# CSE/ECE 848
# Introduction to
# Evolutionary Computation

## Module 1 - Lecture 4 - Part 3

# Octave Code for Some Methods

Wolfgang Banzhaf, CSE
John R. Koza Chair in Genetic Programming

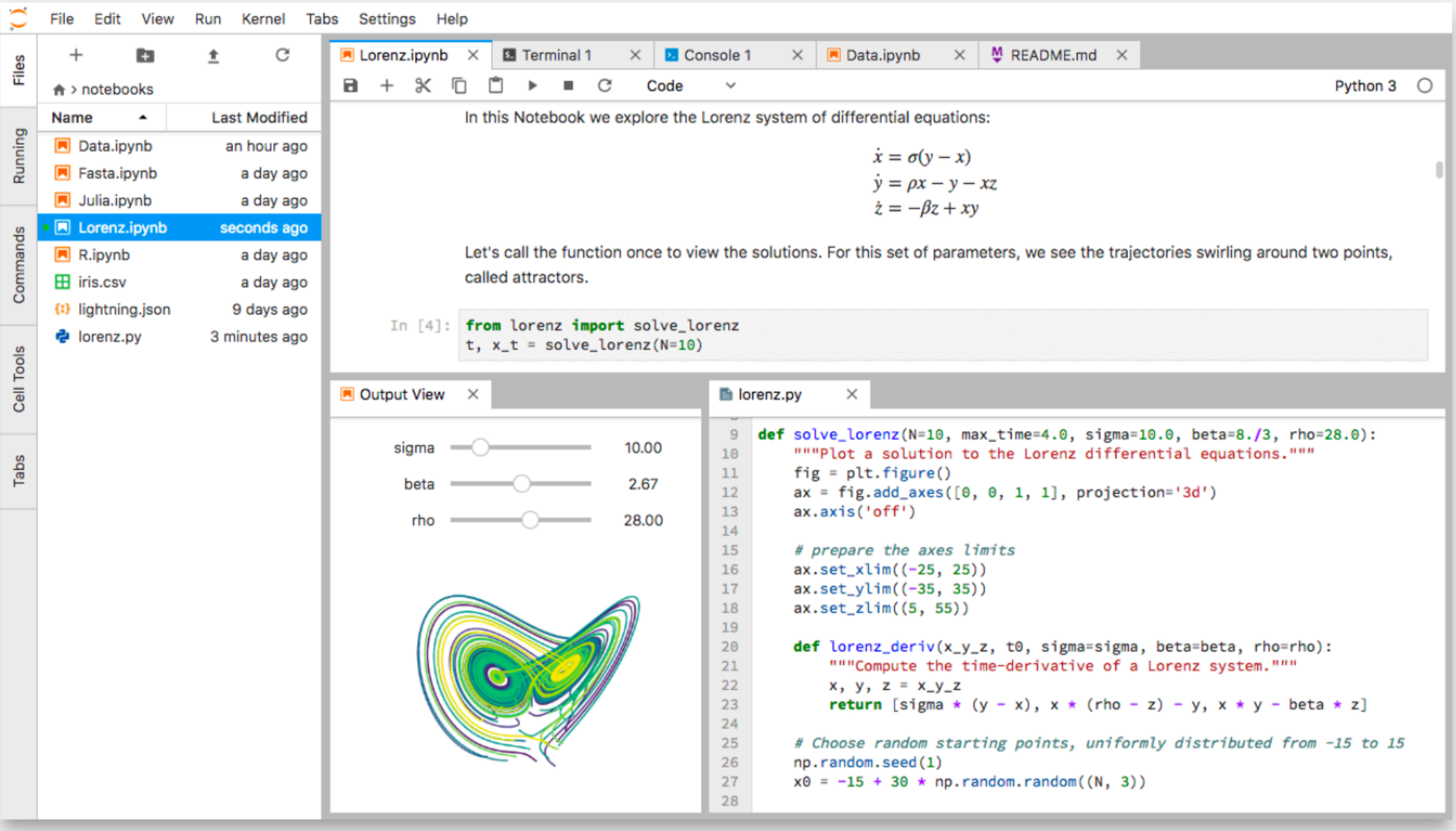
# Jupyter Lab / Jupyter Notebook

- Our way of exploring different algorithms
- Base kernel is Python, but other languages are installable
- Go to https://jupyterlab.readthedocs.io/en/stable/index.html
- Also a lot of information on https://jupyter.org/

- Makes use of the notebook principle first used with Mathematica: Cells that can contain code that can be executed

# JupyterLab Documentation

JupyterLab is the next-generation web-based user interface for Project Jupyter. Try it on Binder.
JupyterLab follows the Jupyter Community Guides.



| Name | Last Modified |
|---|---|
| Data.ipynb | an hour ago |
| Fasta.ipynb | a day ago |
| Julia.ipynb | a day ago |
| Lorenz.ipynb | seconds ago |
| R.ipynb | a day ago |
| iris.csv | a day ago |
| lightning.json | 9 days ago |
| lorenz.py | 3 minutes ago |

In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y-x)\\ \dot{y} &= \rho x - y - xz\\ \dot{z} &= -\beta z + xy\end{aligned}$$

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

```
In [4]: from lorenz import solve_lorenz
        t, x_t = solve_lorenz(N=10)
```

sigma 10.00
beta 2.67
rho 28.00

```
def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):
    """Plot a solution to the Lorenz differential equations."""
    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')

    # prepare the axes limits
    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
        """Compute the time-derivative of a Lorenz system."""
        x, y, z = x_y_z
        return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

    # Choose random starting points, uniformly distributed from -15 to 15
    np.random.seed(1)
    x0 = -15 + 30 * np.random.random((N, 3))

```

# Jupyter Lab / Jupyter Notebook

- You can think about algorithms and observe their behavior in one place
- You need to install the Octave kernel (a public domain version of MATLAB)
- Kernels for many languages are available, e.g., R, Julia, etc, see their GitHub page: https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

- We'll discuss a few very simple search algorithms
    - Random Search
    - Hill Climbing
    - Simulated Annealing