

## FROM DATA TO CURVES AND VISUAL REPRESENTATIONS

There is always a python library for it!

---

Manos Kirtas (*eakirtas@csd.auth.gr*)

Researcher & PhD Candidate

November 14, 2023

Introduction

Virtual Environments

argparse & logging

numpy & matplotlib

scipy & scikit-learn

# INTRODUCTION

---

# HOW TO PRESENT MY WORK?

- Doing the job right is not always enough.
- Most of the time you have also to present it well.



Figure: Bad visualization example <sup>1</sup>

Presenting a work with an effective way needs both experience and tools. In this tutorial we going to learn some tools that will allow us to present our work in an effective manner.

<sup>1</sup>Source: CSG Solution Blog

Python has a massive ecosystem, especially for machine learning.  
For example:

1. Numerical & Data engineering: Numpy, Pandas
2. Statistics & Machine Learning: SciPy, scikit-learn, SymPy, Jax
3. Deep Learning: PyTorch, TensorFlow, Haiku
4. Data visualization: Matplotlib, Seaborn, Plotly, tabulate, gradio
5. Machine Learning pipeline: Ray, wandb, tensorboard


# THERE IS ALWAYS A PYTHON LIBRARY FOR IT!

## python-chess: a chess library for Python [↗](#)

Test [passing](#) pypi package [1.10.0](#) docs [passing](#) chat [on gitter](#)

### Introduction [↗](#)

python-chess is a chess library for Python, with move generation, move validation, and support for common formats. This is the Scholar's mate in python-chess:



build [passing](#) pypi [v2.5.2](#) license [LGPL](#) python [3](#) commits since 2.1.2 [1.9k](#) code style [black](#)

[Pygame](#) is a free and open-source cross-platform library for the development of multimedia applications like video games using Python. It uses the [Simple DirectMedia Layer library](#) and several other popular libraries to abstract the most common functions, making writing these programs a more intuitive task.



pypi [v1.0.0](#) test-pypi [v1.0.0](#) Downloads [23k](#) license [GPL-3.0](#) all contributors [6](#)

# VIRTUAL ENVIRONMENTS

---

# WHAT IS A VIRTUAL ENVIRONMENT?

## What is a virtual environment?

- Creating virtual environments is a way to create isolated environments for your Python projects, helping you manage dependencies and avoid conflicts between different projects.
- A virtual environment is a self-contained directory that contains its own Python interpreter and can have its own installed packages and modules.



## Why use a Virtual Environments?

- **Isolation:** Keeps dependencies for different projects separate, preventing conflicts.
- **Dependency Management:** Easier management of project-specific dependencies.
- **Portability:** Makes it easy to share and reproduce the exact environment of a project.

## CREATE AND ACTIVATE VIRTUAL ENVIRONMENT

- Using venv (built-in module in Python 3.3 and later):

```
$ python -m venv <venv_name>
```

- Activate the virtual environment

- On Windows:

```
$ <venv_name>\Scripts\activate
```

- On Unix or MacOS:

```
$ source <venv_name>/bin/activate
```

- Deactivating the Virtual Environment:

```
$ deactivate
```

## INSTALL A PYTHON LIBRARY

- We can install a python package in a Virtual Environment using pip:

```
$ pip install <package_name>
```

- We can list all the installed packages using:

```
$ pip list
```

- If you want to share your project and its dependencies you just need to provide a txt file

```
$ pip freeze > requirements.txt
```

- In turn someone else can install the project dependencies as:

```
$ pip install -r requirements.txt
```

# INSTALL YOUR PROJECT AS PYTHON LIBRARY

1. We need to create a **setup.py** file. This file includes:

- The name of the project
- Its version
- Text that describes the project
- Information about authors
- Requirements
- etc

2. After that we can install our project as library using:

```
$ pip install -e <project_path>
```

In such case we call modules from our project as following:

```
from my_project.this.tool import awesome_tool
```

# ARGPARSE & LOGGING

---

## ARGPARSE BUILT-IN MODULE

The argparse module makes it easy to write user-friendly **command-line interfaces**. The program defines what arguments it requires, and argparse will figure out how to parse those out of sys.argv. The argparse module also automatically generates help and usage messages.

```
parser = argparse.ArgumentParser(  
    prog='ProgramName',  
    description='What the program does',  
    epilog='Text at the bottom of help')  
  
# positional argument  
parser.add_argument('filename')  
# option that takes a value  
parser.add_argument('-c', '--count')  
# on/off flag  
parser.add_argument('-v', '--verbose',  
                    action='store_true')
```

This module defines functions and classes which implement a flexible event logging system for applications and libraries.

The key benefit of having the logging API provided by a standard library module is that all Python modules can participate in logging, so your application log can include your own messages integrated with messages from third-party modules.

```
logging.info('Welcome to AR.AN tutorial')  
logging.debug('Welcome to AR.AN tutorial')  
logging.warning('Welcome to AR.AN tutorial')  
logging.error('Welcome to AR.AN tutorial')  
logging.critical('Welcome to AR.AN tutorial')
```

We want to test some algorithms that we've learned about fitting curves on data. To do that we want to generate data for 3 functions:

.

$$f(x) = \frac{x}{2} \sin\left(\frac{x}{3} + \frac{\pi}{2}\right) - 0.5$$

.

$$g(x) = \tanh\left(\frac{x\pi}{2}\right) + \cos(x)$$

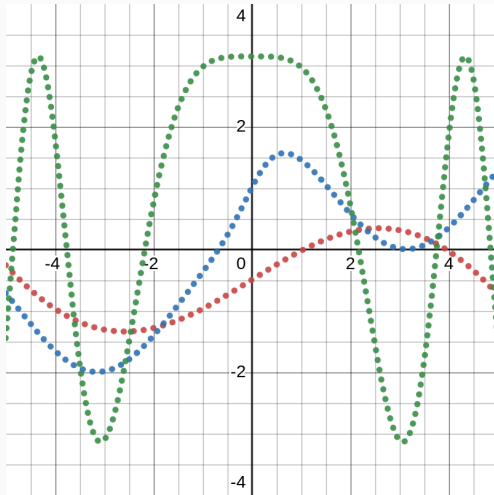
.

$$h(x) = \cos\left(\frac{x^2}{3}\right) \pi$$



## GENERATE DATA

What libraries needed to generate points that follows the aforementioned values? How we can visualize them in python?



**Figure:** Points generated by the aforementioned functions

## ADD THE APPROPRIATE ARGUMENT

```
parser.add_argument(  
    '--generate-at',  
    help='generates data at a file',  
    type=str,  
    default='./data/generated_data.npz',  
)  
parser.add_argument('--num_points',  
                    help='the number of point',  
                    type=int,  
                    default=50)
```



Figure: Code is available at: [aran\\_tutorial\\_cf/v1/run.py](#)

# NUMPY & MATPLOTLIB

---

# WHAT IS NUMPY?

NumPy, which stands for Numerical Python, is a powerful library in Python used for numerical and mathematical operations. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy is particularly useful for tasks involving numerical computations in fields such as linear algebra, statistics, calculus, and more.

```
$ pip install numpy
```

## Key features of NumPy include:

- **Array Objects:** NumPy provides a multidimensional array object called `numpy.ndarray`, which is a fast and flexible container for large datasets.
- **Mathematical Functions:** NumPy includes a wide range of mathematical functions that operate on arrays, making it easy to perform operations on entire datasets without the need for explicit loops.
- **Broadcasting:** NumPy supports broadcasting, a powerful mechanism that allows operations between arrays of different shapes and sizes to be performed

Key features of NumPy include:

- **Linear Algebra Operations:** NumPy provides a set of linear algebra operations, including matrix multiplication, eigenvalue decomposition, and more.
- **Random Number Generation:** NumPy includes functions for random number generation, which is useful in various statistical and scientific simulations.
- **Integration with Other Libraries:** NumPy is often used in conjunction with other scientific computing libraries in the Python ecosystem, such as SciPy, Matplotlib, and scikit-learn.

```
numpy.linspace(start, stop, num=50, endpoint=True,  
               retstep=False, dtype=None, axis=0)
```

- Return evenly spaced numbers over a specified interval.
- Returns num evenly spaced samples, calculated over the interval [start, stop].
- The endpoint of the interval can optionally be excluded.



Save an array to a text file.

```
numpy.savetxt(fname, X, fmt='%.18e', delimiter=' ',  
              newline='\n', header='',  
              footer='', comments='# ',  
              encoding=None)
```

Load from a text file

```
numpy.loadtxt(fname, dtype=<class 'float'>,  
              comments='#', delimiter=None, *)
```

```
def f(x):  
    return x / 2.0 * np.sin(x / 3.0 + np.pi) - 0.5  
def g(x):  
    return np.tanh((x * np.pi) / 2.0) + np.cos(x)  
def h(x):  
    return np.cos(np.power(x, 2) / 2.0) * np.pi  
  
x = np.linspace(-5, 5, num=1000)  
  
f_x = f(x)  
g_x = g(x)  
h_x = h(x)
```



**Figure:** Code is available at: `aran_tutorial_cf/v2`

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

1. Create publication quality plots.
2. Make interactive figures that can zoom, pan, update.
3. Export to many file formats.
4. Embed in JupyterLab and Graphical User Interfaces.
5. Use a rich array of third-party packages built on Matplotlib.

```
$ pip install matplotlib
```

1) A scatter plot of y vs. x with varying marker size and/or color.

```
matplotlib.pyplot.scatter(x, y, s=None, c=None,  
                           marker=None, cmap=None,  
                           norm=None, vmin=None, vmax=None,  
                           alpha=None, linewidths=None,  
                           **kwargs)
```

2) Plot y versus x as lines and/or markers.

```
matplotlib.pyplot.plot(*args, scalex=True,  
                       scaley=True, data=None,  
                       **kwargs)
```



**Figure:** Code is available at: [aran\\_tutorial\\_cf/v2/visualize.py](#)

Return a sample (or samples) from the “standard normal” distribution.

```
np.random.randn(d0, d1, ..., dn)
```

We can manipulate the standard normal distribution by:

```
sigma * np.random.randn(...) + mu
```

However, plotting noise with line plot is not effective. For distributions plots we use histograms:

```
matplotlib.pyplot.hist(x, bins=None, ...)
```

## Distributions

`beta(a, b[, size])`

Draw samples from a Beta distribution.

`binomial(n, p[, size])`

Draw samples from a binomial distribution.

`chisquare(df[, size])`

Draw samples from a chi-square distribution.

`dirichlet(alpha[, size])`

Draw samples from the Dirichlet distribution.

`exponential([scale, size])`

Draw samples from an exponential distribution.

`f(dfnum, dfden[, size])`

Draw samples from an F distribution.

`gamma(shape[, scale, size])`

Draw samples from a Gamma distribution.

`geometric(p[, size])`

Draw samples from the geometric distribution.

`gumbel([loc, scale, size])`

Draw samples from a Gumbel distribution.

`hypergeometric(ngood, nbad, nsample[, size])`

Draw samples from a Hypergeometric distribution.

`laplace([loc, scale, size])`

Draw samples from the Laplace or double exponential distribution (location (or mean) and scale (decay)).





**Figure:** Code is available at: [aran\\_tutorial\\_cf/v2/noise\\_visualization.py](https://github.com/aran-tutorial_cf/v2/noise_visualization.py)

# SCIPY & SCIKIT-LEARN

---

- **Scientific Computing Tools:** SciPy is an open-source library for mathematics, science, and engineering.
- **Modules for Diverse Tasks:** SciPy includes various modules for optimization, integration, interpolation, eigenvalue problems, signal and image processing, statistical functions, and more.
- **Integration with NumPy:** SciPy seamlessly integrates with NumPy, a fundamental package for numerical computing in Python.
- **Optimization and Root Finding:** The library offers powerful tools for optimization and root finding, allowing users to solve complex mathematical problems, minimize or maximize functions, and find roots of equations.

```
$ pip install scipy
```

- **Machine Learning Algorithms:** Scikit-learn is a popular open-source machine learning library for Python. It provides simple and efficient tools for data analysis and modeling, including a wide array of machine learning algorithms such as classification, regression, clustering, and dimensionality reduction.
- **Integration with NumPy and SciPy:** Scikit-learn seamlessly integrates with other scientific computing libraries like NumPy and SciPy. This integration allows for efficient manipulation of data arrays and easy incorporation of machine learning models into broader scientific workflows, making it a versatile tool for researchers and practitioners in various fields.

```
$ pip install scikit-learn
```

- Curve fitting is a type of optimization that finds an optimal set of parameters for a defined function that best fits a given set of observations.
- Unlike supervised learning, curve fitting requires that you define the function that maps examples of inputs to outputs.
- The mapping function, also called the basis function can have any form you like, including a straight line (linear regression), a curved line (polynomial regression), and much more.
- This provides the flexibility and control to define the form of the curve, where an optimization process is used to find the specific optimal parameters of the function.

Use non-linear least squares to fit a function,  $f$ , to data.

```
scipy.optimize.curve_fit(f, xdata, ydata, ...)
```

For reliable results, the model func should not be overparametrized; redundant parameters can cause unreliable covariance matrices and, in some cases, poorer quality fits. As a quick check of whether the model may be overparameterized, calculate the condition number of the covariance matrix



**Figure:** Code is available at: [aran\\_tutorial\\_cf/v3/curve\\_fit.py](https://github.com/aran-tutorial_cf/v3/curve_fit.py)