Problem 9

Plot some time series data for the Lorenz system (8.7) when σ = 10, b = 8/3 and 166 ≤ r ≤ 167. When r = 166.2, the solution shows intermittent behavior, and there are occasional chaotic bursts in between what looks like periodic behavior.

In [1]:
```
%pip install -q ipywidgets

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

from ipywidgets import interactive, fixed

from pylab import rcParams
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
rcParams['figure.figsize'] = 20, 10
```

In [3]:
```
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 * (x - y), rho * x - y - x * z, -beta*z+x*y]

    # 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])

    # 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
```

In [4]:
```
w=interactive(solve_lorenz,sigma=10.0,beta=8./3,rho=(155.0,170.0))
w
```

```
interactive(children=(FloatSlider(value=10.0, description='sigma', max=30.0, min=-10.0), FloatSlider(value=2.6…
```

In [5]:
```
def solve_lorenz2(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)
    ax1 = plt.subplot(311)
    ax2 = plt.subplot(312, sharex=ax1)
    ax3 = plt.subplot(313, sharex=ax1)

    # prepare the axes limits
    ax1.set_xlim((0, 1000))
    #ax1.set_ylim((0, 1000))
    ax2.set_xlim((0, 1000))
    #ax2.set_ylim((0, 1000))
    ax3.set_xlim((0, 1000))
    #ax3.set_ylim((0, 1000))

    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])

    x, y, z = x_t[20,:,:].T

    ax1.plot(z)
    ax1.set_ylabel('$z$')

    ax2.plot(y)
    ax2.set_ylabel('$y$')

    ax3.plot(x)
    ax3.set_ylabel('$x$')
    ax3.set_xlabel('$n$')
    plt.show()

    return t, x_t
```

In [6]:
```
w=interactive(solve_lorenz2,sigma=10.0,beta=8./3,rho=(155.0,170.0))
w
```

```
interactive(children=(FloatSlider(value=10.0, description='sigma', max=30.0, min=-10.0), FloatSlider(value=2.6…
```

Problem 2

In [7]:
```
import numpy as np
import sympy as sm
from sympy import solve
from sympy.abc import x, y, z

Xdot = x+2*z
Ydot = y-3*z
Zdot = 2*y+z

XdotZer = sm.Eq(Xdot, 0)
YdotZer = sm.Eq(Ydot, 0)
ZdotZer = sm.Eq(Zdot, 0)

CriticalPoints = sm.solve( [XdotZer, YdotZer, ZdotZer], x, y, z )
CriticalPoints
```

Out[7]:
```
{x: 0, y: 0, z: 0}
```

In [8]:
```
JM2 = sm.Matrix([ Xdot, Ydot, Zdot ])
JM3 = sm.Matrix([ x, y, z ])
JacobianM = JM2.jacobian(JM3)
JacobianM
```

Out[8]:

In [9]:
```
JM = JacobianM.subs([ (x, 0), (y, 0), (z,0) ])
JM
```

Out[9]:

In [10]:
```
JM = np.float64(JM)
Eig = np.linalg.eig(JM)
Eig
```

Out[10]:
```
(array([1.+0.j        , 1.+2.44948974j, 1.-2.44948974j]),
 array([[ 1.        +0.j        , -0.45883147+0.j        ,
         -0.45883147-0.j        ],
        [ 0.        +0.j        ,  0.6882472 +0.j        ,
          0.6882472 -0.j        ],
        [ 0.        +0.j        ,  0.        -0.56195149j,
          0.        +0.56195149j]]))
```

In [11]:
```
# Eigenvalues are
Eig[0][0], Eig[0][1], Eig[0][2]
```

Out[11]:
```
((1+0j), (1+2.4494897427831783j), (1-2.4494897427831783j))
```

In [12]:
```
rcParams['figure.figsize'] = 20, 10
```

In [16]:
```
def solve_lorenz2(sigma=1, beta=1, rho=1):
    max_time = 4.0
    N = 30

    fig = plt.figure(1)
    ax1 = plt.subplot(311)
    ax2 = plt.subplot(312, sharex=ax1)
    ax3 = plt.subplot(313, sharex=ax1)

    # prepare the axes limits
    ax1.set_xlim((0, 1000))
    #ax1.set_ylim((0, 1000))
    ax2.set_xlim((0, 1000))
    #ax2.set_ylim((0, 1000))
    ax3.set_xlim((0, 1000))
    #ax3.set_ylim((0, 1000))

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
        x, y, z = x_y_z
        return [x+2*z,y-3*z,2*y+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])

    # Change 20 to any other to test different initial condition
    x, y, z = x_t[20,:,:].T

    ax1.plot(z)
    ax1.set_ylabel('$z$')

    ax2.plot(y)
    ax2.set_ylabel('$y$')

    ax3.plot(x)
    ax3.set_ylabel('$x$')
    ax3.set_xlabel('$n$')
    plt.show()

    return t, x_t
```

In [17]:
```
w=interactive(solve_lorenz2,sigma=1,beta=1,rho=1)
w
```

```
interactive(children=(IntSlider(value=1, description='sigma', max=3, min=-1), IntSlider(value=1, description='…
```

In [ ]:

In [14]:
```
def solve_lorenz(sigma=1, beta=1, rho=1):

    max_time = 4.0
    N = 30

    fig = plt.figure(1)
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
        x, y, z = x_y_z
        return [x+2*z,y-3*z,2*y+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])

    # 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
```

In [15]:
```
w=interactive(solve_lorenz,sigma=1,beta=1,rho=1)
w
```

```
interactive(children=(IntSlider(value=1, description='sigma', max=3, min=-1), IntSlider(value=1, description='…
```