SECTION 21: GUIs - 45 minutes, 7 parts

7/7 Example of what a Widget can do!

- **-> a custom-widget example**
  - -> running the notebook directly
- **-> this requires**
  - -> numpy
  - -> maptplotlib
  - -> scipy
- **-> how you would connect the widgets to external libraries**
- **-> these can be installed using conda install**
  - -> or pip install
  - -> there is another bootcamp called the Python for Data Science and Machine Learning Bootcamp
- **-> Lorentz ODEs**
  - -> this is a series of 3D ODEs, which we want to plot
  - -> matplotlib inline
- **-> then he imports modules**
  - -> there are multiple of these
  - -> some of them are for visualisation
  - -> matplotlib
    - -> plotting for Python
  - -> we are visualising the ODE equations (three of them)
- **-> he has defined a plotting function for these ODEs**
  - -> they are three ODEs <- differential, in x, y and z
  - -> creating a figure
  - -> setting the limits of the figure
  - -> computing the time derivative of the system
  - -> choosing random starting points
  - -> solving the equations
  - -> choosing different colours

GUI Example

PIERIAN DATA

Custom Widget

Exploring the Lorenz System of Differential Equations

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

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

This is one of the classic systems in non-linear differential equations. It exhibits a range of different behaviors as the parameters (σ, β, ρ) are varied.

Imports

First, we import the needed things from IPython, NumPy, Matplotlib and SciPy. Check out the class Python for Data Science and Machine Learning Bootcamp if you're interested in learning more about this part of Python!

In [1]:
```
%matplotlib inline
```

In [ ]:
```
from ipywidgets import interact, interactive
from IPython.display import clear_output, display, HTML
```

In [ ]:
```
import numpy as np
from scipy import integrate

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
```

Computing the trajectories and plotting the result

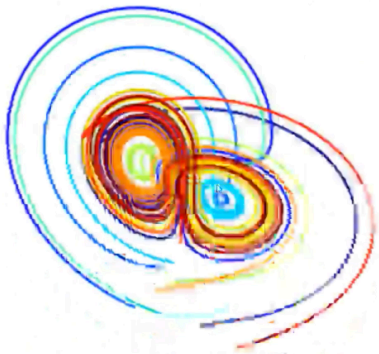
We define a function that can integrate the differential equations numerically and then plot the solutions. This function has arguments that control the parameters of the differential equation (σ, β, ρ), the numerical integration (`N`, `max_time`) and the visualization (`angle`).

In [ ]:
```
def solve_lorenz(N=10, angle=0.0, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):

    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))

    # Solve for the trajectories
    t = np.linspace(0, max_time, int(250*max_time))
    x_t = np.asarray([integrate.odeint(lorenz_deriv, x0i, t)
```

- -> plotting everything per line
- -> this shows us what is possible with Python

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.

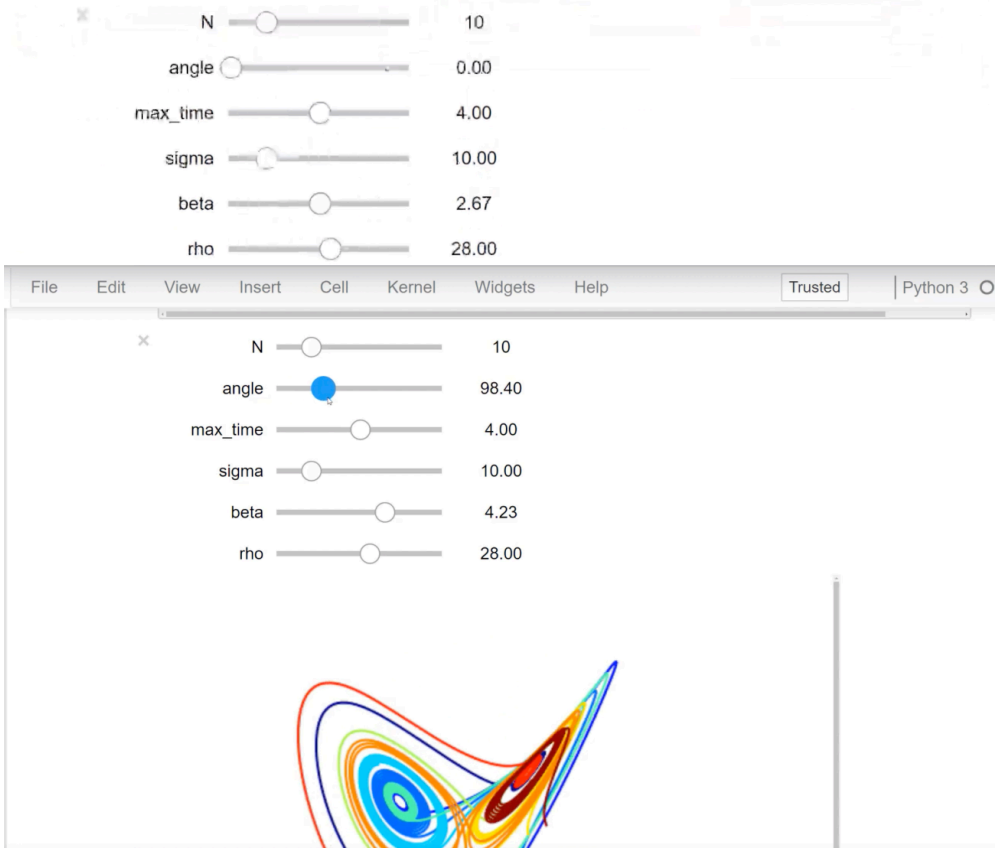
```
In [5]: t, x_t = solve_lorenz(angle=0, N=10)
```



- **-> he runs this cell**
  - -> this returns t and x(t)
  - -> this solves the differential equations
  - -> he runs the function and it plots the different numbers
  - -> he then runs it with the interactive method
  - -> this allows us to change different parameters
    - -> e.g the angle which we are looking at the plot with
    - -> he changes each of the different parameters in the equation using the slider, to see how the graph changes
    - -> the slider parameters can be used for this
  - -> this allows us to connect the function to whatever parameters we want

Using IPython's `interactive` function, we can explore how the trajectories behave as we change the various parameters.

```
In [6]: w = interactive(solve_lorenz, angle=(0.,360.), N=(0,50), sigma=(0.0,50.0), rho=(0
        display(w);
```

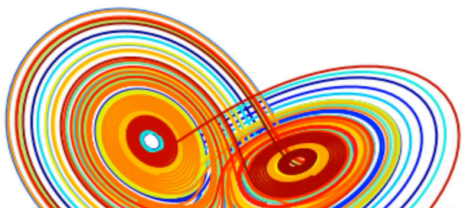


- **-> this can be extended to histograms**
  - -> advanced parameters
  - -> re-running these allows us to see the different angles we had
  - -> new histograms to plot the different elements

- **-> this was**
  - -> GUIs
  - -> ipywidgets
  - -> once we have a function which returns something, we can give user interactions by linking them to widgets
  - -> there are other functions for user interactions
  - -> buttons, checkboxes, text boxes, expandable text boxes, html, widgets

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

N 43
angle 148.20
max_time 6.90
sigma 10.00
beta 4.23
rho 28.00



Using IPython's `interactive` function, we can explore how the trajectories behave as we change the various parameters.

In [6]:
```
w = interactive(solve_lorenz, angle=(0.,360.), N=(0,50), sigma=(0.0,50.0), rho=(0
display(w);
```

N 43
angle 148.20
max_time 7.80
sigma 32.90
beta 2.83
rho 22.90

The object returned by `interactive` is a `Widget` object and it has attributes that contain the current result and arguments:

In [7]:
```
t, x_t = w.result
```

In [8]:
```
w.kwargs
```

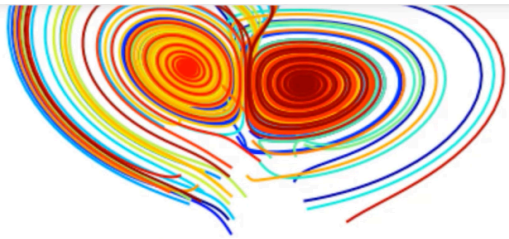
Out[8]:
```
{'N': 43,
 'angle': 148.2,
 'beta': 2.83333,
 'max_time': 7.8,
 'rho': 22.9,
 'sigma': 32.9}
```

After interacting with the system, we can take the result and perform further computations. In this case, we compute the average positions in x, y and z.

In [9]:
```
xyz_avg = x_t.mean(axis=1)
```

In [10]:
```
xyz_avg.shape
```

Out[10]:
```
(43, 3)
```



The object returned by `interactive` is a `Widget` object and it has attributes that contain the current result and arguments:

In [7]:
```
t, x_t = w.result
```

In [8]:
```
w.kwargs
```

Out[8]:
```
{'N': 43,
 'angle': 148.2,
 'beta': 2.83333,
 'max_time': 7.8,
```

In [8]:
```
w.kwargs
```

Out[8]:
```
{'N': 43,
 'angle': 148.2,
 'beta': 2.83333,
 'max_time': 7.8,
 'rho': 22.9,
 'sigma': 32.9}
```

After interacting with the system, we can take the result and perform further computations. In this case, we compute the average positions in x, y and z.

In [ ]:
```
xyz_avg = x_t.mean(axis=1)
```

In [ ]:
```
xyz_avg.shape
```

Creating histograms of the average positions (across different trajectories) show that on average the trajectories swirl about the attractors.

NOTE: These will look different from the lecture version if you adjusted any of the sliders in the

In [9]:
```
xyz_avg = x_t.mean(axis=1)
```

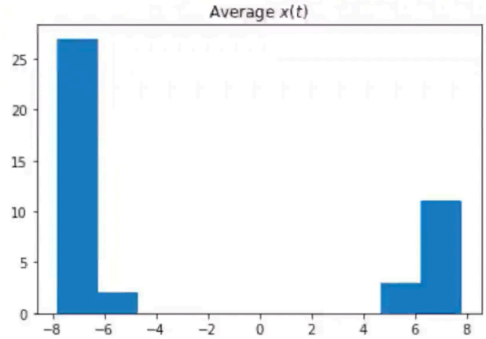
In [10]:
```
xyz_avg.shape
```

Out[10]:
```
(43, 3)
```

Creating histograms of the average positions (across different trajectories) show that on average the trajectories swirl about the attractors.

NOTE: These will look different from the lecture version if you adjusted any of the sliders in the interactive widget and changed the parameters.

In [11]:
```
plt.hist(xyz_avg[:,0])
plt.title('Average $x(t)$');
```



In [ ]:
```
plt.hist(xyz_avg[:,1])
plt.title('Average $y(t)$');
```

In [12]:
```
plt.hist(xyz_avg[:,1])
plt.title('Average $y(t)$');
```