Tutorial: Chaotic Systems: Simulation, Learning and Application

Team (#22) Member:

- Gen Mark Veloso Tanno
- Haiyue Yin
- Lei Jiang

Github repository: ¶

[https://github.gatech.edu/hyin62/ChaoticSysSimulation.git (https://github.gatech.edu/hyin62/ChaoticSysSimulation.git)](https://github.gatech.edu/hyin62/ChaoticSysSimulation.git)

Notes:

The tutorial includes both Jupyter Notebook and Matlab Live Script

The Jupyter Notebook includes animated plots as videos which may not work correctly on the github page. Please use the html file in the repo (chaotic_system_simulation.html), or run the notebook locally to see the animations.

- If any of the above does not work on your machine please let us know. We will be glad to demo them for grading purpose. Thank you!

Content

Part 0: Introduction of chaotic system

Part 1: Analysis and Simulation of Chaotic System

Part 2: Learning chaotic system: using neural networks, delay embeddings, and ensemble kalman filtering

Part 3: Chaos in Multiple Pendulums with application of Neural Networks

Part 4: Real Life Applications: Sending Secret Message

The parts about Lorenz system roughly follows the material in Chapter 9.0 of Steven H. Strogats's book Non-linear dynamics and chaos - 2nd Edition.

Part 0: Introduction of chaotic system

In order to analyze chaotics systems, it is very helpful to visualize the chaotic attractors and other fractal objects. The development of the theory of dynamical system has been contributing to the visualization.

Lorenz system, with its beautiful butterfly shape, has been a typical model for analyzing chaotic systems. We can start with analyzing lorenz systems:

Ed Lorenz (1963) derived the lorenz equations by designing a simplified model of convection rools in the atmosphere.

The lorez equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = rx - y - xz$$
$$\dot{z} = xy - bz$$

Here σ , r, b > 0 are paramters. We can also obtain the same equation in laser and dynamo effects.

And here are the dynamics shown by the deterministic system:

When changing parameters, we will obtain irregularly solutions. However, the solutions will always remain in a bounded region of phase space. The trajectories will settle onto a strange attractor. The strange attractor is a fractal, with a fractional dimentsion between 2 and 3.

For a system with parameter σ , r, b > 0, where σ is the Prandtl number, r is the Rayleigh number and b has no name. The equations has the following simple properties:

1) Nonlinearity The quartic terms xy and xz are two nonlinearities.

2) Symmetry The equations remain the same when we change (x, y) to (-x, -y). So the solutions will either be symmetric with themselves or has a symmetric partner.

3) Volumn Contraction Volumns with extremely large size at the beginning with ends up shrinking to limiting size of zero volumn. So are the trajectories in the volumn.

4) Fixed points There are two types of fixed points in the Lorenz system. For parameter with any values, the origin is always a fixed point. When r > 1, a symmetric fixed points $x^* = y^* = \pm\sqrt{b(r-1)}, z^* = r - 1$. They shows the convections rolls on two sides. They will coalesce with the origin in a pitchfork bifircation. We will call them C^+ and C^-.

5) Linear Stability of the origin When omitting the xy and xz terms in the first two equations. We will have

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = rx - y$$
$$\dot{z} = -bz$$

which is:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\sigma & sigma \\ r & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The trace of it is < 0, and determinant is $\sigma(1 - r)$. Determinant is a saddle node when r > 1. Since on z direction, the system is decaying. So the system will have two incoming directions and one outgoing direction. If r < 1, all three directions are incoming directions. The origin will become a sink. And the origin is a stable node in this case.

6) Global Stability of the Origin The origin is globally stable when r < 1, which means that when t goes to infinity, all the trajectories will become very close to origin.

7) Stability of C^+ and C^-. When r > 1, we need to analyze C^+ and C^-. C^+ and C^- are linearly stable for

$$1 < r < r_H = \frac{\sigma(\sigma + b + 3)}{\sigma - b - 1}$$

Part 1: Analysis and Simulation of chaotic Lorenz system

In this part, we will simulate the lorenz system as an example to understand the properties, including the strange attractor and the chaotic motion on it.

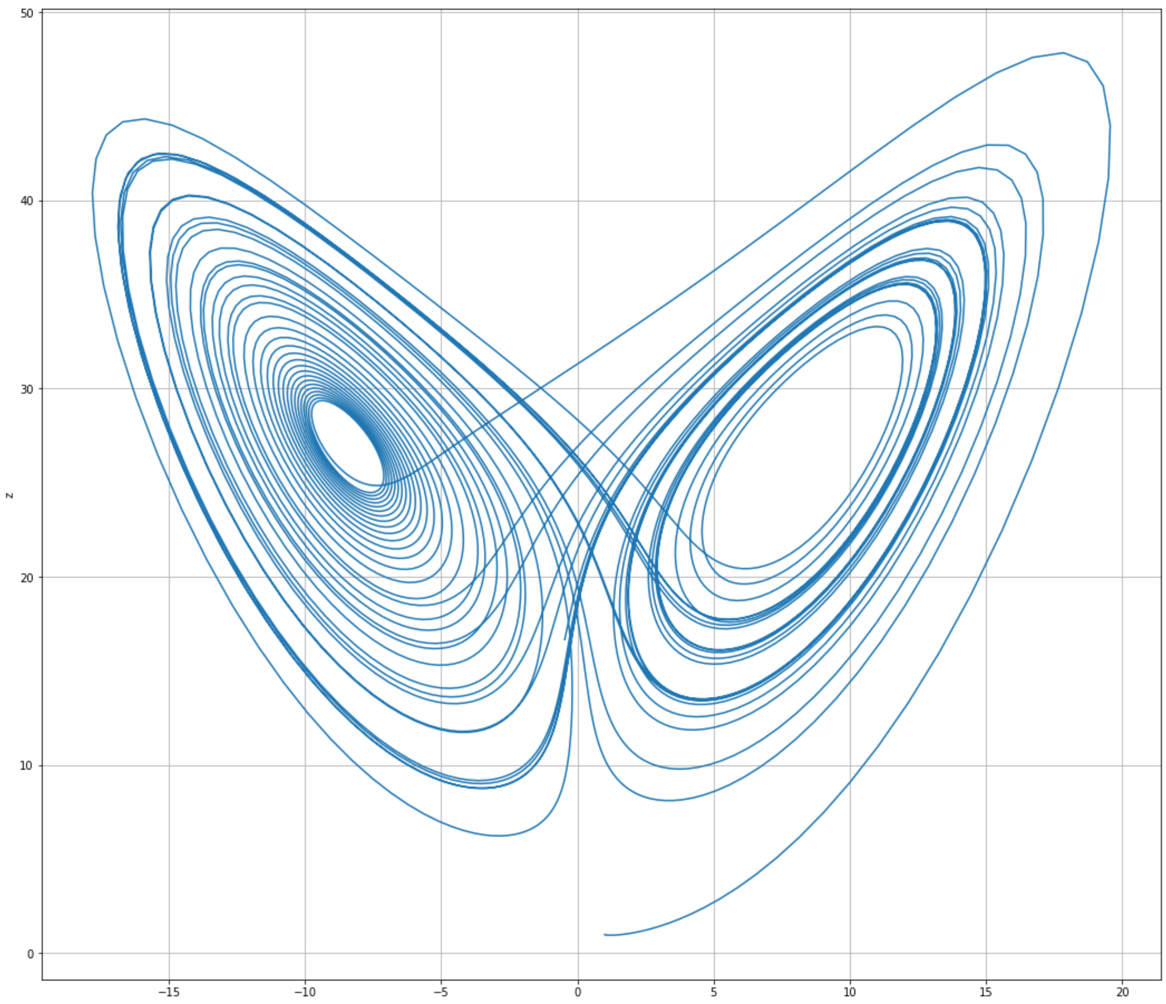
Using the particular case: initial position [1.0, 1.0, 1.0] and parameter σ = 10, β = 8/3 and r = 28. We know that the valud of r is just past the Hopf bifurcation value r_H = σ(σ + b + 3)/(σ - b - 1) = 24.74.

Under such condition, a beautiful butterfly pattern will appear. (See figure below) This demonstrate the strange attractor (that in fact has a fractal geometric shape).

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import odeint
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation, rc
from math import sqrt
```

In [2]:
```
rho = 28.0
sigma = 10.0
beta = 8.0 / 3.0

def lorenz_deriv(state, t0, sigma=sigma, beta=beta, rho=rho):
    x, y, z = statecould
    return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]
```

In [17]:
```
state0 = [1.0, 1.0, 1.0]
t = np.arange(0.0, 40.0, 0.01)

states = odeint(lorenz_deriv, state0, t)

fig, ax = plt.subplots(figsize=(18, 16))
ax.plot(states[:,0], states[:,2])

ax.set(xlabel='x', ylabel='z',
       title='')
ax.grid()

plt.show()
```

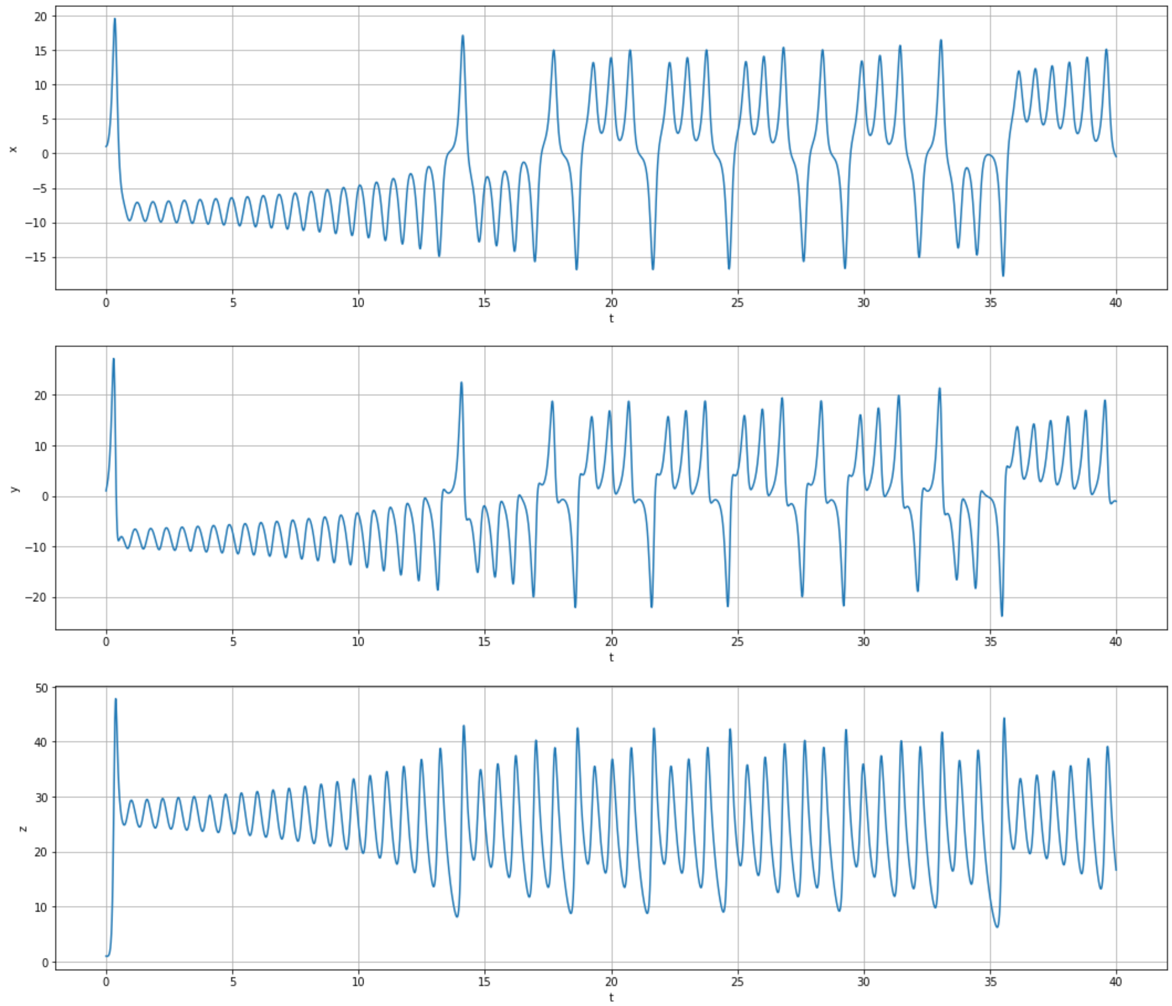


The above code shows the strange attractor in the system. According to Steven H. Strogats's book, the geometrical structure of the strange attractor is a pair of surfaces that merge into one in the lower portion of the plot. Nowadays they are called 'fractal'. 'It is a set of points with zero volumn but infinite surface area.

The **aperiodic** nature of the behavior is better illustrated when we plot the x, y, and z against t.

In [18]:

```
fig, ax = plt.subplots(nrows=3, figsize=(18, 16))

for i, row in enumerate(ax):
    row.plot(t, states[:,i])
    row.set(xlabel='t', ylabel=('x','y','z')[i],
      title='')
    row.grid()

plt.show()
```



To see the bahavior and the shape of the strange attractor more clearly, we can use the following code to shows an animated 3-D plot. The code for the plotting utilities is based on https://jakevdp.github.io/blog/2013/02/16/animating-the-lorentz-system-in-3d/ (https://jakevdp.github.io/blog/2013/02/16/animating-the-lorentz-system-in-3d/)

In [4]:

```
rc('animation', html='jshtml')

def lorenzAnimation(N, s0=None):
    np.random.seed(1)
    if not s0:
        state0 = -15 + 30 * np.random.random((N, 3))
    else:
        state0 = s0
```

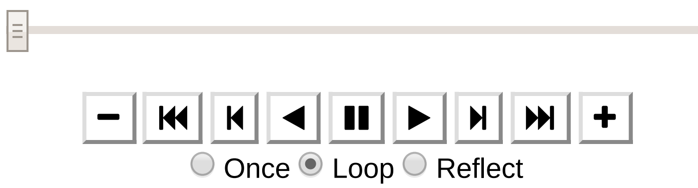
```python
    for i, state0i in enumerate(state0):
        print("Starting point %d: %s" % (i, state0i))

    print("Animation: ")

    t = np.arange(0.0, 20.0, 0.01)
    x_t = np.asarray([odeint(lorenz_deriv, state0i, t)
                    for state0i in state0])

    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')
    colors = plt.cm.jet(np.linspace(0, 1, N))

    lines = sum([ax.plot([], [], [], '-', c=c)
                for c in colors], [])
    pts = sum([ax.plot([], [], [], 'o', c=c)
            for c in colors], [])

    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))

    ax.view_init(30, 0)

    def init():
        for line, pt in zip(lines, pts):
            line.set_data([], [])
            line.set_3d_properties([])

            pt.set_data([], [])
            pt.set_3d_properties([])
        return lines + pts

    def animate(i):
        i = (2 * i) % x_t.shape[1]

        for line, pt, xi in zip(lines, pts, x_t):
            x, y, z = xi[:i].T
            line.set_data(x, y)
            line.set_3d_properties(z)

            pt.set_data(x[-1:], y[-1:])
            pt.set_3d_properties(z[-1:])

        ax.view_init(30, 0.3 * i)
        fig.canvas.draw()
        return lines + pts

    anim = animation.FuncAnimation(fig, animate, init_func=init,
                                   frames=400, interval=20, blit=True)

    return anim
```

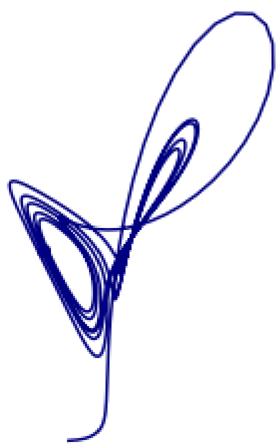
The following shows the trajector for 1 staring points:

In [5]:
```
lorenzAnimation(1)
```

```
Starting point 0: [ -2.48933986   6.6097348  -14.99656876]
Animation:
```

Out[5]:





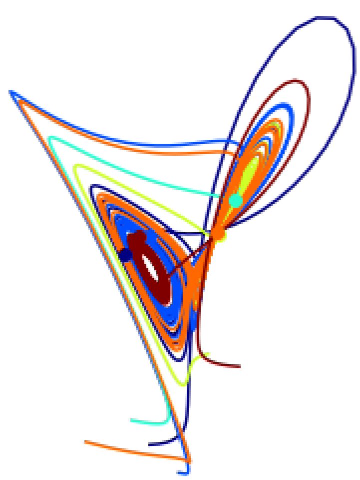
The following shows the trajector for 6 random staring points. This makes it easier to see the shape of the attractor in 3D.

In [26]:
```
lorenzAnimation(6)
```

```
Starting point 0: [ -2.48933986   6.6097348  -14.99656876]
Starting point 1: [ -5.93002282 -10.59732328 -12.22984216]
Starting point 2: [-9.41219366 -4.63317819 -3.09697577]
Starting point 3: [ 1.16450202 -2.42416457  5.55658501]
Starting point 4: [ -8.86643251  11.34352309 -14.1783722 ]
Starting point 5: [ 5.11402531 -2.48085593  1.76069485]
Animation:
```

Out[26]:

○ Once ◉ Loop ○ Reflect



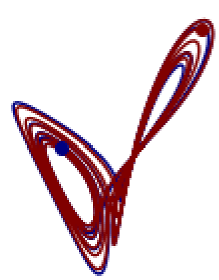
The chaotic system is sensitive to initial conditions. To see how a small perturbation in initial conditions results in diverging trajectory, we simulate the system with two very close starting points. The following animation shows how the two trajectory significantly diverges.

In [7]:
```
state00 = [2.5, 6.6, 15.0]
state01 = [2.49, 6.61, 14.99]
lorenzAnimation(2, [state00, state01])
```

```
Starting point 0: [2.5, 6.6, 15.0]
Starting point 1: [2.49, 6.61, 14.99]
Animation:
```

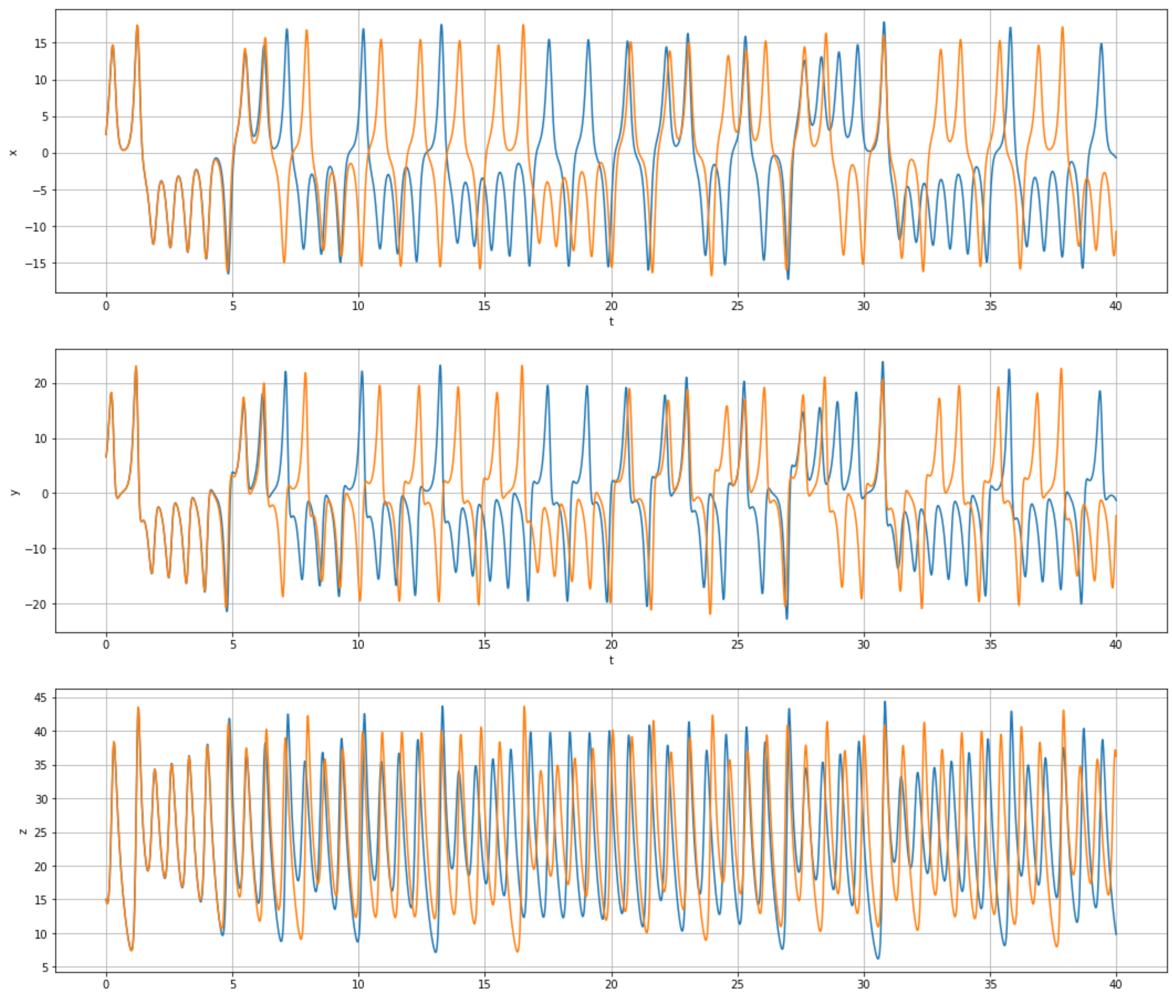
Out[7]:

Once Loop Reflect



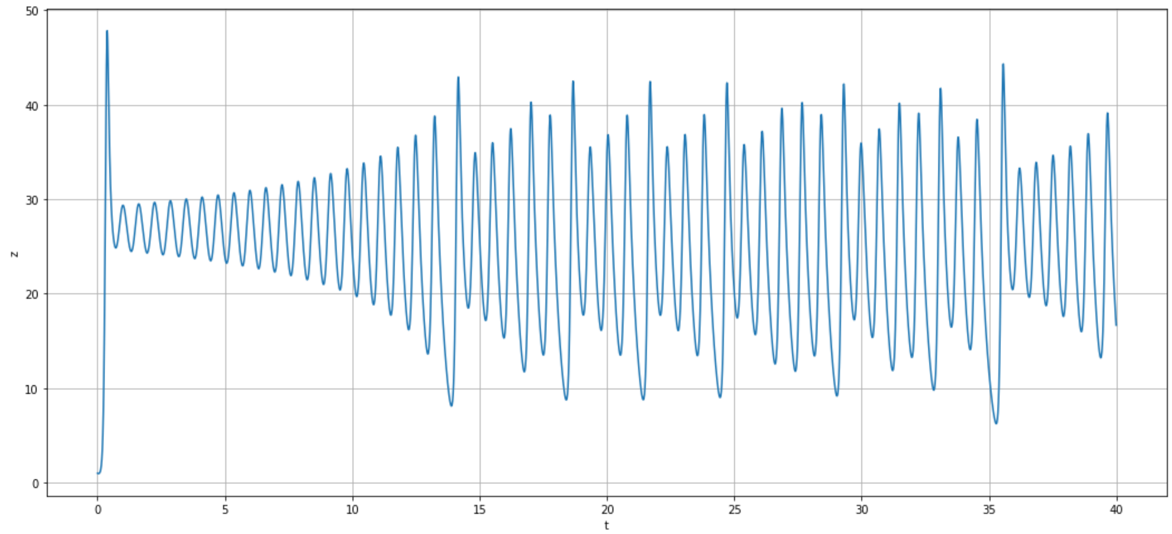
The following shows the x, y, and z of the two trajectories in the following example. We can see how the trajectories stay close for a short while and then diverge.

In [8]:
```
t = np.arange(0.0, 40.0, 0.01)

states0 = odeint(lorenz_deriv, state00, t)
states1 = odeint(lorenz_deriv, state01, t)

fig, ax = plt.subplots(nrows=3, figsize=(18, 16))

for i, row in enumerate(ax):
    row.plot(t, states0[:,i])
    row.plot(t, states1[:,i])
    row.set(xlabel='t', ylabel=('x','y','z')[i],
      title='')
    row.grid()


plt.show()
```



Chaotic system does not imply no order at all. In the following we show the Lorenz Map that depicts the relation between the peaks of z values along time. If we plot z against t again (see figure below), we can see that there are at least some order, in terms of the correlation between consecutive peaks in the plot.

In [20]:
```
fig, ax = plt.subplots(figsize=(18, 8))

ax.plot(t, states[:,2])
ax.set(xlabel='t', ylabel='z',)
ax.grid()

plt.show()
```

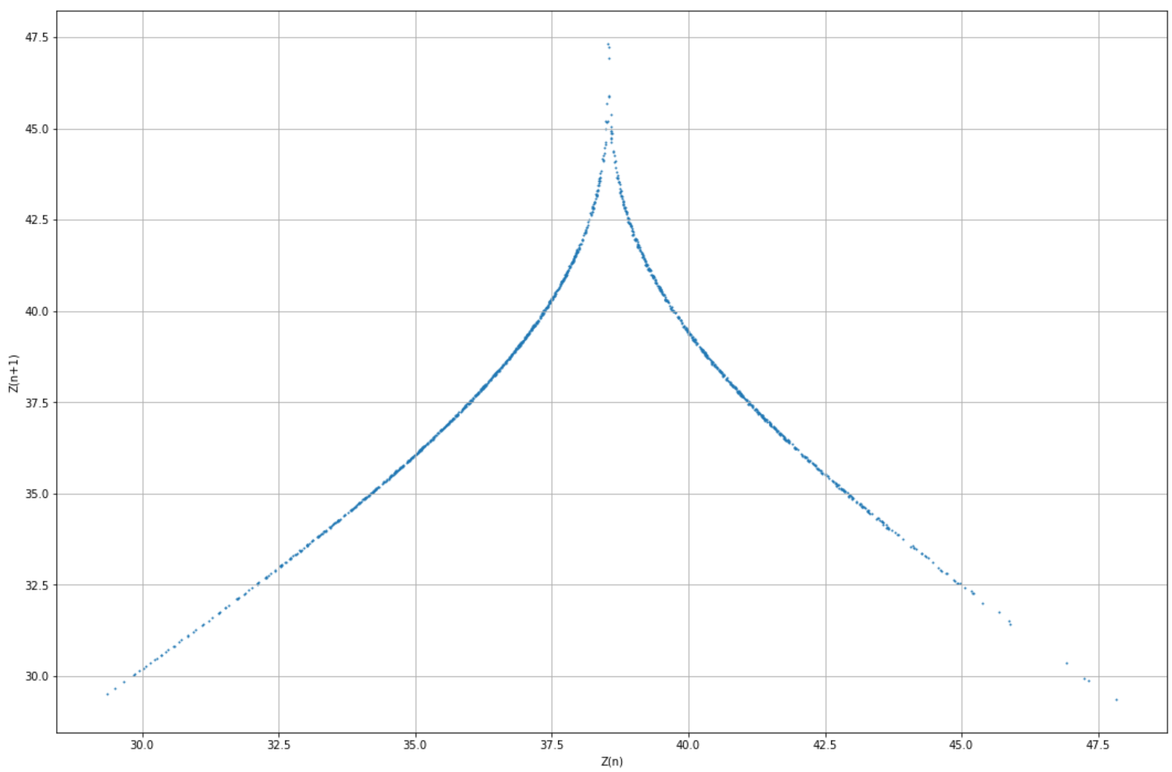


After obtaining the z-t plot, we could check the relationship between z_{n+1} - z_n(shown by the code below), which is called the Lorenz Map. It shows the dynamics on the attractor: we could predict z_{n+1} with z_n. So, starting from z_0, we could move forward with time using itrative calculation.

At t = 0 to 100 with step = 0.01, we have enough points to start to see the "thickness" of the curve.

In [10]:

```
t2 = np.arange(0.0, 1000.0, 0.01)
states2 = odeint(lorenz_deriv, state0, t2)
statesz2 = states2[:,2]

zmax = []
for i, z in enumerate(statesz2):
    if i == 0 or i == len(statesz2) - 1:
      continue
    if z > statesz2[i-1] and z > statesz2[i+1]:
      zmax.append(z)

fig, ax = plt.subplots(figsize=(18, 12))
ax.scatter(zmax[:-1], zmax[1:], s=1)

ax.set(xlabel='Z(n)', ylabel='Z(n+1)',
       title='')
ax.grid()

plt.show()
```



Part 2: Learning chaotic system: using neural networks, delay embeddings, and ensemble kalman filtering

Part 3: Chaos in Multiple Pendulums with application of Neural Networks

These two parts of the coding is done in Matlab and the tutorial is written in Matlab Live Script.

Please check the two Live Script files **NeuralNetworksChaos.mlx** and **ForcastingChaos.mlx** for the tutorial. Use Matlab to open these files.

The Live Script tutorial contains explorations on use machine learning techniques to see if they can be applied to learn and predict the behavior of chaotic systems. The techniques includes **neural networks, delay embeddings, and ensemble kalman filtering** on chaotic systems. The Live Script tutorial also includes an application to **multiple pendulums**

Part 4: Real Life Applications: Sending Secret Message

In this part we look at another simple application.

The unpreditability of the chaotic systems can actually be useful in encryption. For example, the chaotic nature of the system implies that it is practically very difficult to reversely map from trajectories back to initial conditions, and that is the computational asymmetry that we can use to create encrypted messages.

In this particular example, we demostrate the validity of the conclusion in the textbook, that the x projection x(t) of a lorenz system can be used to recover the whole trajectory. This way, an encrytion scheme can be developed in which the message is masked by a chaotic signal. Then a partial representation x(t) is transimitted so that the receiver can recover the complete chaotic signal from the partial signal, while an eavesdropper cannot.

In the following, we show how the "receiver" can generate trajectories from x(t) that ultimately gets arbitrarily close to the original trajectory. We show 3 different starting points, all ends up in sychronization with the original system.

In [11]:
```
def receiver_deriv(state, t0, sender_xt):
    x, y, z = state
    return [sigma * (y - x),
            sender_xt(t0) * (rho - z) - y,
            sender_xt(t0) * y - beta * z]

def messageDemo(N_receiver):
    N = N_receiver + 1
    np.random.seed(1)
    state0 = -15 + 30 * np.random.random((N, 3))

    for i, state0i in enumerate(state0):
        if i == 0:
            print("Sender starting point: %s" % (state0i,))
        else:
            print("Receiver %d starting point: %s" % (i, state0i))Rea
l life applications for chaotic system

    print("Animation: ")
```

```python
    start = 0.0
    end = 20.0
    step = 0.01
    t = np.arange(start, end, step)

    sender_states = odeint(lorenz_deriv, state0[0], t)
    # print(sender_states)
    sender_xs = sender_states[:, 0]
    sender_xt = lambda t: sender_xs[int((t - start) / step)]

    receiver_states = [odeint(receiver_deriv, state0i, t, (sender_xt
,))
            for state0i in state0[1:]]
    # print(receiver_states)

    x_t = np.asarray([sender_states] + receiver_states)
    err_t = [
        [sqrt((r[0]-s[0])**2 + (r[1]-s[1])**2 + (r[2]-s[2])**2)
            for r, s in zip(sender_states, receiver_states_i)]
        for receiver_states_i in receiver_states
    ]


    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')
    colors = plt.cm.jet(np.linspace(0, 1, N))
    lines = sum([ax.plot([], [], [], '-', c=c)
                for c in colors], [])
    pts = sum([ax.plot([], [], [], 'o', c=c)
            for c in colors], [])

    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))
    ax.view_init(30, 0)

    def init():
        for line, pt in zip(lines, pts):
            line.set_data([], [])
            line.set_3d_properties([])

            pt.set_data([], [])
            pt.set_3d_properties([])
        return lines + pts

    def animate(i):
        i = (2 * i) % x_t.shape[1]
        for line, pt, xi in zip(lines, pts, x_t):
            x, y, z = xi[:i].T
            line.set_data(x, y)
            line.set_3d_properties(z)

            pt.set_data(x[-1:], y[-1:])
            pt.set_3d_properties(z[-1:])

        ax.view_init(30, 0.3 * i)
```

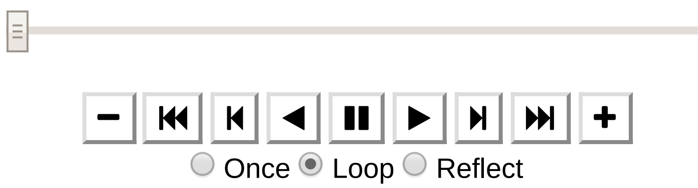
```
        fig.canvas.draw()
        return lines + pts

    anim = animation.FuncAnimation(fig, animate, init_func=init,
                                   frames=400, interval=20, blit=True)

    return anim, err_t, t
```

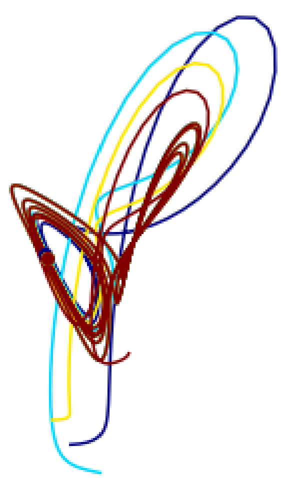
In the following animation, the blue trajectory is the original one, where the other three are generated from x(t) by the receiver. We can observe that all of them come to synchronize with the original.

In [12]:
```
anim, err_t, t = messageDemo(3)
anim
```

```
Sender starting point: [ -2.48933986   6.6097348  -14.99656876]
Receiver 1 starting point: [ -5.93002282 -10.59732328 -12.22984216]
Receiver 2 starting point: [-9.41219366 -4.63317819 -3.09697577]
Receiver 3 starting point: [ 1.16450202 -2.42416457  5.55658501]
Animation:
```

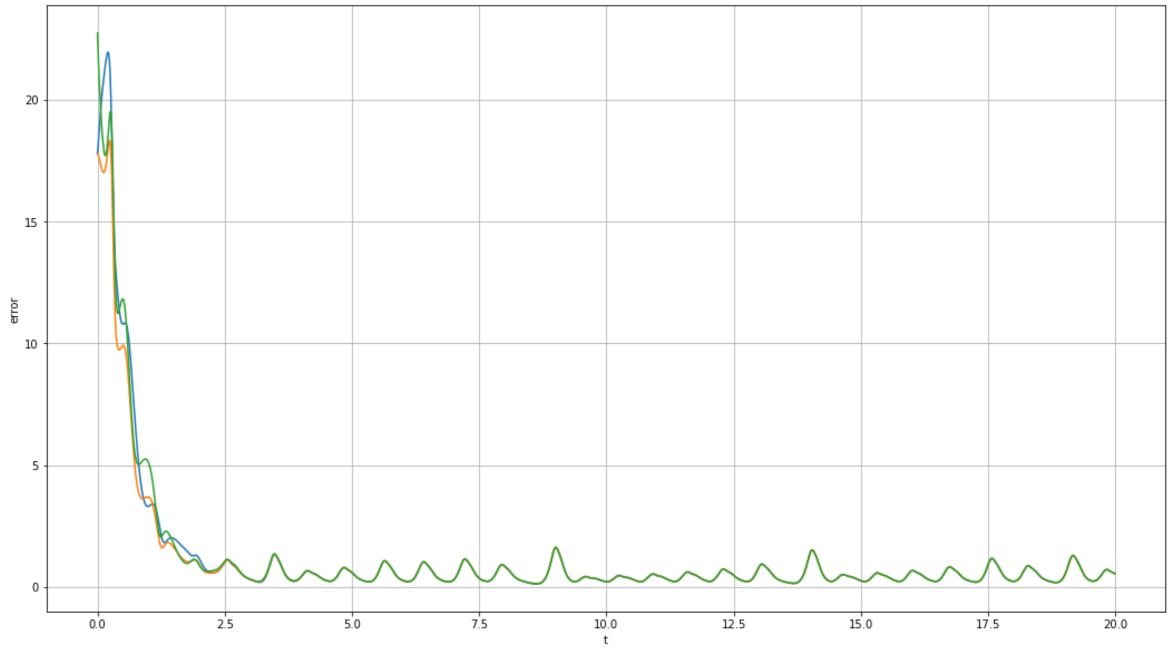
Out[12]:





Below, we plot the error term between the receiver trajectory and the original ("sender") trajectory. One can see that this is consistent with the theoretical result in the textbook (EXAMPLE 9.6.1) that the error term converges to 0.

In [13]:
```python
fig, ax = plt.subplots(figsize=(18, 10))
for i in range(len(err_t)):
    ax.plot(t, err_t[i])

ax.set(xlabel='t', ylabel='error',
       title='')
ax.grid()

plt.show()
```



Division of Work:

- **Gen Mark Veloso Tanno**:
    - Theoretical modeling and coding for applying machine learning techniques to chaotic system.
    - Creating Live Script tutorials.
    - Validation of project ideas.
- **Haiyue Yin**:
    - Validation of project ideas.
    - Research for applications of chaotic system.
    - Theoretical modeling and analysis of chaotic systems.
    - Creating Jupyter Notebook and writing tutorial.
- **Lei Jiang**:
    - Research for applications of chaotic system.
    - Python coding for simulation and plotting of Lorenz system.
    - Creating Jupyter Notebook.