Python 11

Elias Wachmann

2024

Content

1. Uncertainties in Python
2. Test Driven Development (TDD) in Python
3. What is object oriented programming?
4. Python classes

Uncertainties in Python

Uncertainties Package

The `uncertainties` package allows you to perform calculations with `uncertainties` It supports calculations with numbers that have uncertainties and automatically propagates those uncertainties through mathematical operations.

Install the package with `pip`:

```
pip install uncertainties
```

Basic Usage

To use the `uncertainties` package, you need to import the `ufloat` function, which creates numbers with uncertainties:

```
1 from uncertainties import ufloat
2
3 x = ufloat(2.0, 0.1)  # 2.0 +/- 0.1
4 y = ufloat(1.0, 0.2)  # 1.0 +/- 0.2
```

Propagation of Uncertainties

When you perform operations with these numbers, the uncertainties are automatically propagated using linear error propagation

This is great, because you don't have to manually calculate the uncertainties of the results anymore. Just use the `nominal_value` and `std_dev` functions to get the result and its uncertainty:

Propagation of Uncertainties

```python
#...
z = x + y
print(z)  # 3.00 +/- 0.223606797749979
print(z.nominal_value)  # 3.0
print(z.std_dev)  # 0.223606797749979
print(z.n)  # 3.0
print(z.s)  # 0.223606797749979
```

Mathematical Functions

The `uncertainties` package supports many mathematical functions, including trigonometric, exponential, and logarithmic functions:

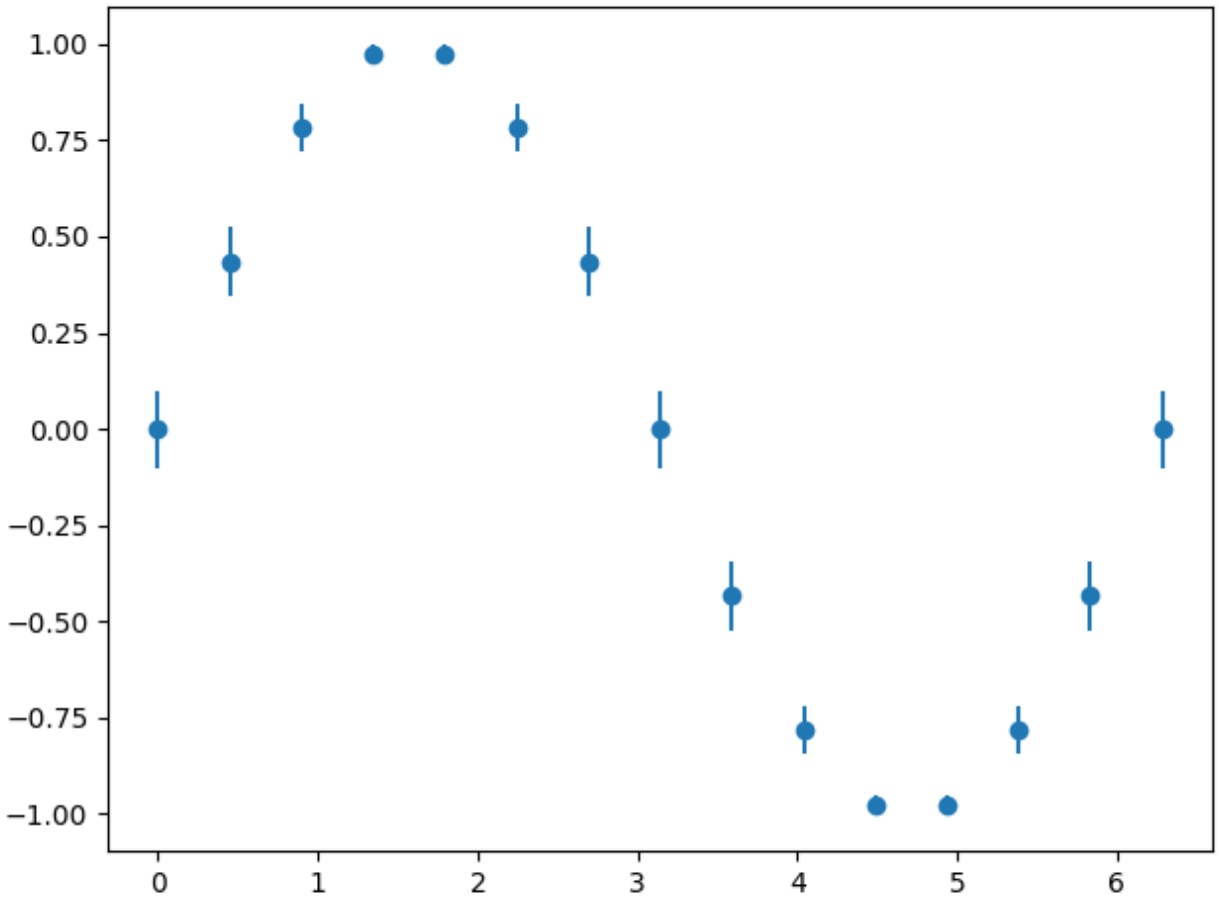
```
1 from uncertainties.umath import sin, cos
2 from uncertainties import ufloat
3
4 angle = ufloat(0.5, 0.05)  # 0.5 +/- 0.05
5 sine_value = sin(angle)
6 cosine_value = cos(angle)
7 print(sine_value)  # 0.48 +/- 0.04
8 print(cosine_value)  # 0.878 +/- 0.024
```

Plotting with Uncertainties

You can also plot data with uncertainties using the `matplotlib` library:

```
1  import matplotlib.pyplot as plt
2  from uncertainties import unumpy as unp
3  import numpy as np
4
5
6  x = np.linspace(0, 2 * np.pi, 15)
7  xU = unp.uarray(x, 0.1)
8  y = unp.sin(xU)
9  plt.errorbar(x, unp.nominal_values(y), yerr=
       unp.std_devs(y), fmt='o')
10 plt.show()
```

Plotting with Uncertainties



Example: Beer-Lambert Law

The Beer-Lambert law relates the absorption of light to the properties of the material it passes through. It is given by:

$$I(x) = I_0 e^{-\alpha x}$$

where $I(x)$ is the intensity of the light after passing through a material of thickness x, I_0 is the initial intensity, and α is the absorption coefficient.

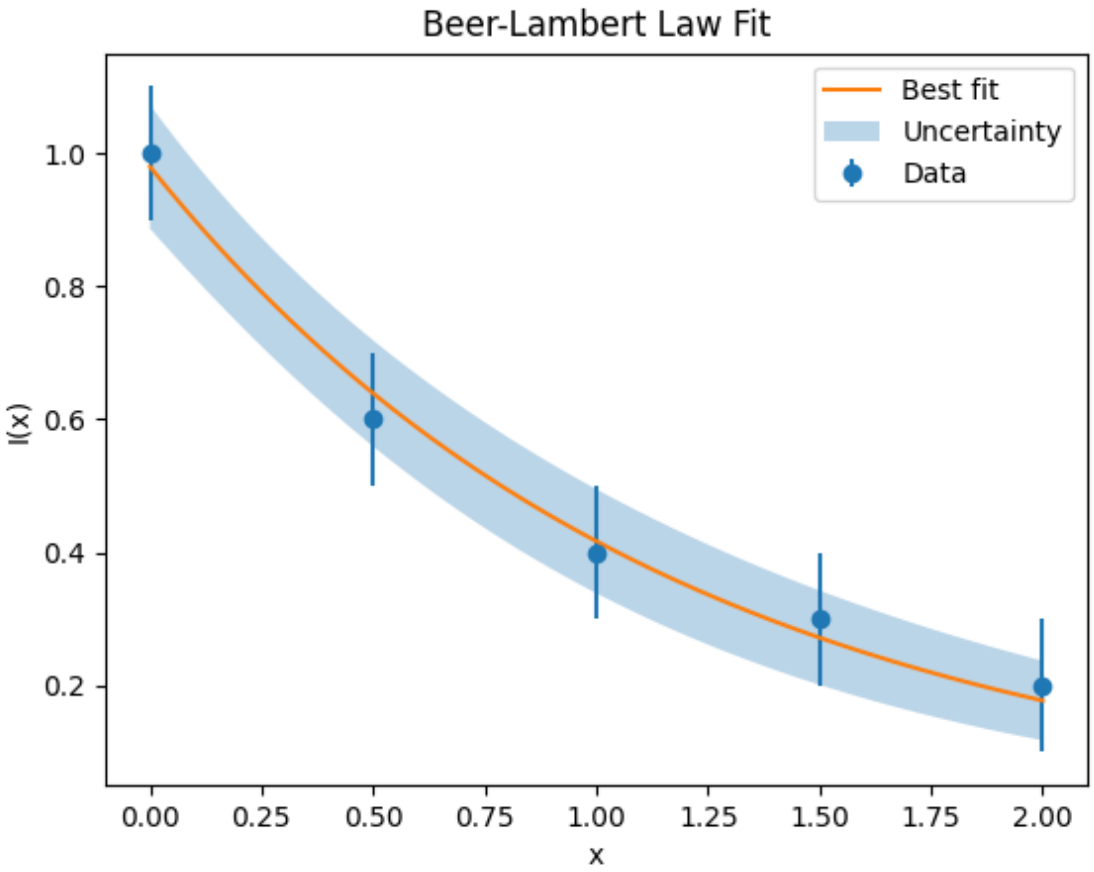
Example: Beer-Lambert Law

Suppose we have the following measurements:

x	$I(x)$	$\Delta I(x)$
0.0	1.0	0.1
0.5	0.6	0.1
1.0	0.4	0.1
1.5	0.3	0.1
2.0	0.2	0.1

and we want to determine the absorption coefficient α using a fit to the Beer-Lambert law.

Example: Beer-Lambert Law



Test Driven Development (TDD) in Python

Introduction to TDD

Test Driven Development (TDD) is a software development methodology in which tests are written before the code that needs to be tested.

- Write a test for the new functionality.
- Write the minimum amount of code to make the test pass.
- Refactor the code while ensuring that all tests still pass.

Benefits of TDD

- Ensures code quality and correctness.
- Facilitates better design and architecture.
- Provides documentation for the code.
- Helps in detecting bugs early.

TDD Cycle

The TDD cycle consists of the following steps:

1. **Red:** Write a test that fails.
2. **Green:** Write the minimal code to pass the test.
3. **Refactor:** Improve the code while ensuring tests pass.

Example: Simple Calculator

Let's consider a simple calculator as an example. We want to implement an `add` function.

```
1 def add(a, b):
2     # Add two numbers and return the result
```

Let us write a test for this function ...

Writing Tests with Pytest

Pytest is a testing framework for Python that makes it easy to write simple and scalable test cases. Install pytest using pip:

```
pip install pytest
```

We specify the test functions using the `test_` prefix. They are automatically detected and run by Pytest.

Pytest Example

Here is an example of a test for the `add` function using Pytest:

```
1 from calculator import add
2
3 def test_add():
4     assert add(1, 2) == 3
5     assert add(-1, 1) == 0
6     assert add(-1, -1) == -2
```

Place this test in a file prefixed with `test_` (e.g., `test_calculator.py`) into the same directory as the code.

Running Tests

To run the tests, simply execute the following command in the terminal:

```
pytest
```

Pytest will automatically discover and run all the test files that match the pattern `test_*.py`.

Refactoring

After making the test pass, you can refactor the code to improve its quality. Ensure that all tests still pass after refactoring.

```
1 def add(a, b):
2     return a + b
3
4 def subtract(a, b):
5     return a - b
```

Refactoring

```
1  from calculator_refactored import add,
       subtract
2
3  def test_add():
4      assert add(1, 2) == 3
5      assert add(-1, 1) == 0
6      assert add(-1, -1) == -2
7
8  def test_subtract():
9      assert subtract(2, 1) == 1
10     assert subtract(1, 1) == 0
11     assert subtract(0, 1) == -1
```

Best Practices

- Write small, focused tests.
- Use descriptive test names.
- Keep tests independent.
- Regularly run all tests.
- Refactor tests along with the code.

Conclusion

Test Driven Development (TDD) helps in ensuring **code quality**, detecting bugs early, and improving the design and architecture of the code.
Pytest is a powerful and easy-to-use testing framework for implementing TDD in Python.
Start with small steps, and iteratively build a robust suite of tests alongside your code.

What is object oriented programming?

Intro to object oriented programming

How do we classify things in the real world?

We group them by their properties and their behavior.

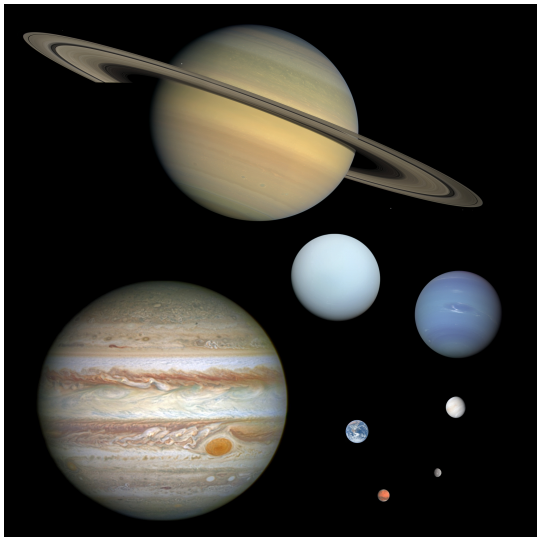
An **object** like a planet can be described by its properties like mass, radius, position, velocity, etc.

We can also describe its behavior, like how it moves around the sun.

What are classes?

Classes are a way to abstract real world things into categories.

Not all planets are the same, but they can be described by the same properties and behavior.
We can use a **class** that describes the properties and behavior of a planet (an **object**).



https://upload.wikimedia.org/wikipedia/commons/c/cf/Planet_collage_to_scale.jpg

Describe objects within a class

How can we distinguish between different planets now?

Earth: m = 5.97×10^{24} kg, r = 6371 km, ...

Mars: m = 6.41×10^{23} kg, r = 3389.5 km, ...

Jupiter: m = 1.90×10^{27} kg, r = 69911 km, ...

Python classes

Let's define a Planet class

In python we can define a class like this:

```
1 class Planet():
2     def __init__(self, name_in, radius_in,
    mass_in):
3         self.name = name_in
4         self.radius = radius_in
5         self.mass = mass_in
```

The **class**-keyword tells python that we want to define a class.

What is the __init__ function?

The **__init__** function is called when we create a new object of the class.

Objects are therefore concrete **instances** of a class.

In our example the Planet class takes the arguments **mass** and **radius** and stores them as **attributes** of the object.

How can we access these attributes and what is the **self**-keyword?

Idea behing self keyword

The **self** keyword is a reference to the object itself.

What does this mean?

2 options:

- Just accept it as is and use it.
- Understand what it does.

When we create an object of a class, we can access its attributes and methods using the **self** keyword inside the class.

Pythons self keyword

The **self** keyword is a reference to the object itself.

```
1     def __init__(self, name_in, radius_in,
    mass_in):
2         self.name = name_in
3         self.radius = radius_in
4         self.mass = mass_in
```

This means: If you want to create an attribute of a class, you simply write `self.variable_name`

self also has to be passed in as the first argument to every function defined inside the class (for context)!

Methods – class functions

Methods are functions inside a class. They can only be accessed by objects of that class and **self** is their first argument.

```
1 class Planet():
2     def __init__(self, name_in, radius_in,
    mass_in):
3         self.name = name_in
4         self.radius = radius_in
5         self.mass = mass_in
6 
7     def getData(self):
8         return {"name": self.name, "radius":
    self.radius, "mass": self.mass}
```

Let's create a Planet

Finally let's create a planet object:

```
earth = Planet("Earth", 6371, 5.972e24)
mars = Planet("Mars", 3389, 6.39e23)
```

Just pass in the initial arguments just like you would for functions.

```
earth_data = earth.getData()
print(earth_data["name"])  # Earth
```

We can now call our defined class method on objects.

Modify objects

Certainly a method can modify attributes of an object like in the following example the name.

```
    def changeName(self, newname):
        self.name = newname
```

```
earth.changeName("Earth2.0")
earth_data = earth.getData()
print(earth_data["name"])  # Earth2.0
```

This only changes the name of the ‚earth'-object while the ‚mars'-object is not affected.