We will also see what happens when you have a bad initial guess. Let's start the code:

Importing Packages

I usually start all of my Python codes by importing N*umPy* and *pyplot* from *Matplotlib*. *NumPy* (defined as *np* for ease of calling) is used for array

manipulation and basic mathematic functions such as cosine, sine, exponentials, and logarithmic functions. *Pyplot* (defined as *plt* for ease of calling) is used to create plots and visualize data. Both are great packages that have much more functionality than what I am showing here.

```
# Importing Packages
import numpy as np
import matplotlib.pyplot as plt
```

Defining Functions

In this section of code, we will be defining the equation we are trying to find the roots of and its derivative. These two equations will be defined as Python functions, so we can provide them an input value, x, and they will return the value of the equation and its derivative at that value. For reference, the equation we will be using is the one from the earlier in the article.

$$f(x) = \cos x - 2x^3$$
$$f'(x) = -\sin x - 6x^2$$

```
# Defining Equation and Derivative
def f(x):
    res = np.cos(x)-2*x**3
    return res

def dfdx(x):
    res = -np.sin(x)-6*x**2
    return res
```

Newton-Raphson Loop

This section provides the iteration loop necessary to determine the root of the equation of interest. Before the iterative *while* loop, it is good practice to

include a maximum number of iterations variable, ***max_iter***. This is important because it will prevent the *while* loop from running indefinitely if the algorithm cannot converge on a solution. The tolerance, ***tol***, is used to ensure we get the accuracy we desire. We also include the initial guess, x_0, before beginning our iterations.

Now, inside the while loop we use the Newton-Raphson general equation to get our next guess, ***xi***, from the previous guess, ***xi_1*** (x_0 for the first iteration). The ***xi*** value is then input in the original equation to check against the chosen tolerance. Here, we are determining if our new guess makes the original equation nearly zero. The *while* loop will be broken when either the absolute value of the equation is less than our tolerance or if we have hit the max number of iterations. If it is not broken, the loop continues to update guesses for the value of the root.

```
# Newton-Raphson Algorithm
max_iter = 20  # Max iterations
tol = 1E-15  # Tolerance
i = 0  # Iteration counter
x0 = 1  # Initial guess
xi_1 = x0
print('Iteration ' + str(i) + ': x = ' + str(x0) + ', f(x) = ' +
      str(f(x0)))

# Iterating until either the tolerance or max iterations is met
while abs(f(xi_1)) > tol or i > max_iter:
    i = i + 1
    xi = xi_1-f(xi_1)/dfdx(xi_1)  # Newton-Raphson equation
    print('Iteration ' + str(i) + ': x = ' + str(xi) + ', f(x) = ' +
          str(f(xi)))
    xi_1 = xi
```

You might have noticed that we print the values of ***xi*** and ***f(xi)*** at each of the iterations and what iteration they occur at. This helps us track how the algorithm is performing. The code will output the following:

```
Iteration 0: x = 1, f(x) = -1.4596976941318602
Iteration 1: x = 0.7866397888154096, f(x) = -0.2673205221391448
Iteration 2: x = 0.7261709381607133, f(x) = -0.018132645287873284
Iteration 3: x = 0.7214340390454733, f(x) = -0.0001059518195203335
Iteration 4: x = 0.7214060336500903, f(x) = -3.6893424981698786e-09
Iteration 5: x = 0.721406032674848, f(x) = -1.1102230246251565e-16
```

After looking at our output, it looks like the Newton-Raphson algorithm converges within 5 iterations. Pretty quick. It also approximates the solution to our very small tolerance. This means we had a successful convergence!

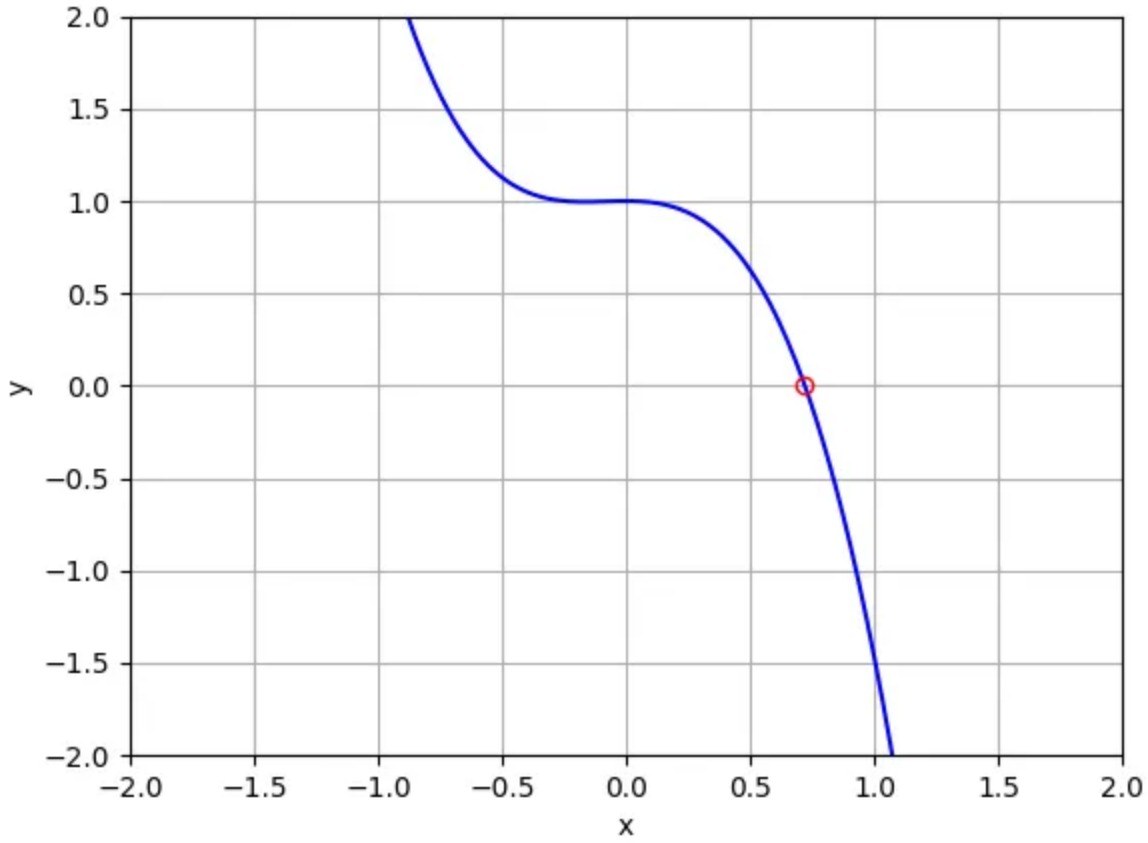
Plotting Equation and Result

You can check your result by plotting the equation for a range that encompasses your final root result. You can also plot your root value to see how your result matches up with the line.

```
# Creating Data for the Line
x_plot = np.linspace(-2, 2, 1000)
y_plot = f(x_plot)

# Plotting Function
fig = plt.figure()
plt.plot(x_plot, y_plot, c='blue')
plt.plot(xi, f(xi), c='red', marker='o', fillstyle='none')
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.show()
```

This will create the plot below. As you can see, the result from the Newton-Raphson algorithm lines up pretty well with where the line crosses the x-axis. This means we calculated a good approximation for the root of this example equation.



Newton-Raphson Example [Created by Author]

Let's try a bad guess and see how well the Newton-Raphson method does. If we had no idea where the zero might be, we might guess that root is around 100. This would result in the algorithm taking 17 iterations to reach the final result, but it still reaches the final result to the desired tolerance. That's the power of the algorithm. Here is the output from this trial for reference:

```
Iteration 0: x = 100, f(x) = -1999999.1376811278
Iteration 1: x = 66.66639972214995, f(x) = -592586.2434629743
Iteration 2: x = 44.44370527065607, f(x) = -175573.33453879558
Iteration 3: x = 29.629768898508654, f(x) = -52025.53704873974
Iteration 4: x = 19.751306671606308, f(x) = -15409.906816284196
Iteration 5: x = 13.170008297565373, f(x) = -4567.829385058682
Iteration 6: x = 8.783189380524185, f(x) = -1355.9491714365759
Iteration 7: x = 5.857511541738072, f(x) = -401.03685300814516
Iteration 8: x = 3.9055163566277287, f(x) = -119.86426242296075
Iteration 9: x = 2.585811794335439, f(x) = -35.42914973938403
```

```
Iteration 10: x = 1.7141632818943917, f(x) = -10.216519425753903
Iteration 11: x = 1.1654743473093252, f(x) = -2.7718839271038136
Iteration 12: x = 0.859829105239771, f(x) = -0.6187868181406497
Iteration 13: x = 0.7406842579217826, f(x) = -0.07469127827436395
Iteration 14: x = 0.7218536001388265, f(x) = -0.0016940906277266299
Iteration 15: x = 0.7214062815658834, f(x) = -9.415553438030244e-07
Iteration 16: x = 0.721406032674925, f(x) = -2.9121149935917856e-13
Iteration 17: x = 0.721406032674848, f(x) = -1.1102230246251565e-16
```

. . .

As you can see, this is a powerful method to get a very close approximation to the zero of an equation. The root can be solved for without guessing based on a plot or without solving for it algebraically (if even possible). This can be applied to many different fields, so try it out for yourself!

Thank you for reading the article! Let me know if you have any problems with the code or want to know more about this method. If you are interested, check out my other articles on Python, orbital mechanics, and physics!

Machine Learning | Python | Coding | Programming | Algorithms

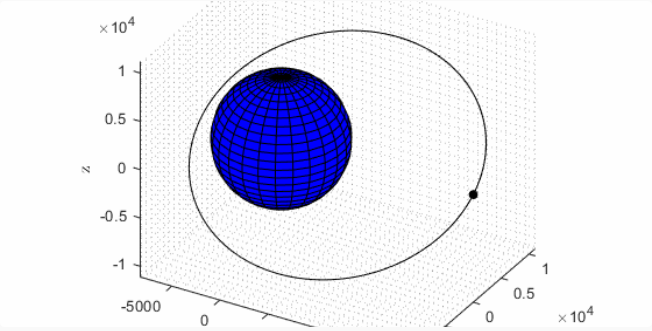
Written by Zack Fizell

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



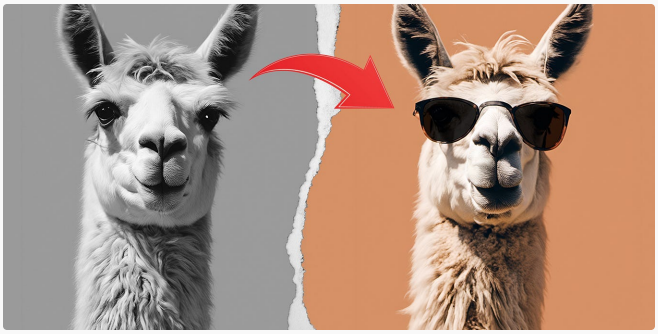
Zack Fizell in Towards Data Science

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



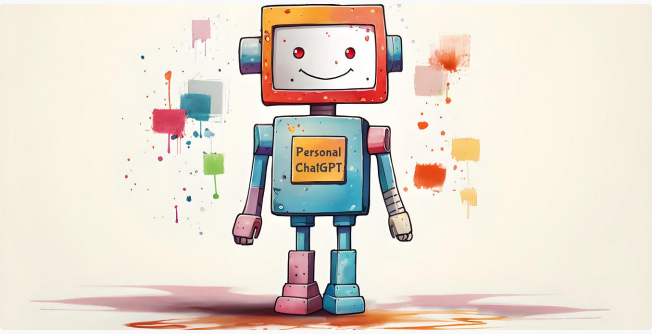
Maxime Labonne in Towards Data Science

A Beginner's Guide to LLM Fine-Tuning

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

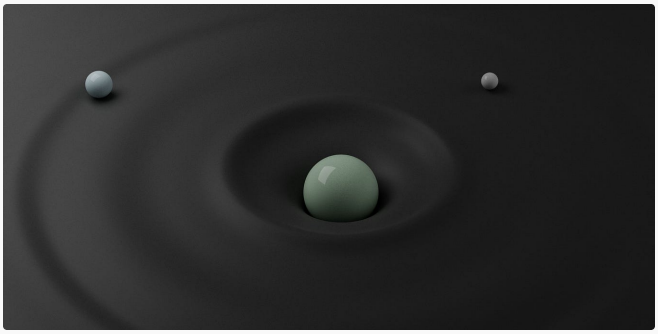
✦ · 8 min read · Aug 30

411 5



Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT



Zack Fizell in Intuition

What Are Gravity Wells?

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

How do we escape the gravitational pull of a planet?

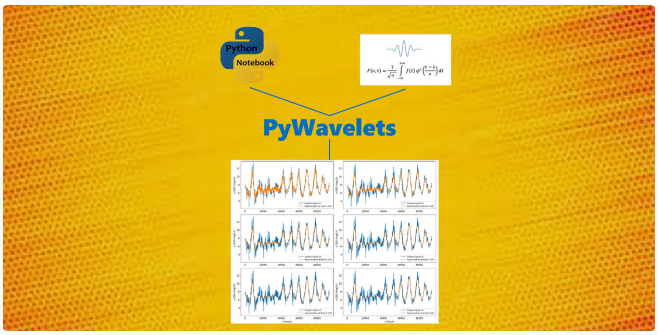
✦ · 5 min read · May 2, 2022

99 4

See all from Zack Fizell

See all from Towards Data Science

Recommended from Medium



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

Hennie de Harder in Towards Data Science

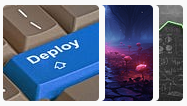
An Introduction to a Powerful Optimization Technique: Simulate...

Explanation, parameters, strengths, weaknesses and use cases

· 9 min read · Mar 15

416 7

Lists



Predictive Modeling w/ Python

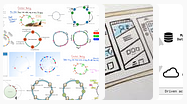
20 stories · 397 saves

Practical Guides to Machine Learning

10 stories · 457 saves

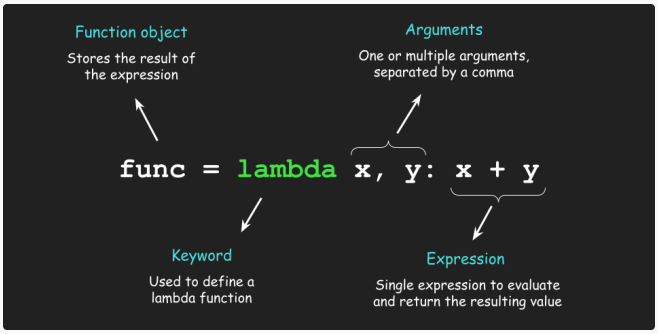
Coding & Development

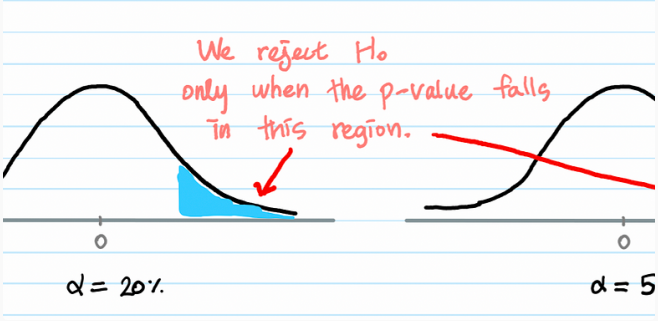
11 stories · 181 saves



General Coding Knowledge

20 stories · 353 saves





Ernest Asena

Lambda Functions in Python: Unleashing the Magic of Concise...

Welcome to a magical journey through the world of Python lambda functions, where...

4 min read · Aug 23

8

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

0	78.5	7	26	6	60
1	74.3	1	29	15	52
2	104.3	11	56	8	20
3	87.6	11	31	8	47

Ray Chen

Using Monte Carlo Simulation, Parameter Optimization and...

Monte Carlo simulation is a powerful technique used in various fields to model an...

17 min read · Jul 15

12 1

Siddharth Kumar

Statistical Arbitrage in the DEFI Index

Today I'm gonna implement the strategy described in the paper 'Statistical Arbitrage i...

5 min read · Mar 30

34 2

See more recommendations