Set 1: Lissajous Figures and Beats

Aritra Biswas

1 Computing sinusoids

We compute sequences of the following trigonometric functions:

$$X(t) = A_X \cos(2\pi f_X t) \tag{1}$$
$$Y(t) = A_Y \sin(2\pi f_Y t + \phi) \tag{2}$$
$$Z(t) = X(t) + Y(t) \tag{3}$$

for $t = n\Delta t$ with $n = 0, \ldots, N$. We do this first with `python` lists:

```
import math
import sys

def usage():
    '''Prints an usage message.'''
    print 'Usage: %s [f_X] [f_Y] [A_X] [A_Y] [phi] [dt] [N]' % sys.argv[0]
    print 'N must be an integer, all other values must be integers or floats'

def calctXYZ(f_X, f_Y, A_X, A_Y, phi, dt, N):
    '''Given relevant constants, evaluates X(t), Y(t), and Z(t) for t = n * dt
    where n = 0, ..., N. Returns t, X, Y, and Z arrays.'''
    t = []
    X = []
    Y = []
    Z = []
    for n in range(N + 1):
        t.append(n * dt)
        X.append(A_X * math.cos(2 * math.pi * f_X * t[n]))
        Y.append(A_Y * math.sin( (2 * math.pi * f_Y * t[n]) + phi))
        Z.append(X[n] + Y[n])
    return (t, X, Y, Z)

def writetXYZ(t, X, Y, Z):
    '''Given t, X, Y, and Z arrays, prints CSV output as t,X,Y,Z.'''
    assert len(t) == len(X) == len(Y) == len(Z)
    print len(t)
    for i in range(len(t)):
        print '%f,%f,%f,%f' % (t[i],X[i],Y[i],Z[i])

if __name__ == '__main__':
    try:
        f_X = float(sys.argv[1])
        f_Y = float(sys.argv[2])
```

```
        A_X = float(sys.argv[3])
        A_Y = float(sys.argv[4])
        phi = float(sys.argv[5])
        dt = float(sys.argv[6])
        N = int(sys.argv[7])
        (t, X, Y, Z) = calctXYZ(f_X, f_Y, A_X, A_Y, phi, dt, N)
        writetXYZ(t, X, Y, Z)
    except IndexError, TypeError:
        usage()
```

The program can be made neater and more efficient with `numpy` arrays. We rewrite the following two functions:

```
def calctXYZ(f_X, f_Y, A_X, A_Y, phi, dt, N):
    '''Given relevant constants, evaluates X(t), Y(t), and Z(t) for t = n * dt
    where n = 0, ..., N. Returns t, X, Y, and Z arrays.'''
    n = np.arange(0, N + 1)
    t = n * dt
    X = A_X * np.cos(2 * np.pi * f_X * t)
    Y = A_Y * np.sin((2 * np.pi * f_Y * t) + phi)
    Z = X + Y
    return (t, X, Y, Z)

def writetXYZ(t, X, Y, Z):
    '''Given t, X, Y, and Z arrays, prints CSV output as t,X,Y,Z.'''
    data = np.column_stack((t, X, Y, Z))
    np.savetxt(sys.stdout, data, delimiter=',')
```

To save the data to an ASCII file as desired, we can redirect output to a file at runtime, or in the `numpy` version we can simply replace `sys.stdout` with the desired file.

2 Lissajous figures

We generate Lissajous figures as follows, using `varf` and `varphi` as necessary in the interpreter. Predetermined conditions are: $A_X = 1, A_Y = 2, \phi = 0.5, dt = 0.001, N = 1000$).

```
from computeXYZ_numpy import *
import matplotlib.pyplot as plotter

def varf(f_X, f_Y):
    '''Generates a Lissajous figure (XY plot) with given f_X and f_Y and
    preset values for other variables.'''
    (t, X, Y, Z) = calctXYZ(f_X, f_Y, 1, 2, 0.5, 0.001, 1000)
    plotter.plot(X, Y)
    plotter.xlabel('X')
    plotter.ylabel('Y')
    plotter.show()

def varphi(phi):
    '''Generates a Lissajous figure (XY plot) with given list of phi
    values and preset values for other variables.'''
    plotter.xlabel('X')
```

```
plotter.ylabel('Y')
for phi_i in phi:
    (t, X, Y, Z) = calctXYZ(1, 1, 1, 2, phi_i, 0.001, 1000)
    plotter.plot(X, Y, label=str(phi_i/np.pi) + 'pi')
plotter.legend()
plotter.show()
```

2.1 Closed curves when f_X/f_Y is rational

Proposition. If f_X/f_Y is a rational number, then the Lissajous figure is a closed curve.

Proof. A sinusoidal function $\sin(\omega t)$ has period $2\pi/\omega$. Thus $X(t)$ has period $1/f_X$ and $Y(t)$ has period $1/f_Y$. In other words, for all integers j and k, we have:

$$X(t + j/f_X) = X(t) \text{ and } Y(t + k/f_Y) = Y(t)$$

Since f_X/f_Y is rational, it can be written as a fraction of two integers. Let $f_X/f_Y = j/k$ where j and k are integers. Then $j/f_X = k/f_Y$. Let $\varphi = j/f_X = k/f_Y$. Then, for any point, $X(t + \varphi) = X(t)$ and $Y(t + \varphi) = Y(t)$.

Thus, for any point $(X(t), Y(t))$, there exists an identical point $(X(t+\varphi), Y(t+\varphi))$ in the parametrization, so each point is revisited infinitely many times, implying the graph is a closed curve. □

Figure 1 illustrates this phenomenon. Notably, the curve in 1f would actually be closed if we plotted with high enough N, since the program uses `np.pi`, a finite-length rational approximation of π.

2.2 Effect of f_X/f_Y on curve shape

From Figure 1, we can infer some apparent patterns. The relationship between figures 1b and 1e suggests that inverting f_X/f_Y causes the Lissajous figure to flip over the $Y = -X$ line.

Furthermore, when $f_X > f_Y$, increasing the ratio seems to increase the number of turns in the loop. Given the prior observation, this suggests that the opposite will be true when $f_Y > f_X$.

We also consider the special case with $f_X = 2$, $f_Y = 1$, and $\phi = 0$, shown in figure 2. Although the figure doesn't enclose an area, it is closed in the sense that every point is revisited infinitely many times in a parametrization of the curve. In the time that $Y(t)$ completes a cycle (-1 to 1 and back to -1), $X(t)$ completes two (-1 to 1 -1 on the way up, and another cycle on the way down).

2.3 Effect of phase shift ϕ with $f_X = f_Y$

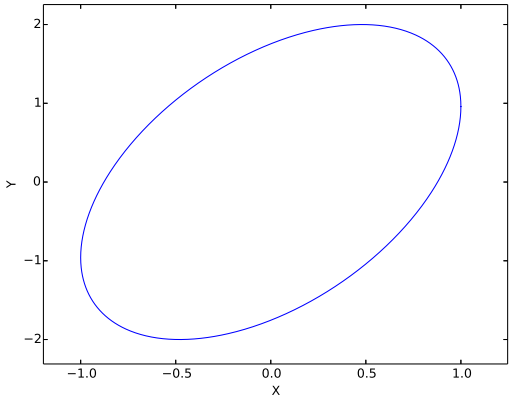
Whenever ϕ is an integer multiple of π, the Lissajous figure is circular (identical to the red curve with $\phi = 0$ in figure 3).

Furthermore, figure 3 also shows that the Lissajous figure deviates from a circular shape as ϕ becomes farther from an integer multiple of π.

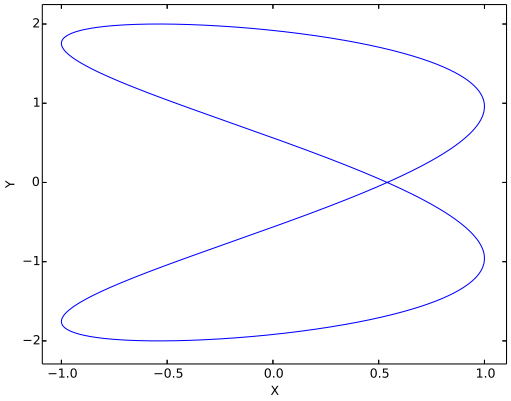
Thus, the Lissajous figure shows how close two sinusoids are to being exactly in or out of phase. To tune circuits using an oscilloscope, one can adjust the phase shift until the visible Lissajous figure is circular.

3 Beats

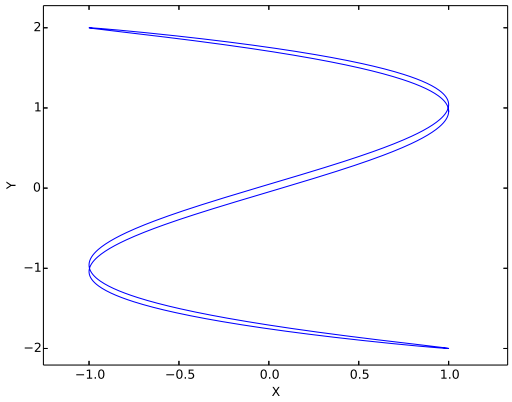
We generate the $Z(t)$ vs. t graph, and optionally its Fourier transform, in `plotZ`, with the predetermined conditions $A_X = 1, A_Y = 1, \phi = 0.5, dt = 0.001, N = 50000$.



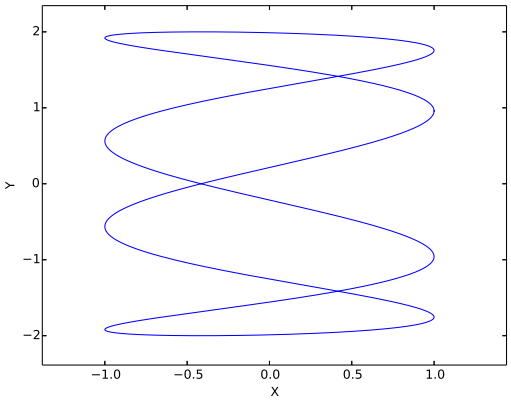
(a) $f_X/f_Y = 1$



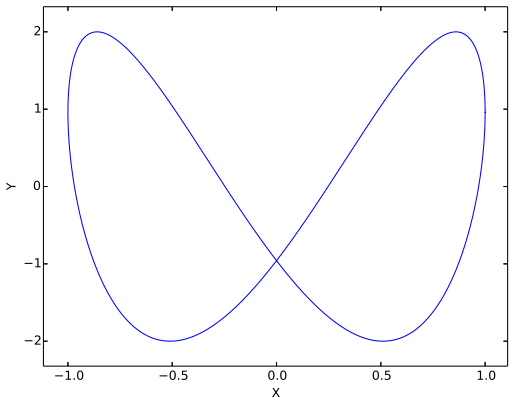
(b) $f_X/f_Y = 2$



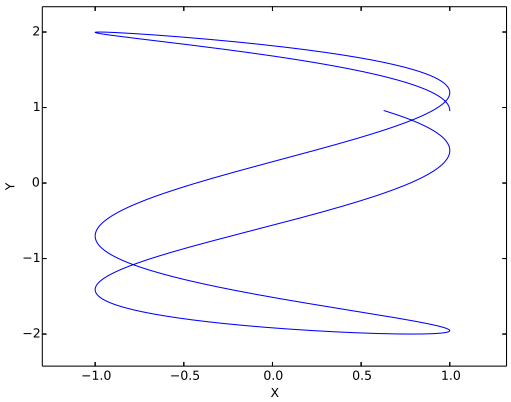
(c) $f_X/f_Y = 3$



(d) $f_X/f_Y = 4$



(e) $f_X/f_Y = 1/2$



(f) $f_X/f_Y = \pi$

Figure 1: Lissajous figures with various f_X/f_Y.

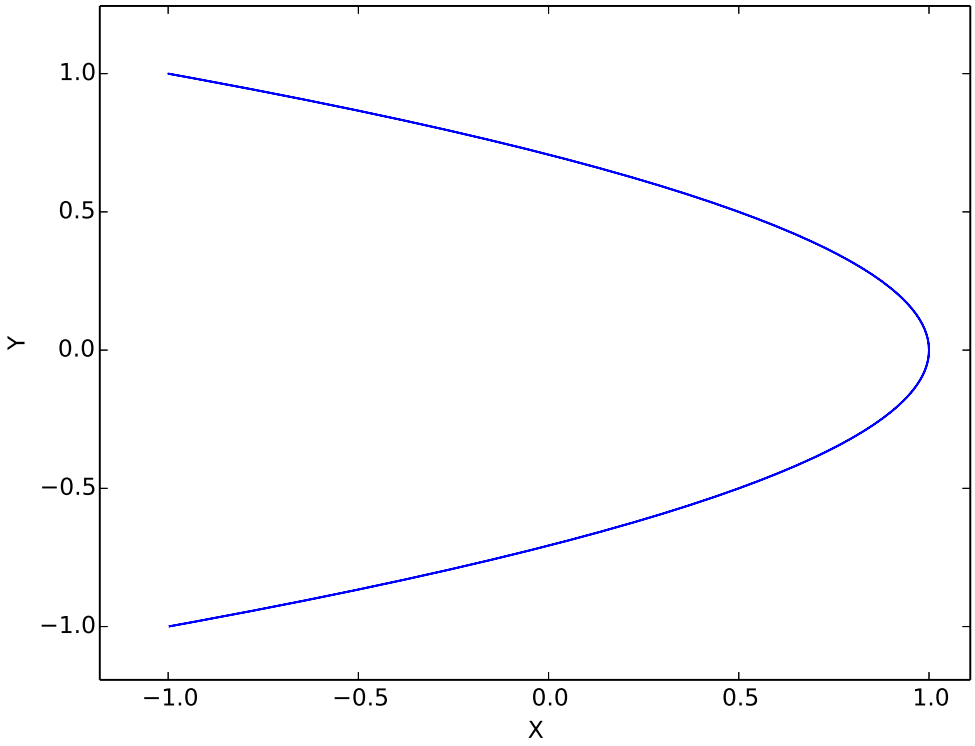


Figure 2: Lissajous figure with $f_X = 2$, $f_Y = 1$, and $\phi = 0$.

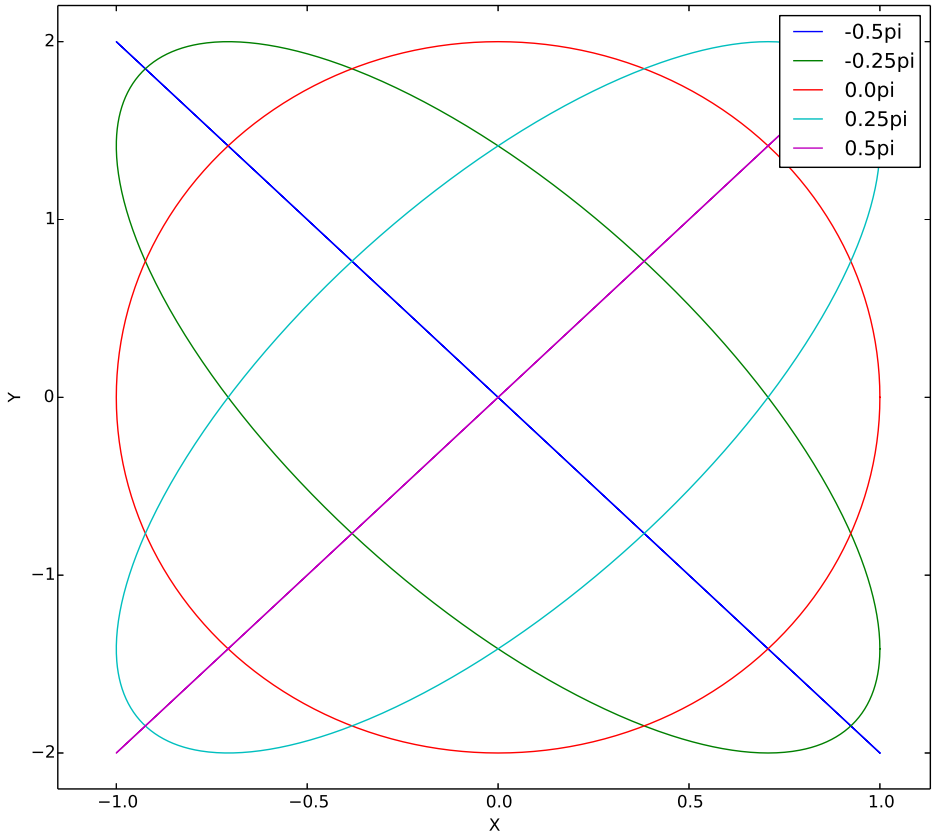


Figure 3: Lissajous figures with varying ϕ values.

```
from computeXYZ_numpy import *
import matplotlib.pyplot as plotter

# predetermined values
A_X = 1
A_Y = 1
phi = 0.5
dt = 0.001
N = 50000

def plotZ(f_X, f_Y, fourier=False):
    '''Given f_X, f_Y, and predetermined values, plot Z vs. t. If fourier
    variable is True, plot Z(f) vs. f.'''
    (t, X, Y, Z) = calctXYZ(f_X, f_Y, A_X, A_Y, phi, dt, N)
    if fourier:
        assert t.size == N + 1
        f = np.fft.fftfreq(t.size, d=dt) # frequency samples
        Zh = np.fft.fft(Z)
        plotter.plot(f, Zh.real, f, Zh.imag)
        plotter.xlabel('f')
        plotter.ylabel('Zh')
    else:
        plotter.plot(t, Z)
        plotter.xlabel('t')
        plotter.ylabel('Z')
    plotter.show()
```

We use $f_X = 1.1$ and $f_Y = 1$. The periods corresponding to ω_1 and ω_2 are given by:

$$T\left(\frac{\omega_1+\omega_2}{2}\right) = \frac{4\pi}{2\pi f_X + 2\pi f_Y} = \frac{2}{f_X+f_Y} = \frac{2}{1.1+1} = 0.952381$$
$$T(\omega_1-\omega_2) = \frac{2\pi}{2\pi f_X - 2\pi f_Y} = \frac{1}{0.1} = 10$$

Figure 4 demonstrates many cycles with period 0.952381, and modulation cycles with period 10. As expected, figure 5 exhibits a peak at each of the component frequencies, $f_X = 1.1$ and $f_Y = 1$.

3.1 Symmetry as an explanation of modulation frequency doubling

We explore why the modulation frequency is twice what is expected. In a generic case, suppose we have two sinusoids $\cos(\omega_1 t)$ and $\cos(\omega_2 t)$. We know that the sum of these two sinusoids is given by:

$$\cos(\omega_1 t) + \cos(\omega_2 t) = 2\cos\left(\frac{\omega_1+\omega_2}{2}t\right)\cos\left(\frac{\omega_1-\omega_2}{2}t\right).$$

We can interpret this as a single sinusoid $Z(t) = A\cos\left(\frac{\omega_1+\omega_2}{2}t\right)$, with the changing amplitude given by $A(t) = 2\cos\left(\frac{\omega_1-\omega_2}{2}t\right)$. Since the amplitude is expressed as a cos function, the amplitude oscillates around 0, i.e. $A < 0$ half the time. "Negative amplitude" doesn't intuitively mean much to us; we can interpret this as simply flipping the sinusoid $Z(t)$ about the x-axis when the amplitude is negative. However, this is

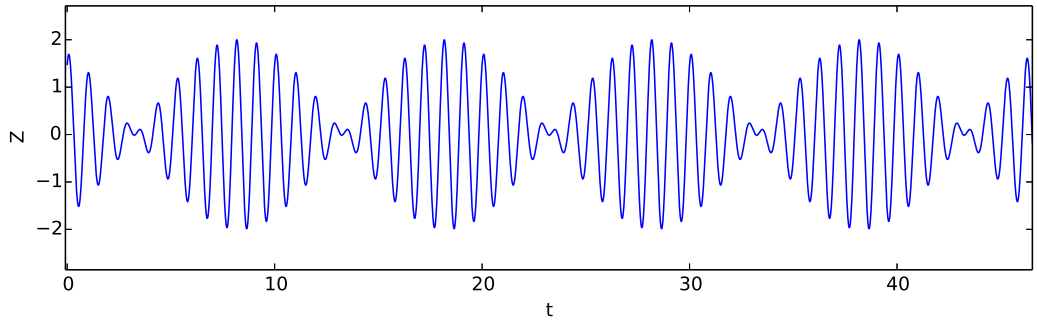


Figure 4: $Z(t)$ vs. t, with $f_X = 1.1$ and $f_Y = 1$.

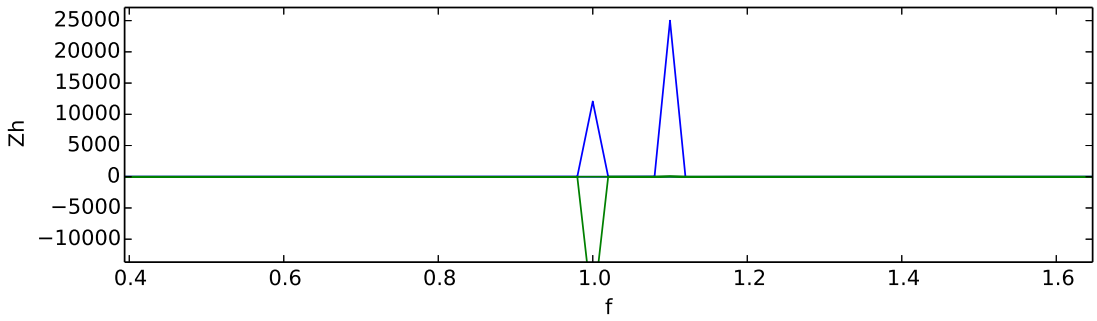


Figure 5: $\hat{Z}(f)$ vs. f, showing component frequencies $f_X = 1.1$ and $f_Y = 1$.

not visible because $\cos\left(\frac{\omega_1+\omega_2}{2}t\right)$ is itself a symmetric sinusoid, and flipping the curve about the x-axis twice results in the original curve.

Thus, we can say that the amplitude is given by $A(t) = \left|2\cos\left(\frac{\omega_1-\omega_2}{2}t\right)\right|$, which has half the period (i.e. twice the frequency), so the amplitude modulation frequency is simply $\omega_1 - \omega_2$.

4 Programming Reflections

I've programmed in C and Python before. Python is more high-level, so it's a lot more convenient for numerical applications such as these. Guido van Rossum's description of Python touts its "efficient high-level data structures," "elegant syntax and dynamic typing," features which make Python code simpler and shorter. However, Python is notably inefficient. For example, Python 2's `range` function returns an actual list, so using it to loop through the indices of a string (e.g. `for i in range(len(string))`) requires storing the entire list in memory. In C, we can just store a counter variable: `for (i = 0, i < strlen(string), i++)`.

`numpy` has the added advantage of data structures designed specifically for numerical analysis. Since we deal with operations on arrays so often, it's very convenient to be able to operate on elements of a `np.array` object by treating the array name as a variable, while looping through the elements of the array is done behind-the-scenes and efficiently with C.