Nonlinear Project

May 4, 2015

1 Synchronization in Chaotic Systems

Based on the work of Louis Pecora and Thomas Carroll in 1989, and Kevin Cumo and Alan Oppenhiem in 1993

What does it mean?

- Systems that follow each other

Why does it happen?

- Math!

Who cares?

- It's cool! Some applications, a simple crypt is implemented here

2 Quick Review of Chaotic Systems

Examples will be worked through the familiar (and awesome) Lorenz Equation.

$$\dot{x} = s(y - x) \tag{1}$$

$$\dot{y} = rx - y - xz \tag{2}$$

$$\dot{z} = xy - bz \tag{3}$$

Import necessary python toolkits

```
In [1]: import numpy as np
        import math
        import matplotlib.pyplot as plt
        from scipy.integrate import odeint
        from mpl_toolkits.mplot3d.axes3d import Axes3D
        from scipy.io.wavfile import read, write
        %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Define our parameters and initial conditions

```
In [2]: s = 10
        r = 28
        b = 8.0/3

        tf = 100.0
        numSteps = 100000
        dt = tf/numSteps

        t = np.linspace(0,tf,numSteps)

        x = np.zeros(numSteps)
        y = np.zeros(numSteps)
        z = np.zeros(numSteps)

        x[0] = 0
        y[0] = 1
        z[0] = 0
```

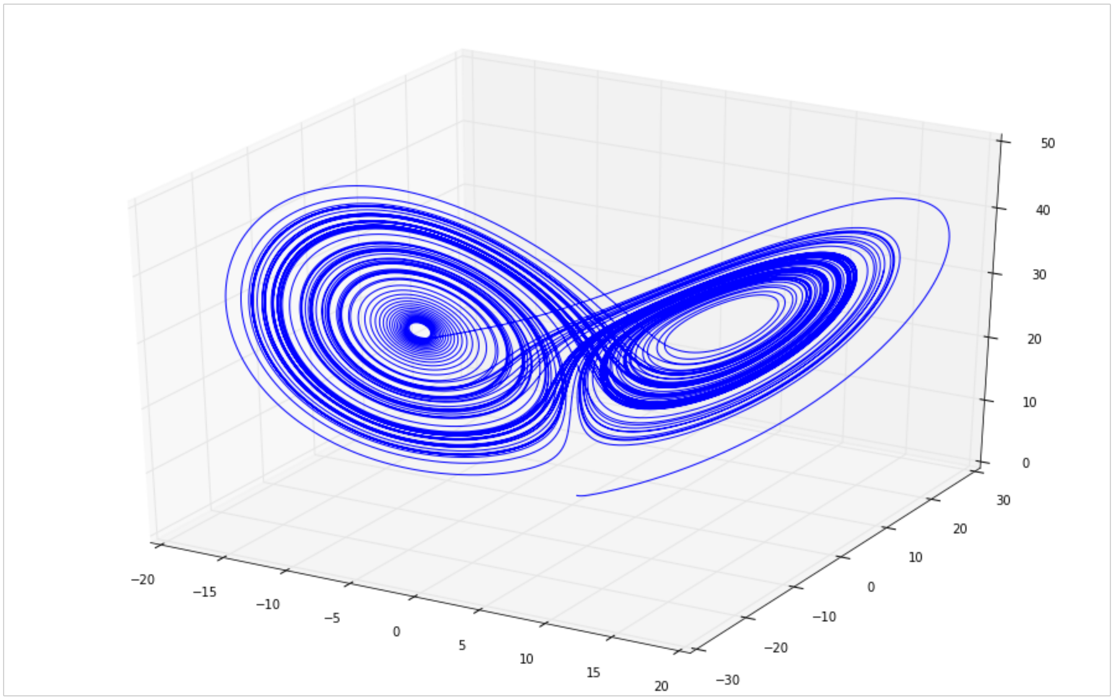
Solve the system (numerically)

```
In [3]: for i in range(numSteps-1):
            x[i+1] = x[i] + (s * (y[i]-x[i]))*dt
            y[i+1] = y[i] + (r*x[i] - y[i] - x[i]*z[i])*dt
            z[i+1] = z[i] + (x[i]*y[i]- b*z[i])*dt
```

Plot phase space

```
In [4]: fig = figure(figsize=(16,10))
        ax = fig.gca(projection='3d')
        ax.plot(x, y, z)

Out[4]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x106623990>]
```



Now lets see what happens when we change the initial condition very slightly.

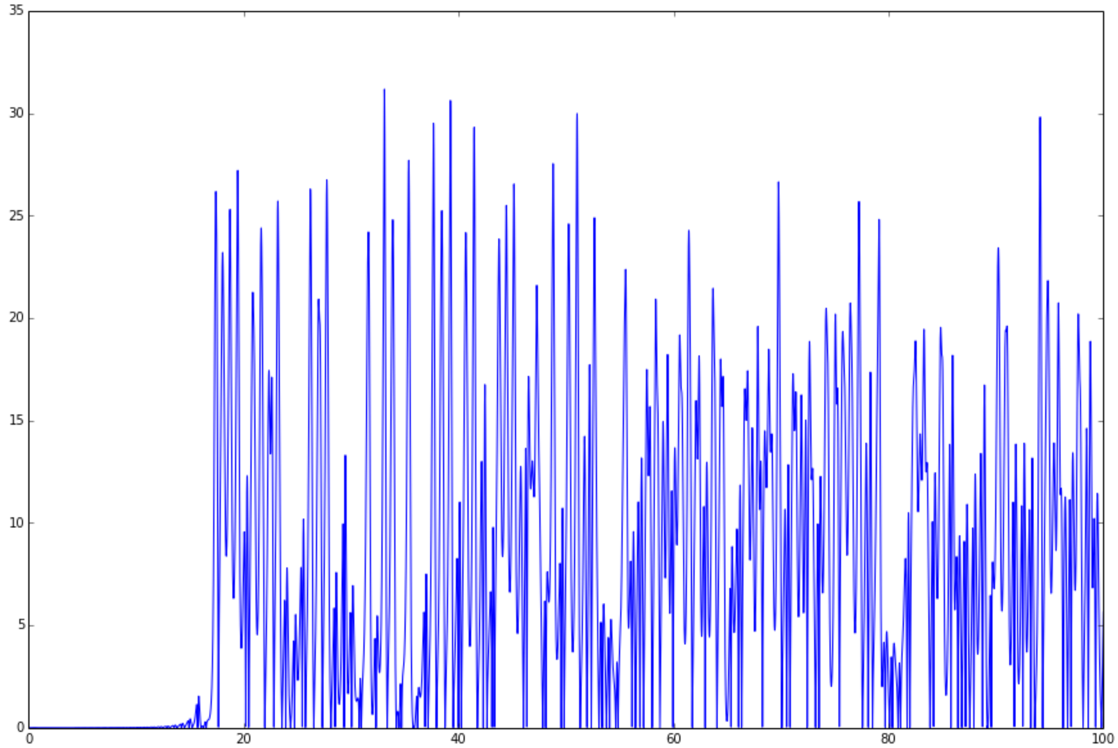
In [5]:
```
x_s = np.zeros(numSteps)
y_s = np.zeros(numSteps)
z_s = np.zeros(numSteps)

x_s[0] = 0
y_s[0] = 1
z_s[0] = 0.01

for i in range(numSteps-1):
    x_s[i+1] = x_s[i] + (s * (y_s[i]-x_s[i]))*dt
    y_s[i+1] = y_s[i] + (r*x_s[i] - y_s[i] - x_s[i]*z_s[i])*dt
    z_s[i+1] = z_s[i] + (x_s[i]*y_s[i] - b*z_s[i])*dt

fig, ax = plt.subplots();
fig.set_size_inches(15,10);
ax.plot(t,abs(x-x_s))
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x1067563d0>]
```



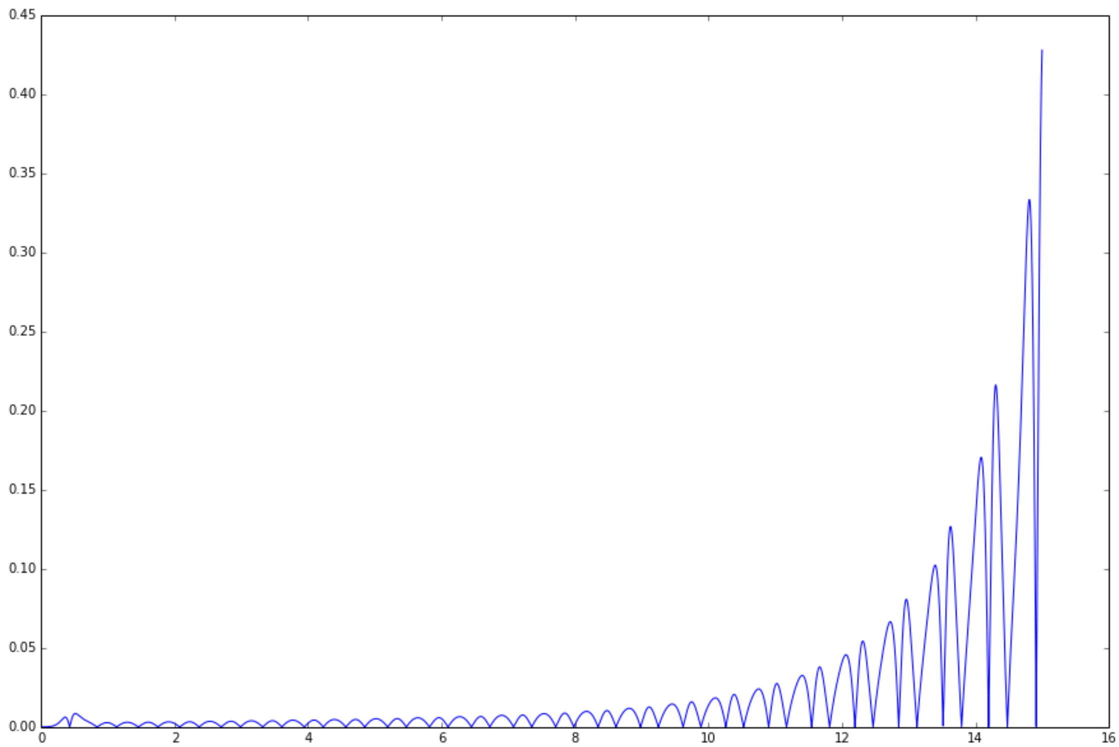
3 Lyapunov Exponents

$$|\delta \mathbf{Z}| \approx e^{\lambda t} |\delta \mathbf{Z_o}| \tag{4}$$

** It's intutitive!**

In [6]:
```
fig, ax = plt.subplots();
fig.set_size_inches(15,10);
ax.plot(t[0:15000],abs(x-x_s)[0:15000])
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x10a01d050>]
```



No surprise here! The lorenz system has positive Lyapunov exponents

4 Synchronization

From what we know about choas, chaotic systems should never synchronize right?!

Subsystems will synchronize only if the subsystem Lyapunov exponents are all negative

What does that mean?

5 Subsystems

$$\dot{u} = f(u) \tag{5}$$

Divide the system up totally arbitrarily. Let

$$v = [u_1, ..., u_m]^T \tag{6}$$

$$w = [u_{m+1}, ..., u_n]^T \tag{7}$$

Then

$$g = [f_1(u), ..., f_m(u)]^T \tag{8}$$

$$h = [f_{m+1}(u), ..., f_n(u)]^T \tag{9}$$

It's clear that

$$\dot{v} = g(v, w) \tag{10}$$

$$\dot{w} = h(v, w) \tag{11}$$

Great! We now know what subsystems are

6 Output to input subsystems

Create a new subsystem w' but use the solution v

$$\dot{v} = g(v, w) \tag{12}$$

$$\dot{w} = h(v, w) \tag{13}$$

$$\dot{w}' = h(v, w') \tag{14}$$

Now lets look at the error of w' and w
Obviously synchronization will occur only if

$$w' - w \to 0 \quad as \quad t \to \infty \tag{15}$$

7 Back to Lorenz!

$$\dot{x} = s(y - x) \tag{16}$$

$$\dot{y} = rx - y - xz \tag{17}$$

$$\dot{z} = xy - bz \tag{18}$$

Consider these subsystems

$$\dot{x}_1 = s(y - x_1) \tag{19}$$

$$\dot{z}_1 = x_1 y - b z_1 \tag{20}$$

$$\dot{y}_2 = rx - y_2 - x z_2 \tag{21}$$

$$\dot{z}_2 = x y_2 - b z_2 \tag{22}$$

Proof error goes to 0 for subsystem 1

$$J = \begin{bmatrix} -s & 0 \\ y & -b \end{bmatrix} \tag{23}$$

The characteristic polynomail is

$$(s+\lambda)(b+\lambda) = 0 \tag{24}$$

The roots are

$$\lambda_1 = -b \tag{25}$$

$$\lambda_2 = -s \tag{26}$$

It is more complicated to show subsystem 2 has negative Laypunov exponents, numerical techniques are needed.

IDEA: Let $x(t)$ drive subsystem 2, then let y_2 drive subsystem 1!

$$\dot{x}_s = s(y_s - x_s) \tag{27}$$

$$\dot{y}_s = rx - y_s - x z_s \tag{28}$$

$$\dot{z}_s = x y_s - b z_s \tag{29}$$

This should work, but let's check out the error dynamics

$$e = [x \quad y \quad z]^T - [x_s \quad y_s \quad z_s]^T \tag{30}$$

$$\dot{e}_1 = s(e_2 - e_1) \tag{31}$$

$$\dot{e}_2 = -e_2 - x e_3 \tag{32}$$

$$\dot{e}_3 = x e_2 - b e_3 \tag{33}$$

Lyapunov function!

$$V = \frac{1}{2}(\frac{1}{s}e_1^2 + e_2^2 + 4e_3^2) \tag{34}$$

$$\dot{V} = \frac{1}{s}e_1\dot{e}_1 + e_2\dot{e}_2 + 4e_3\dot{e}_3 \tag{35}$$

$$\dot{V} = -(e_1 - \frac{1}{2}e_2)^2 - \frac{3}{4}e_2^2 - 4be_3^2 \tag{36}$$

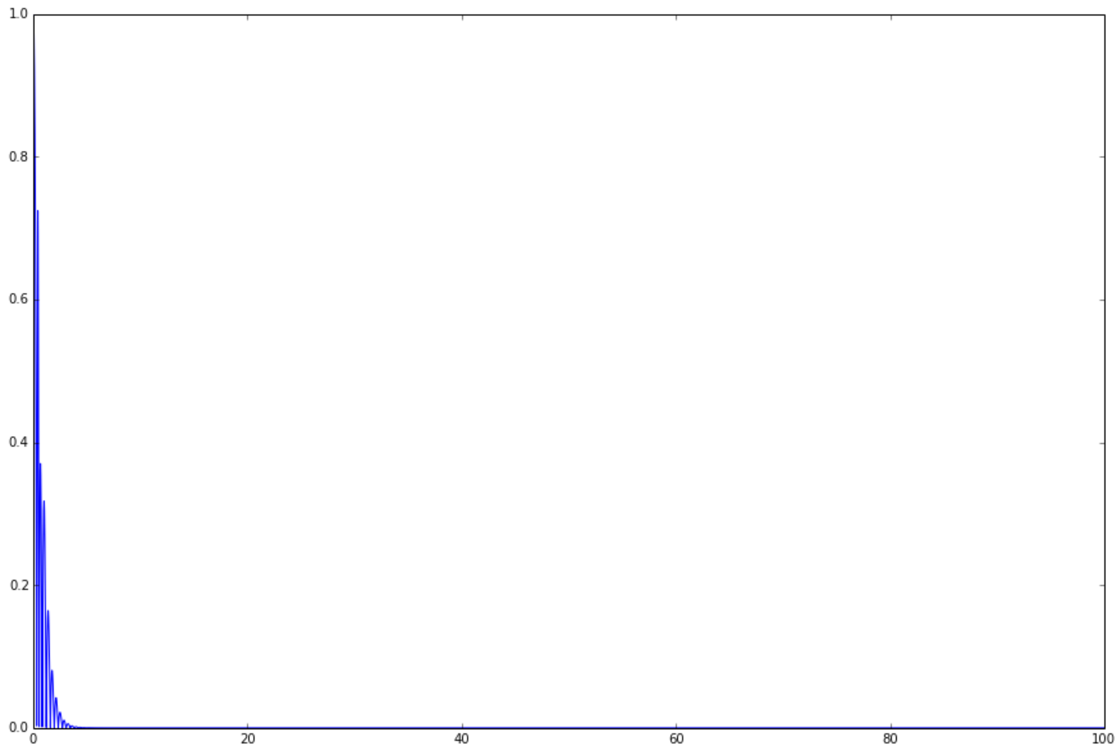
Negative definite -> Proof of synchronization!

Simulation Time!

```
In [7]: x_s[0] = 1.
        y_s[0] = 2.
        z_s[0] = 3.
        for i in range(numSteps-1):
            x_s[i+1] = x_s[i] + (s * (y_s[i]-x_s[i]))*dt
            y_s[i+1] = y_s[i] + (r*x[i] - y_s[i] - x[i]*z_s[i])*dt
            z_s[i+1] = z_s[i] + (x[i]*y_s[i] - b*z_s[i])*dt

        fig, ax = plt.subplots();
        fig.set_size_inches(15,10);
        ax.plot(t,abs(x-x_s))
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x106738a10>]
```



Wow that worked better than expected didn't it!

8 Application - Sending Your Fav Embarrasing Song To Your Bff

Plan of Attack - Read in song - Tell your friend beforehand s, r, b - Add $x(t)$ to the song such that $x(t)$ is much more powerful than the song - Send message $m(t) = song + x(t)$ - Your friend will read in $m(t)$ to his synchronous system - Your friend will subtract his $x_s(t)$ from the message $m(t)$ - Your friend will enjoy your favorite song

Read in your fav song!

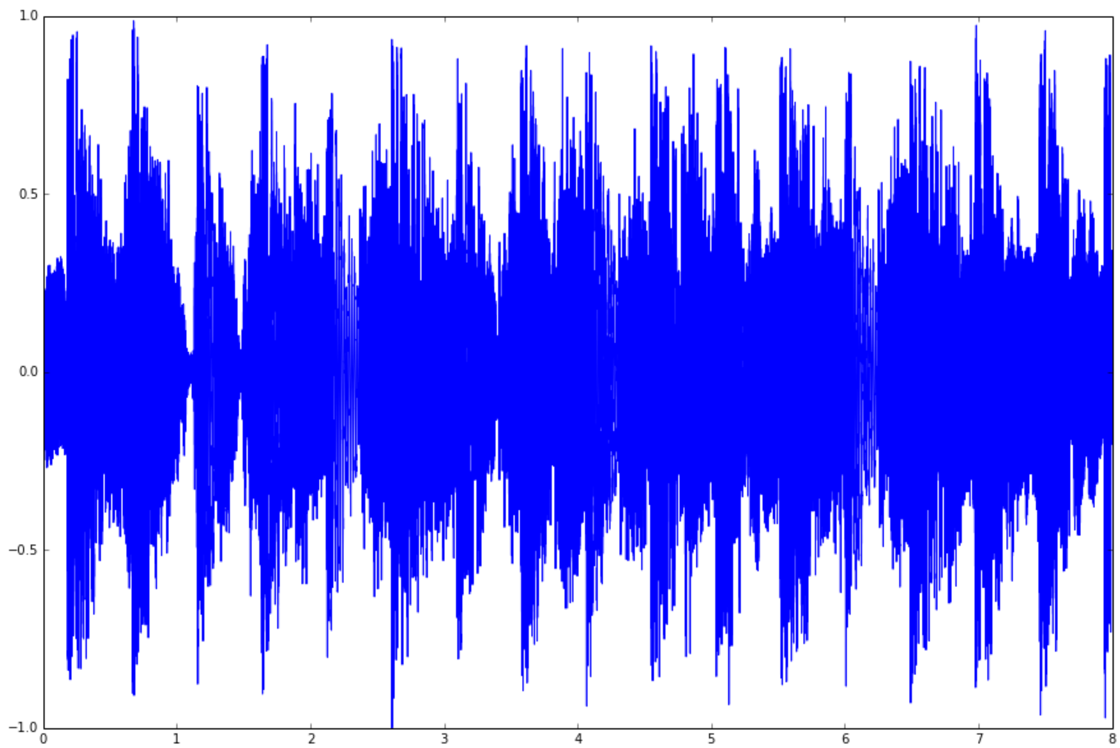
In [8]:
```
(sample_rate,song) = read("FavSong.wav")

tf = len(song)/sample_rate
numSteps = len(song)
dt = tf/numSteps

t = np.linspace(0,tf,numSteps)

fig, ax = plt.subplots();
fig.set_size_inches(15,10);
ax.plot(t,song)

!afplay FavSong.wav
```



Solve the System Numerically, then add the system to the song

In [11]:
```
x = np.zeros(numSteps)
y = np.zeros(numSteps)
z = np.zeros(numSteps)
```

```
x[0] = 0
y[0] = 1
z[0] = 0

tf = 200.
dt = tf/numSteps
t = np.linspace(0,tf,numSteps)

for i in range(numSteps-1):
    x[i+1] = x[i] + (s * (y[i]-x[i]))*dt
    y[i+1] = y[i] + (r*x[i] - y[i] - x[i]*z[i])*dt
    z[i+1] = z[i] + (x[i]*y[i] - b*z[i])*dt

message = x + .001*song

write("Message.wav",sample_rate,message)
!afplay Message.wav
```

Set up the synchronous system

In [12]:
```
x_s = np.zeros(numSteps)
y_s = np.zeros(numSteps)
z_s = np.zeros(numSteps)

x_s[0] = 1.
y_s[0] = 2.
z_s[0] = 3.

for i in range(numSteps-1):
    x_s[i+1] = x_s[i] + (s * (y_s[i]-x_s[i]))*dt
    y_s[i+1] = y_s[i] + (r*message[i] - y_s[i] - message[i]*z_s[i])*dt
    z_s[i+1] = z_s[i] + (message[i]*y_s[i] - b*z_s[i])*dt
```

Decrypt the song and see how we did!

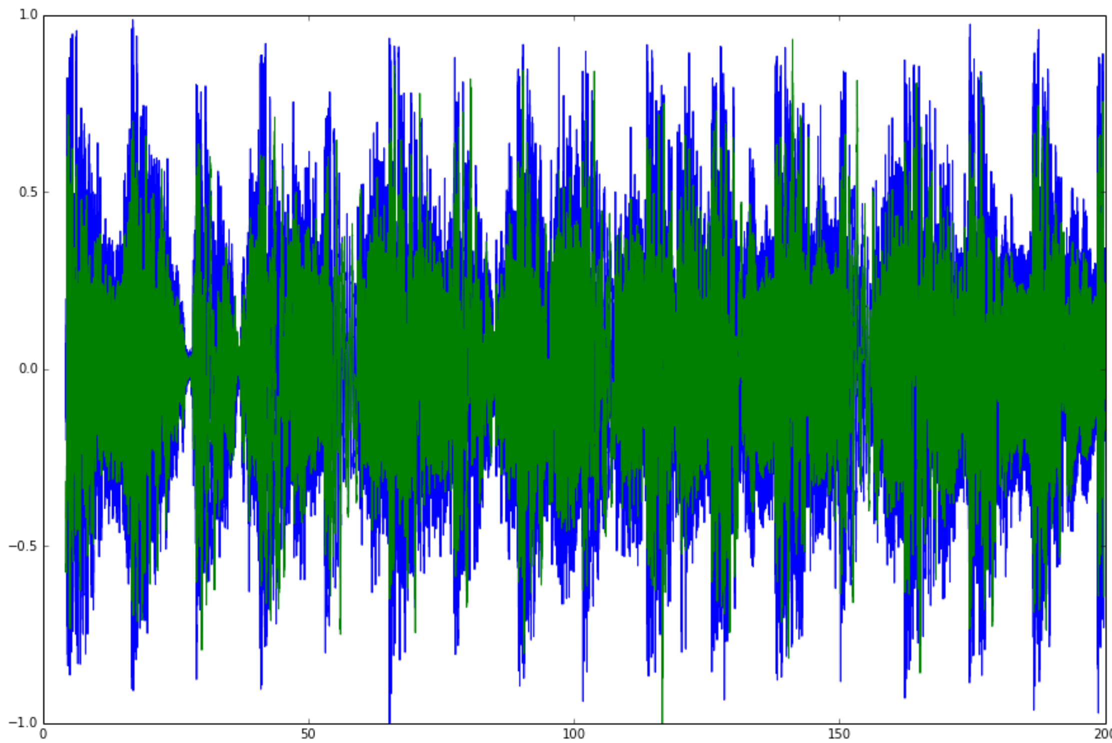
In [13]:
```
ic_error_length = 8000

decrypt = message - x_s
decrypt = decrypt[ic_error_length:]
decrypt = decrypt/abs(decrypt).max()

fig, ax = plt.subplots();


fig.set_size_inches(15,10);
ax.plot(t[ic_error_length:],song[ic_error_length:],t[ic_error_length:],decrypt)
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x10b0d2850>,
 <matplotlib.lines.Line2D at 0x112547ed0>]
```




```
In [40]: write("Decrypt.wav",sample_rate,decrypt)
         !afplay Decrypt.wav
```

Woooo it works

9 Sensitivity to the Key - Experimental Approach

Set up system with parameters changed by 30%

```
In [18]: s2 = s*1.01
         r2 = r*1.01
         b2 = b*1.01

         x_s2 = np.zeros(numSteps)
         y_s2 = np.zeros(numSteps)
         z_s2 = np.zeros(numSteps)


         x_s2[0] = 1.
         y_s2[0] = 2.
         z_s2[0] = 3.
```

```
for i in range(numSteps-1):
    x_s2[i+1] = x_s2[i] + (s2 * (y_s2[i]-x_s2[i]))*dt
    y_s2[i+1] = y_s2[i] + (r2*message[i] - y_s2[i] - message[i]*z_s2[i])*dt
    z_s2[i+1] = z_s2[i] + (message[i]*y_s2[i] - b2*z_s2[i])*dt
```

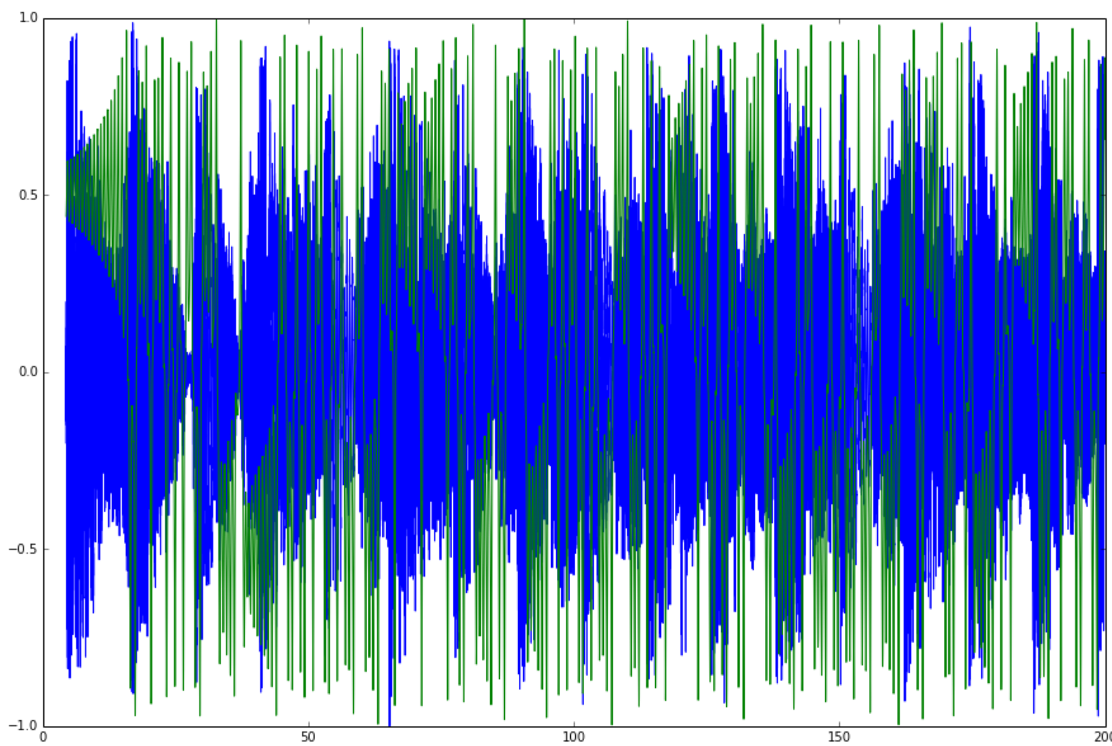
Decrypt and Compare

```
In [19]: ic_error_length = 8000

         decrypt2 = message - x_s2
         decrypt2 = decrypt2[ic_error_length:]
         decrypt2 = decrypt2/abs(decrypt2).max()

         fig, ax = plt.subplots();
         fig.set_size_inches(15,10);
         ax.plot(t[ic_error_length:],song[ic_error_length:],t[ic_error_length:],decrypt2)
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x112599110>,
          <matplotlib.lines.Line2D at 0x10b01db10>]
```




```
In [20]: write("Decrypt2.wav",sample_rate,decrypt2)
         !afplay Decrypt2.wav
```

No synchronization occurs, the embarrasing (and fantastic) song is safe!