Member-only story

Numerical Integration using Python

A simple method to numerically integrate equations and visualize results in Python

Zack Fizell · Following

Published in Towards Data Science · 6 min read · Jan 17, 2022

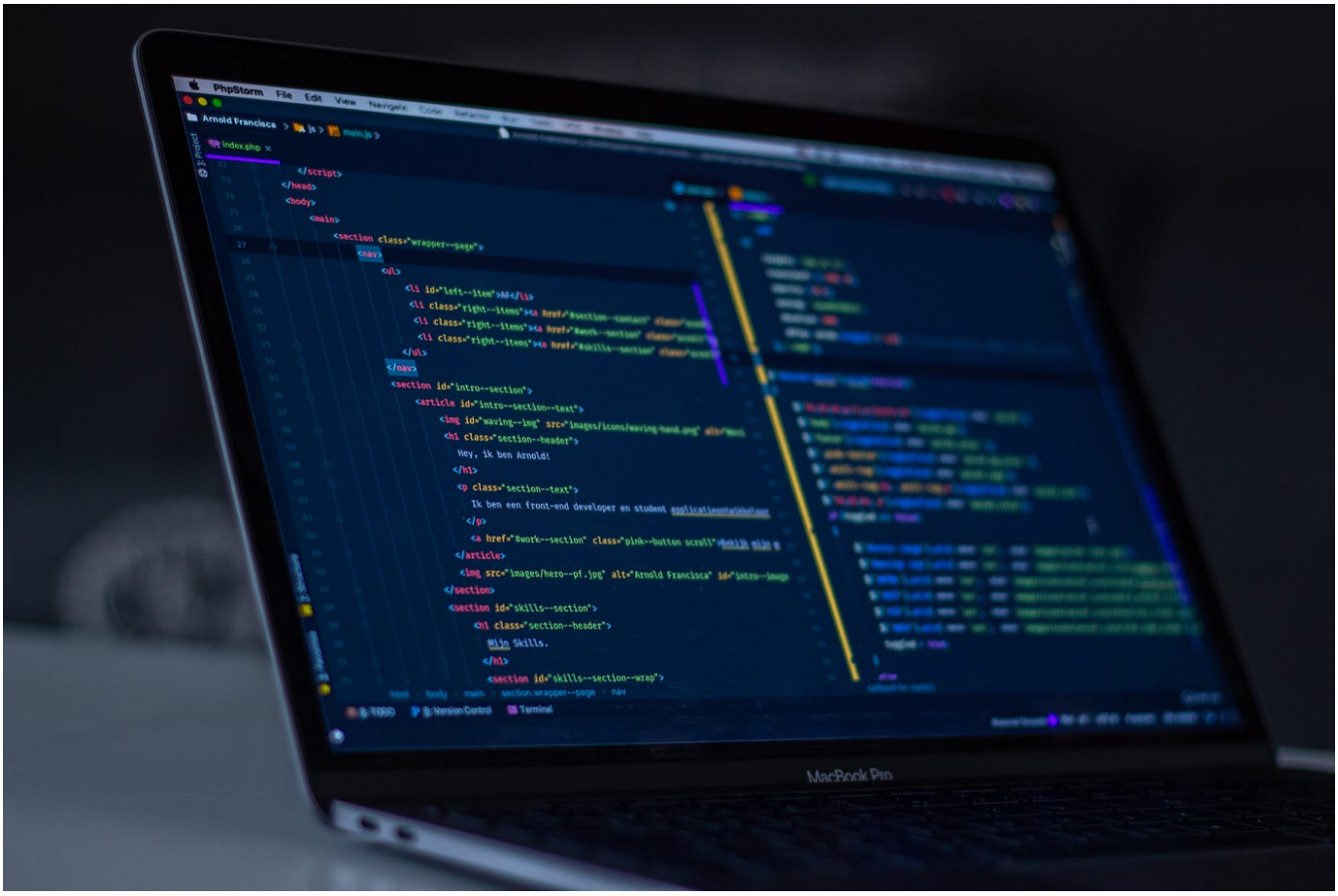
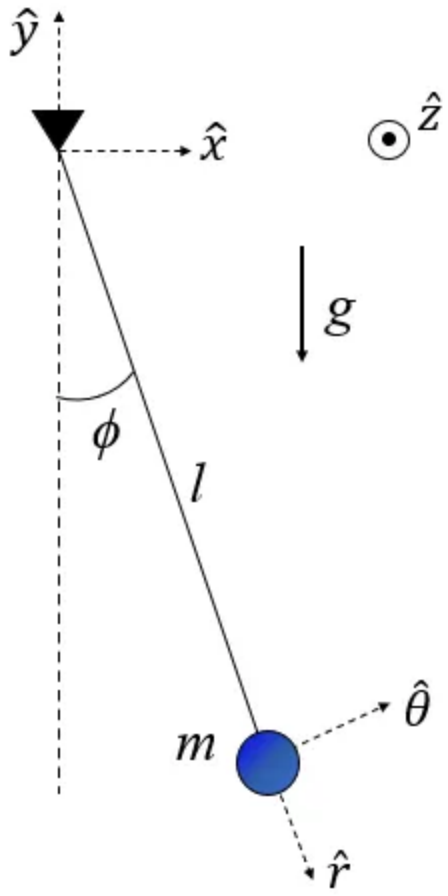


Photo by Arnold Francisca on Unsplash

Whether you are an engineer, physicist, or just a dynamics enthusiast, chances are you've had to work with equations of motion (EOMs) in the form of ordinary differential equations (ODEs). For those unfamiliar, ODEs are equations consisting of one or more functions of one independent variable along with their derivatives. In dynamics problems, the independent variable is typically time, or *t*, and the equations are typically related to position, velocity, and acceleration of a mass of interest.

While challenging, and often impossible, to solve analytically, ODEs can be approximated to high degree of accuracy using numerical methods. A daunting task before the age of the computer, but we have come a long way, so don't worry. Nowadays, most programming languages have ODE solvers available. Typically, they even have more than one solver, so there are no shortage of options. Python itself has numerous options to pick from, but we will focus on one for this demonstration.

. . .



Simple Pendulum Problem [Image by Author]

Throughout this article, we will use a simple pendulum as an example to guide us through the process. The equation of motion for the mass, *m,* (see equation below) is relatively straightforward, so it will serve as a good example. The principles of the code remain the same for all equations of motion, however, so you can use this for a wide variety of problems with a few adjustments.

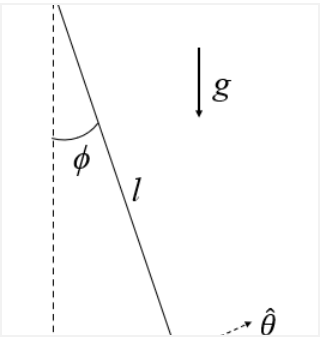
$$\ddot{\phi} = -\frac{g}{l} sin\phi$$

If you are interested in how the equation was derived, read article linked below.

How to Use Lagrangian Mechanics to Solve Dynamics Problems

An elegantly simple step-by-step process to solve conservative dynamics problems

medium.com



Before we dive into the code, it's good to know a little about how an ODE solver works. In this particular example, we are going to be solving an initial value problem, or IVP. For IVPs, the ODE solver starts with an initial time and initial conditions (these will be described soon), steps through a time interval, and calculates a solution at each time step. This is a high-level explanation, so I encourage you to do your research if you want to learn more.

ODE solvers require our equation of motion to be set up as a system of first-order differential equations. In its current state, our equation of motion is a second-order differential equation due to the second time derivative on ϕ. To change that, we make a "change of variables" to y_1 and y_2 and will have two first-order differential equations. Our new sets of equations are:

$$\vec{y} = \begin{bmatrix} y_1 = \phi \\ y_2 = \dot{\phi} \end{bmatrix}$$

$$\vec{y'} = \frac{d\vec{y}}{dt} = \begin{bmatrix} y_1' = \dfrac{dy_1}{dt} = \dot{\phi} = y_2 \\ y_2' = \dfrac{dy_2}{dt} = \ddot{\phi} = -\dfrac{g}{l} sin\phi = -\dfrac{g}{l} siny_1 \end{bmatrix}$$

Now that we have our new sets of differential equations, the next step is defining a starting point, or our initial conditions. They take the form of the y vector from the above equations. We don't need the y' vector just yet; we

will use that later. Our initial conditions are typically a starting position and velocity. In our case, it's our initial angle, ϕ, and its angular rate.

We can arbitrarily set the angle to 15˚ or π/12 radians and the angular rate to 0 radians per second (you can change these values, but your results will differ from the plots at the bottom of the article). The 0 subscript on our *y*-vector denotes the initial time.

$$\vec{y_0} = \begin{bmatrix} y_{1,0} \\ y_{2,0} \end{bmatrix} = \begin{bmatrix} \phi_0 \\ \dot{\phi}_0 \end{bmatrix} = \begin{bmatrix} \frac{\pi}{12} \\ 0 \end{bmatrix} \begin{matrix} radian \\ radian/s \end{matrix}$$

Starting the code, we need to import our necessary packages from Python. *NumPy* is a very powerful Python package used for scientific computation. We define it as *np* for ease of calling. Additionally, we need the *odeint* function from the *SciPy* package. This will be used to numerically integrate our ODEs. Lastly, we need to include *Pyplot* from *Matplotlib* to visualize our solved system. Like *NumPy*, we define *Pyplot* as *plt* for easier calling.

```
# Importing Packages
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

Next, we need to define a couple parameters for the *odeint* function. It takes three inputs in our code: the model (or our new first-order differential equations, *y'* vector), the initial conditions, and a time array that we want to integrate. Let's define our model function, *model(t, y)*, first. Here is what's in our code below:

- *g* = gravitational constant (9.81 m/s^2)

- l = length of massless rod (1 m)
- y = vector of our angle and angular rate at the current time step (defined earlier in article)
- *dydt* = time derivative of y vector (defined earlier in article)

```
# Model
def model(y,t):
    g = 9.81
    l = 1

    phi = y[0]
    phi_dot = y[1]
    phi_ddot = -g/l*np.sin(phi)
    dydt = [phi_dot, phi_ddot]
    return dydt
```

The second item our *odeint* function needs is our initial conditions. We already stated what those would be, so let's also state what time interval we are interested in. Say we want to look at how the pendulum mass behaves for 20 seconds. We could use the *linspace* function from *NumPy* (*np*) to create an array of time points from 0 to 20 seconds. I used 1,000 increments to ensure I get a smooth output from the ODE solver. The more increments you use, the more accurate your results will be. At some point, there is little to gain from the amount of increments, so use your best judgment when using this for future projects.

```
# Initial Conditions
y0 = [np.pi/12, 0]

# Time Array
t = np.linspace(0, 20, 1000)
```

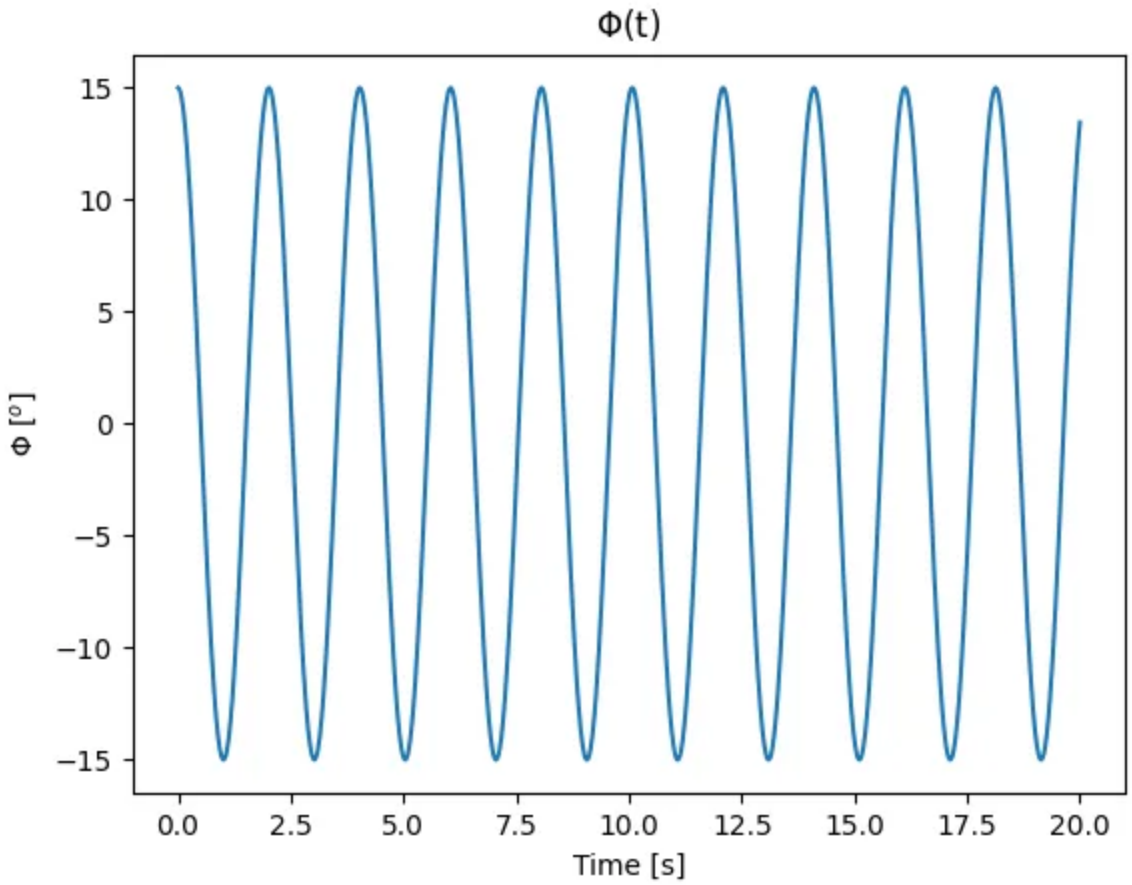
Finally, we just need to run the ODE solver with our model, initial conditions, and time array. We should set that equal to another variable, *sol*, so that we can manipulate the data for visualization using *Pyplot* (*plt*). We can pull our angle and angular rate time history by indexing the entirety of the first and second columns of *sol* (e.g. for the first column sol[:, 0]). The histories are in units of radians, so we can convert it to degrees for better understanding.

```
# Solving ODE
sol = odeint(model, y0, t)
phi_radians = sol[:, 0]
phidot_radians_s = sol[:, 1]

# Converting Radians to Degrees
phi_degrees = phi_radians*180/np.pi
phidot_degrees_s = phidot_radians_s*180/np.pi
```

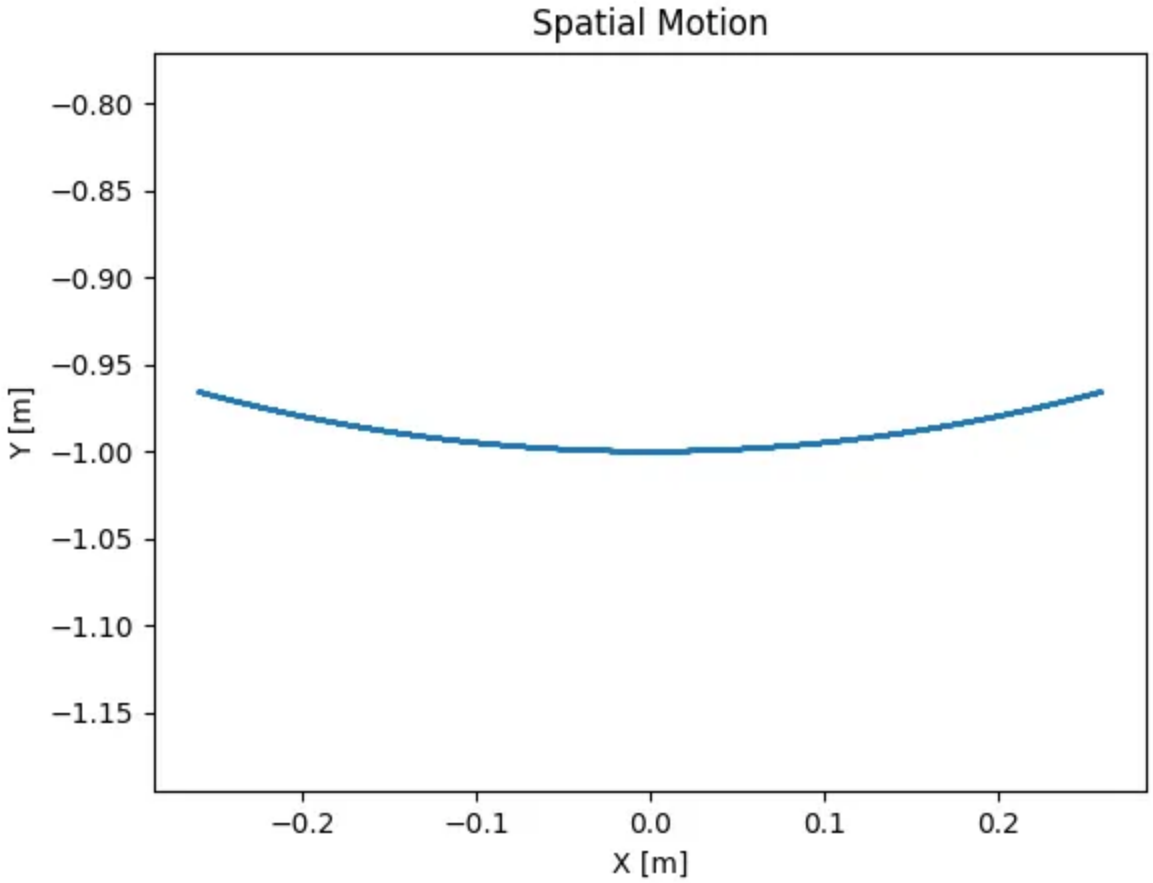
To visualize the data, we can use *plt* to create plots of the data that we are interested in. There are a lot of customization options with *Pyplot*, so you can easily change how your lines and plots look if you don't like the default.

```
# Plotting Results
plt.plot(t, phi_degrees)
plt.title('$\Phi$(t)')
plt.xlabel('Time [s]')
plt.ylabel('$\Phi$ [$^o$]')
plt.show()

X = np.sin(phi_radians)
Y = -np.cos(phi_radians)
plt.plot(X, Y)
plt.title('Spatial Motion')
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.axis('equal')
plt.show()
```

After running all of the code together, your plots should look something like this:



Angle Time History [Image by Author]



Motion of Pendulum Mass [Image by Author]

Our two plots show how our mass moves over the chosen time interval. The first plot shows how ϕ evolves over our time interval. Since we are looking at a pivot with no friction, we expect the mass to oscillate between +/- of our max angle, which happens to be our starting angle of 15˚ (since we did not have an initial angular rate). The second plot shows the trajectory of the mass over the 20 seconds interval. Intuitively, this motion makes sense, so we can reasonably assume our ODE solver and equations were set up correctly.

. . .

That is all for this example. If you have any questions, feel free to comment. I'd be more than happy to help. If this helped you out in any way, please give it a like and follow. Thank you!

Physics Python Mathematics Engineering Programming

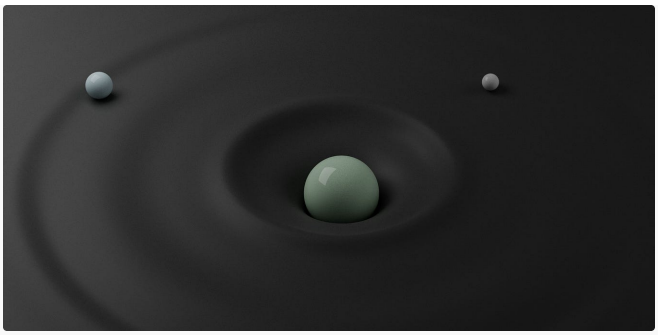
Written by Zack Fizell

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



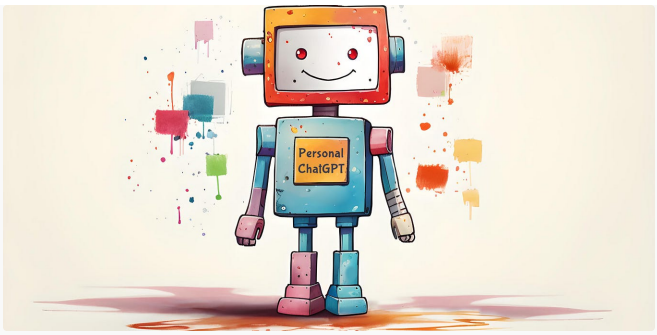
Zack Fizell in Intuition

What Are Gravity Wells?

How do we escape the gravitational pull of a planet?

✦ · 5 min read · May 2, 2022

99 4



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

```
class DataPreprocessor:
    def __init__(self, data):
        self.raw_data = data
        self.cleaned_data = self.clean_data(data)

    def clean_data(self):
        # imputation, outlier treatment
        ...

    def transform_data(self):
        # transformations and encode data
        ...
```

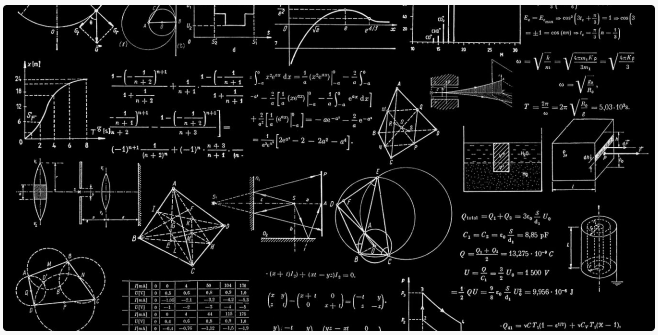
Molly Ruby in Towards Data Science

Object Oriented Data Science: Refactoring Code

Elevating machine learning models and data science products with efficient code and...

✦ · 7 min read · Aug 24

243 4



Zack Fizell in ILLUMINATION

Are you Smart Enough to Solve this 300-Year-Old Problem?

A look into the problem that gave birth to modern optimal control theory

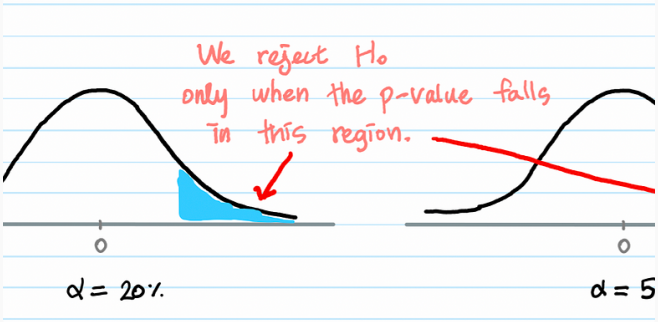
✦ · 5 min read · Aug 1, 2022

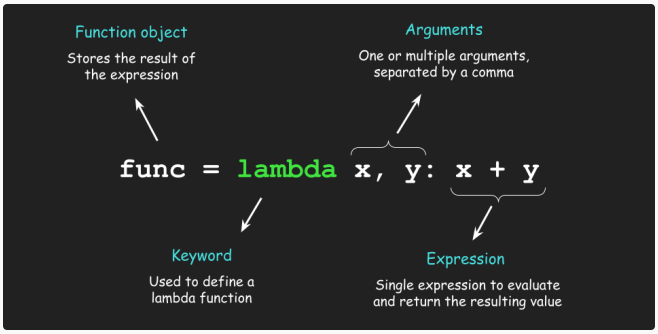
294 1

See all from Zack Fizell

See all from Towards Data Science

Recommended from Medium





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

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

Lists

It's never too late or early to start something

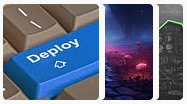
15 stories · 128 saves

Coding & Development

11 stories · 181 saves

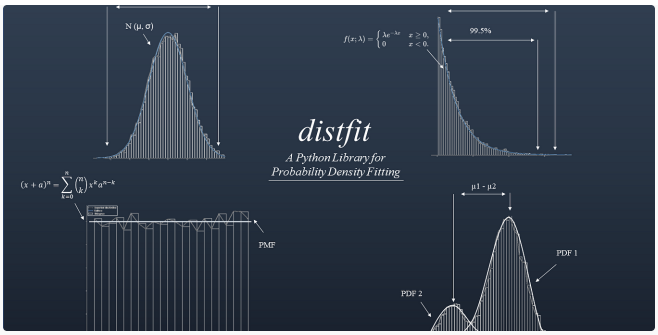
General Coding Knowledge

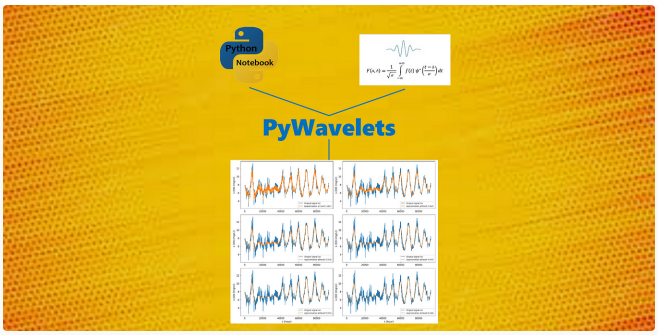
20 stories · 353 saves



Predictive Modeling w/ Python

20 stories · 397 saves





Erdogan Taskesen in Towards Data Science

How to Find the Best Theoretical Distribution for Your Data

Knowing the underlying data distribution is an essential step for data modeling and has...

✦ · 19 min read · Feb 3

1K 10

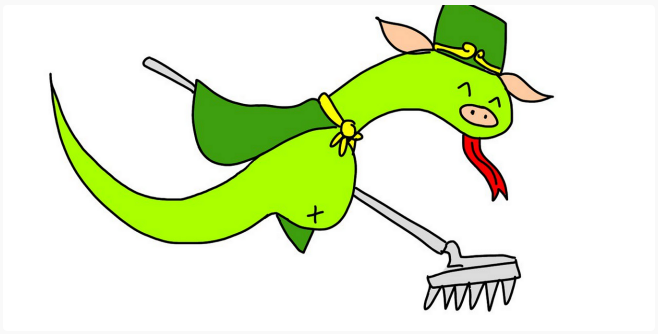
Dr. Shouke Wei

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



Rohit Saroj

How to Turn Your Python Script into an Executable File

.py to .exe quickly!

4 min read · Jul 5

116

Liu Zuo Lin in Python in Plain English

20 Python Recursion Practice Questions

10 Manageable Questions + 10 Less Manageable Questions

✦ · 7 min read · Dec 27, 2022

386 6

See more recommendations