The Lorenz Differential Equations

This is a demo notebook (based on the notebooks from https://jupyter.org/try-jupyter/retro/notebooks/?path=notebooks/Intro.ipynb) which uses Python to demonstrate interactive visualizations and computations around the Lorenz system. It shows off basic Python functionality, including more visualizations, data structures, and scientific computing libraries.

We start to explore the Lorenz system of differential equations:

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

σ, β, ρ are free parameters which we are using for the visualization.

We will can define the actual solver and plotting routine:

In [ ]:

```
import numpy as np
from matplotlib import pyplot as plt
from scipy import integrate

from ipywidgets import interactive, fixed

def solve_lorenz(sigma=10.0, beta=8./3, rho=28.0):
    """Plot a solution to the Lorenz differential equations."""

    max_time = 4.0
    N = 30

    fig = plt.figure(1)
    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)
                      for x0i in x0])
```

```python
    # choose a different color for each trajectory
    colors = plt.cm.viridis(np.linspace(0, 1, N))

    for i in range(N):
        x, y, z = x_t[i,:,:].T
        lines = ax.plot(x, y, z, '-', c=colors[i])
        plt.setp(lines, linewidth=2)
    angle = 104
    ax.view_init(30, angle)
    plt.show()

    return t, x_t
```

To create a proper visualization we can use these commands:

In [ ]:
```python
w=interactive(solve_lorenz,sigma=(0.0,50.0),rho=(0.0,50.0))
w    # display the resulting widget
```

For the default set of parameters, we see the trajectories swirling around two points, called attractors.