Wien Bridge Oscillator - Table of Contents

Electronic Oscillator Design: the Wien bridge oscillator and the dynamics of self-sustained oscillations

1 Introduction

This document models the circuit behavior of an audio-frequency Wien bridge oscillator. The goal is to examine this system as a practical implementation of important ideas in nonlinear dynamics. These include concepts of linear instability, bifurcations (including the Hopf bifurcation), limit cycles, the Hopf equation, van der Pol equation and related amplitude equations, the dependence of amplitude on control parameters, and critical behavior at the onset of oscillation.

This sets the foundation for using this circuit as a model for more general classes of nonlinear self-sustained oscillation (including physiological examples, such as models of the inner ear), and as a building-block for more complex systems such as oscillator networks that show a variety of synchronization behaviors.

A companion document will describe actual circuit construction and testing.

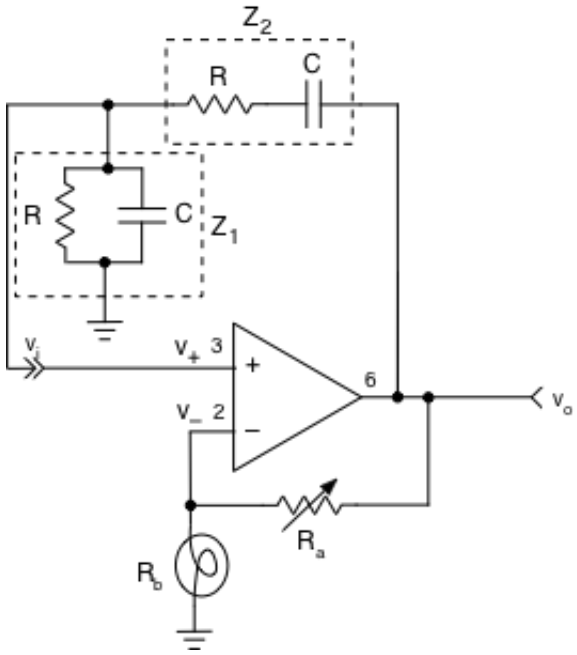
1.1 Here is one type of Wien bridge oscillator

In this picture, a crane assists with construction of the Suedbahnhofbruecke in Vienna (Wien): a "Wien bridge." We are familiar with the pendulum-like swinging of the load hanging from the crane: one hopes this swinging is quickly stopped and the load safetly transfered. But in some cases we want oscillators to keep running indefinitely: how is this done? (How, for example, is it done for the pendulum of a grandfather clock?)

By GuentherZ (Own work) [CC BY 3.0 (http://creativecommons.org/licenses/by/3.0)] (http://creativecommons.org/licenses/by/3.0)%5D), via Wikimedia Commons from Wikimedia Commons.

1.2 Here is the Wien bridge oscillator that we want to understand

A resistor-capacitor network that are two arms of a so-called "Wien bridge" feed the output of an amplifier back to its input. Under what conditions will this feedback produce self-sustained oscillation? What governs the amplitude and spectral content of the oscillation?

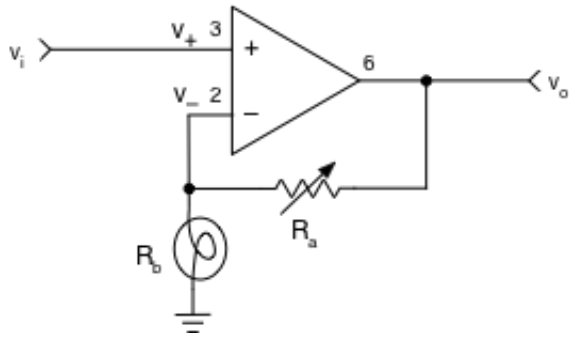


2 Conventional analysis of electronic oscillators

2.1 Barkhausen criterion

In this section, we'll establish a self-consistency criterion for oscillation to occur.

Consider the amplifier shown below.



The potential at the noninverting input is equal to the input signal:

$$v_+ = v_i.$$

The potential at the inverting input is related to the output via a voltage-divider:

$$v_- = \frac{R_b}{R_a + R_b} v_o.$$

Using the ideal op-amp assumption that $v_+ = v_-$ leads to

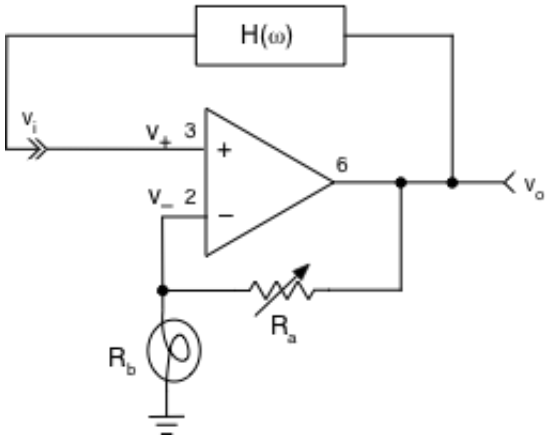
$$v_o = G v_i$$

where

$$G \equiv 1 + \frac{R_a}{R_b}.$$

This is the standard non-inverting amplifier configuration.

Now let's ask if we can sustain a finite output if, instead of an external input, we feed the output back to the input through a frequency-dependent network.



Using complex phasor notation such that $v = \mathbb{R}\hat{V}e^{j\omega t}$ (where $\mathbb{R}$ means "real part of") , we write

$$\widehat{V}_i = H(\omega)\widehat{V}_o.$$

We also have

$$\widehat{V}_o = G(\omega)\widehat{V}_i.$$

(Note: we will assume below that G does not have any frequency dependence because the oscillator is designed to operate within the gain-bandwidth limits of the op amp.)

Self-consistency requires

$$\widehat{V}_i = G(\omega)H(\omega)\widehat{V}_i.$$

This requires that the loop gain

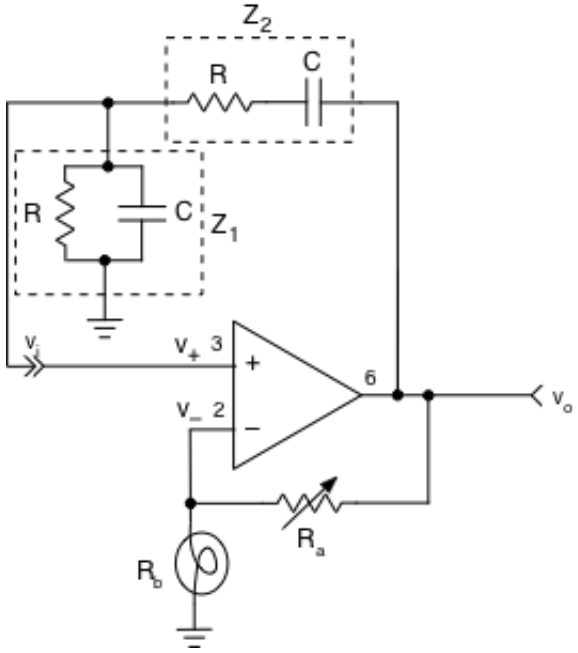
$$G(\omega)H(\omega) = 1.$$

This is the Barkhausen criterion for oscillation, which requires that both that the magnitude of the loop gain is unity and that the phase shift is zero or a multiple of 2π .

2.2 Application to the Wien Bridge oscillator circuit

2.2.1 The Feedback Transfer Function

Consider the resistance - capacitance network shown below. The network is composed of two arms of a so-called Wien bridge (https://en.wikipedia.org/wiki/Wien_bridge).



The complex impedances of the series and parallel combinations of resistor and capacitor are respectively:

$$Z_1 = \left(\frac{1}{R} + j\omega C\right)^{-1} = \frac{R}{1 + j\omega RC}.$$

$$Z_2 = R + \frac{1}{j\omega C} = R\left(1 - j\frac{1}{\omega RC}\right).$$

Here we use the simple j for $\sqrt{-1}$.

The complex impedances form a voltage divider, giving a relationship between the input and output voltages:

$$\widehat{V}_i = \frac{Z_1}{Z_1 + Z_2}\widehat{V}_o.$$

$$\widehat{V}_i = \frac{\frac{R}{1+j\omega RC}}{\frac{R}{1+j\omega RC} + R(1 - j\frac{1}{\omega RC})}\widehat{V}_o.$$

$$\widehat{V}_i = \frac{1}{3 + j\left(\omega RC - \frac{1}{\omega RC}\right)}\widehat{V}_o.$$

Thus

$$H(\omega) = \frac{1}{3 + j\left(\omega RC - \frac{1}{\omega RC}\right)}.$$

Define a characteristic frequency

$$\omega_0 \equiv \frac{1}{RC}.$$

After some algebraic manipulation, the magnitude and phase of $H(\omega)$ are given by:

$$|H(\omega)| = \frac{\frac{\omega}{\omega_0}}{\sqrt{9(\frac{\omega}{\omega_0})^2 + \left[(\frac{\omega}{\omega_0})^2 - 1\right]^2}}$$

$$\phi(\omega) \equiv \angle H(\omega) = \arctan\left[\frac{1}{3}\left(\frac{\omega_0}{\omega} - \frac{\omega}{\omega_0}\right)\right]$$

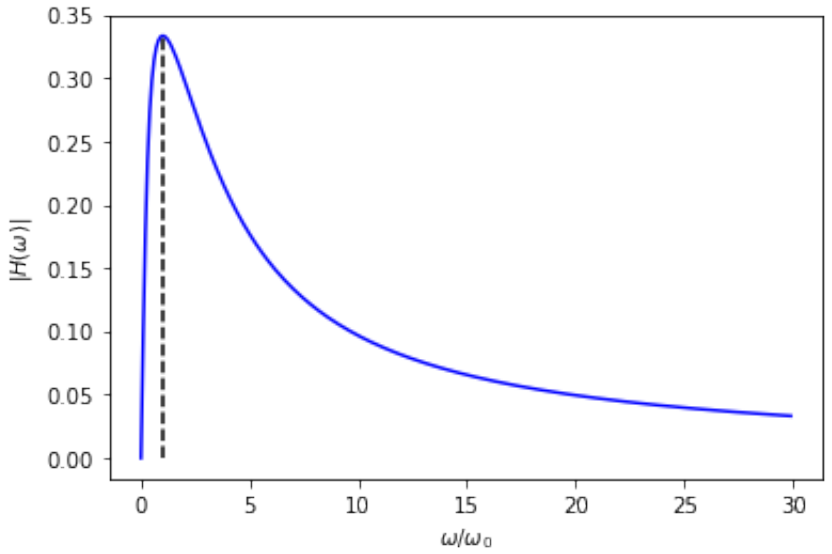
2.2.2 Computing the magnitude and phase of the feedback transfer function

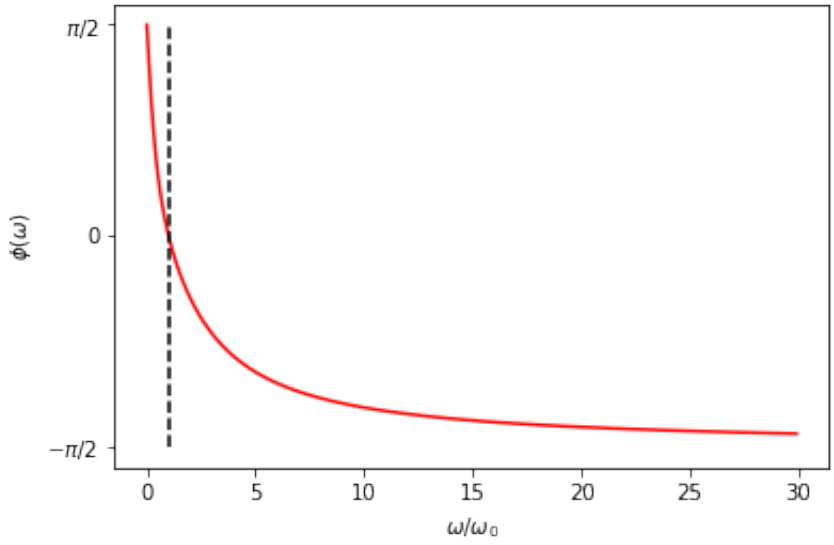
The following computation shows that $|H(\omega)|$ starts at 0, rises to a peak equal to 1/3 when $\omega/\omega_0 = 1$ (dashed line) and then falls to 0 again. The phase starts at $\frac{\pi}{2}$, goes to 0 when $\omega/\omega_0 = 1$ (dashed line), and levels out at $-\frac{\pi}{2}$.

In [1]:
```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

In [2]:
```
import math
import numpy as np
import matplotlib.pyplot as plt

a_r = [0]
a_H = [0]
a_phi = [math.pi/2]

# r is the ratio omega/omega_0
for tenr in range(1,300):
    r=0.1*tenr
    H = r/math.sqrt(9*r**2+(r**2-1)**2)
    a_r = np.append(a_r,r)
    a_H = np.append(a_H,H)
    a_phi = np.append(a_phi,math.atan((1/r-r)/3.))

plt.figure(0)
plt.plot(a_r,a_H,'b',[1,1],[0,1./3],'k--')
plt.xlabel('$\omega/\omega_0$')
plt.ylabel('$|H(\omega)|$')

plt.figure(1)
plt.plot(a_r,a_phi,'r',[1,1],[-math.pi/2,math.pi/2],'k--')
plt.xlabel('$\omega/\omega_0$')
plt.ylabel('$\phi(\omega)$')
plt.yticks(np.arange(-math.pi/2,1.6,math.pi/2),('$-\pi/2$','0','$\pi/2$'
```

```
plt.show()
```


0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
$|H(\omega)|$
0
5
10
15
20
25
30
ω/ω_0


$\pi/2$
0
$-\pi/2$
$\phi(\omega)$
0
5
10
15
20
25
30
ω/ω_0

2.2.3 Asserting the Barkhausen criteria

Remember that

$$\widehat{V}_o = G\widehat{V}_i$$

with G having the real value

$$G = 1 + \frac{R_a}{R_b}$$

.

So the loop gain requirement

$$GH(\omega) = 1$$

becomes

$$\frac{G}{3 + j\left(\omega RC - \frac{1}{\omega RC}\right)} = 1.$$

This is true provided

$$G = 3.$$

$$\omega = \frac{1}{RC} \equiv \omega_0.$$

These then are the conditions for oscillation with the RC network forming $H(\omega)$.

Note that the above conditions then require

$$\frac{R_a}{R_b} = 2.$$

In the practical circuit, we will select resistor R_a so that it can be varied until sufficient gain is reached to start oscillations. We consider

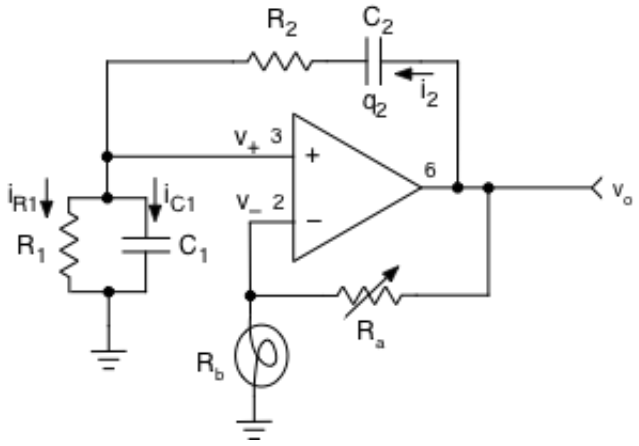
$$\epsilon \equiv \frac{R_a}{R_{b0}} - 2$$

to be the *bifurcation parameter* of the system. (Here the subscript 0 on R_{b0} represents a value for the resistance R_b in the limit of vanishing current. We will see below how a change in R_b with non-zero current plays a role in the nonlinear stabilization of the oscillator.)

3 Time-Domain Analysis

The above criteria are useful in creating the oscillator design. However, insight into the oscillator dynamics - leading to an understanding of what controls the amplitude and shape of oscillation - requires a time-domain analysis.

Consider the same circuit, re-drawn below with currents and voltages indicated at various points in the circuit.



3.1 Linear circuit analysis

We assume that all components have constant values. As noted above, the analysis will later be generalized to allow the gain-determing resistor R_b to increase with potential drop across the resistor due to self-heating, introducing nonlinearity into the system and thereby limiting the amplitude of oscillation.

3.1.1 Signal at the non-inverting input

Let's first first sum potential drops across the $R_2 C_2$ branch.

$$v_+ + i_2 R_2 + \frac{1}{C_2} q_2 = v_o.$$

Differentiate this equation and use the fact that $i_2 = dq_2/dt$:

$$\frac{dv_+}{dt} + R_2 \frac{di_2}{dt} + \frac{1}{C_2} i_2 = \frac{dv_o}{dt}.$$

Now from Kirchoff's Law for the sum of currents into a junction, using the ideal op amp behavior that zero current flows into the non-inverting (+) input:

$$i_2 = i_{R1} + i_{C1}.$$

Thus

$$i_2 = \frac{1}{R_1} v_+ + C_1 \frac{dv_+}{dt}.$$

Inserting this into the equation with dv_+/dt above, we get:

$$\frac{dv_+}{dt} + R_2 \frac{d}{dt}\left(\frac{1}{R_1} v_+ + C_1 \frac{dv_+}{dt} \right) + \frac{1}{C_2}\left(\frac{1}{R_1} v_+ + C_1 \frac{dv_+}{dt} \right) = \frac{dv_o}{dt}.$$

Collecting terms:

$$\left[R_2 C_1 \frac{d^2}{dt^2} + \left(1 + \frac{R_2}{R_1} + \frac{C_1}{C_2} \right) \frac{d}{dt} + \frac{1}{R_1 C_2} \right] v_+ = \frac{dv_o}{dt}.$$

3.1.2 Signal at the inverting input

At the inverting input we have

$$v_- = \frac{R_b}{R_a + R_b} v_0.$$

$$v_- = \frac{1}{1 + \frac{R_a}{R_b}} v_0.$$

$$v_- = \frac{1}{G} v_0,$$

where the gain is given by

$$G \equiv 1 + \frac{R_a}{R_b}.$$

3.1.3 Combined results

Now use again use ideal op-amp behavior to require that

$$v_+ = v_-,$$

so that in the above differential equation we can substitute for v_+ using

$$v_+ = v_- = \frac{1}{G} v_0.$$

Thus we re-write the original differential equation entirely in terms of v_0:

$$\left[R_2 C_1 \frac{d^2}{dt^2} + \left(1 + \frac{R_2}{R_1} + \frac{C_1}{C_2} \right) \frac{d}{dt} + \frac{1}{R_1 C_2} \right] \frac{1}{G} v_o = \frac{dv_o}{dt}.$$

Multiply through by G and re-arrange terms:

$$\frac{d^2 v_o}{dt^2} + \left(1 + \frac{R_2}{R_1} + \frac{C_1}{C_2} - G\right) \frac{1}{R_2 C_1} \frac{dv_o}{dt} + \frac{1}{(R_1 C_2)(R_2 C_1)} v_o = 0.$$

If we match resistors so that $R_1 = R_2 \equiv R$ and capacitors so that $C_1 = C_2 \equiv C$ then this simplifies to:

$$\frac{d^2 v_o}{dt^2} + (3 - G)\frac{1}{RC}\frac{dv_o}{dt} + \frac{1}{(RC)^2} v_o = 0.$$

3.2 Solving the linear equation and obtaining characteristic exponents

Again, assuming that all of the coefficients are constant, we can insert a soluton $v_o = V_o e^{st}$. This leads to the characteristic equation for the exponent s,

$$s^2 + (3 - G)\frac{1}{RC} s + \frac{1}{(RC)^2} = 0.$$

The solution is

$$s \equiv \sigma + i\omega = \frac{(G - 3) \pm \sqrt{(G - 3)^2 - 4}}{2} \omega_0 .$$

where $\omega_0 \equiv 1/RC$. The values are complex for $1 < G < 5$ and the solution grows exponentially when $G > 3$. Of course, indefinite exponential growth is not possible; eventually the power supply limits are reached. However, if an introduction of nonlinearity into the circuit causes the gain to diminish with growing amplitude (see below), the oscillations will saturate to a fixed amplitude. In such a case we have a bifurcation to limit cycle behavior at $G = 3$. This occurs when

$$\frac{R_a}{R_b} = 2,$$

for which case

$$s = \pm j\omega_0 .$$

Note that we can re-write the differential equation (flipping the middle term to conform better with the form of the Van der Pol equation - see below) as:

$$\frac{d^2 v_o}{dt^2} - \left(\frac{R_a}{R_b} - 2\right) \omega_0 \frac{dv_o}{dt} + \omega_0^2 v_o = 0.$$

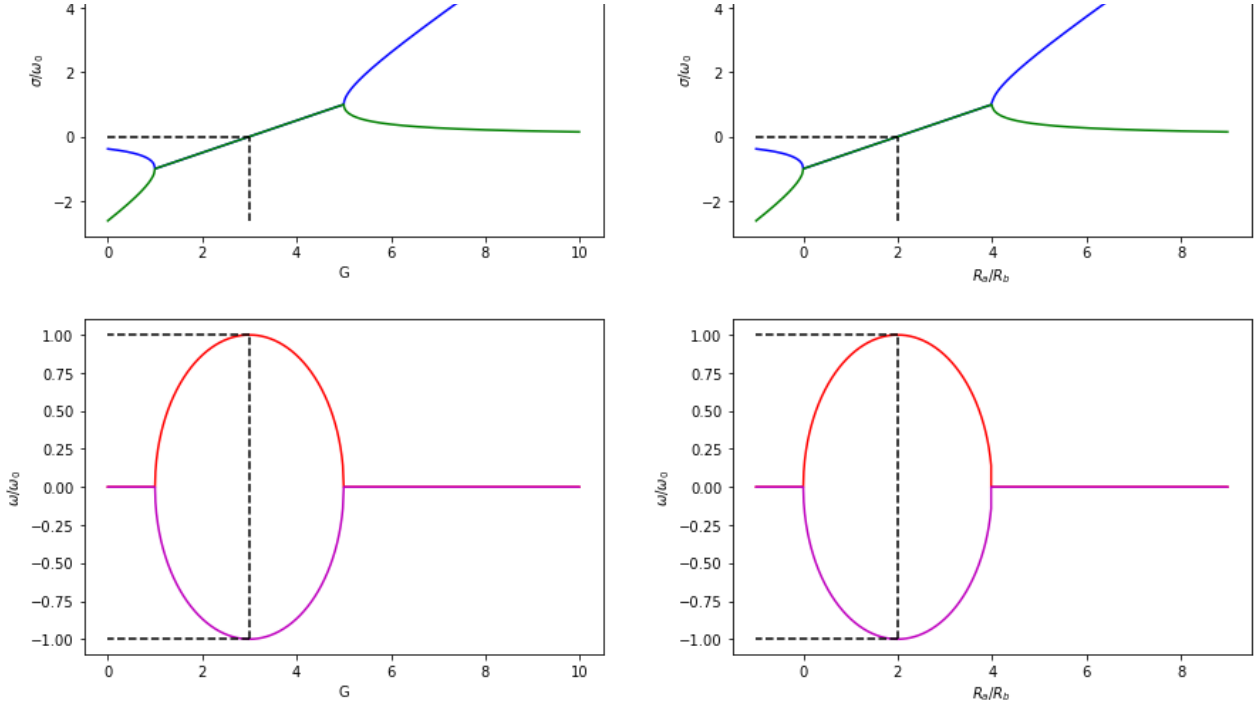
3.3 Plotting the real and imaginary parts of the characteristic exponent (eigenvalue) spectrum as a function of gain parameters

The following plots growth rate and frequency versus gain (first pair of plots) and versus negative feedback resistance ratio (second pair of plots.) We see that the condition of neutral stability occurs when the real part of the exponent passes through zero: $\sigma = 0$. Here oscillation exists but it neither grows nor decays, occurs when $G = 3$ and thus when $R_a/R_b = 2$. At this condition the frequency $\omega = \omega_0 \equiv 1/\sqrt{RC}$. (See the dashed lines in the plots that identify these conditions.)

Note: we will see below in a matrix formulation of the dynamical problem why we also refer to the values of s as "eigenvalues".

In [2]:
```
import numpy as np
import math
import matplotlib.pyplot as plt

a_G = [] #array of gain parameter values
a_RR = [] #array of resistor ratio Ra/Rb where G = 1+Ra/Rb
a_srpos = [] #array of positive real part of characteristic exponent (eig
a_srneg = [] #array of negative real part of characteristic exponent (eig
a_sipos = [] #array of positive imaginary part of characteristic exponent
a_sineg = [] #array of negative imaginary part of characteristic exponent

# Compute the eigenvalue spectrum for the range 0<G<1
for hunG in range(0,99):
    G = hunG/100.
    a_G = np.append(a_G,G)
    a_RR = np.append(a_RR,G-1.)
    a_srpos = np.append(a_srpos,((G-3)+math.sqrt((G-3.)**2-4))/2.)
    a_srneg = np.append(a_srneg,((G-3)-math.sqrt((G-3.)**2-4))/2.)
    a_sipos = np.append(a_sipos,0.)
    a_sineg = np.append(a_sineg,0.)
# Note that the values of Ra/Rb are negative for G<1, which is unrealizab

# Compute the eigenvalues for G=1
a_G = np.append(a_G,1.)
a_RR = np.append(a_RR,0.)
a_srpos = np.append(a_srpos,-1.)
a_srneg = np.append(a_srneg,-1.)
a_sipos = np.append(a_sipos,0.)
a_sineg = np.append(a_sineg,0.)

# Compute the characteristic exponent (eigenvalue) spectrum for the range
for hunG in range(101,499):
    G = hunG/100.
    a_G = np.append(a_G,G)
    a_RR = np.append(a_RR,G-1.)
    a_srpos = np.append(a_srpos,(G-3)/2.)
    a_srneg = np.append(a_srneg,(G-3)/2.)
    a_sipos = np.append(a_sipos,math.sqrt(4-(G-3.)**2)/2.)
    a_sineg = np.append(a_sineg,-math.sqrt(4-(G-3.)**2)/2.)

# Compute the characteristic exponent (eigenvalue) for G=5
a_G = np.append(a_G,5.)
```

```
a_RR = np.append(a_RR,G-1.)
a_srpos = np.append(a_srpos,1.)
a_srneg = np.append(a_srneg,1.)
a_sipos = np.append(a_sipos,0.)
a_sineg = np.append(a_sineg,0.)

# Compute the characteristic exponent (eigenvalue) spectrum for the range
for hunG in range(501,1000):
    G = hunG/100.
    a_G = np.append(a_G,G)
    a_RR = np.append(a_RR,G-1.)
    a_srpos = np.append(a_srpos,((G-3)+math.sqrt((G-3.)**2-4))/2.)
    a_srneg = np.append(a_srneg,((G-3)-math.sqrt((G-3.)**2-4))/2.)
    a_sipos = np.append(a_sipos,0.)
    a_sineg = np.append(a_sineg,0.)

plt.figure(1)

# Note that black dashed lines in these figures indicate the critical val
# and corresponding characteristic exponents (eigenvalues).

plt.subplot(2,2,1)
plt.plot(a_G,a_srpos,'b',a_G,a_srneg,'g',[0,3],[0,0],'k--',[3,3],[-2.618,
plt.xlabel('G')
plt.ylabel('$\sigma/\omega_0$')

plt.subplot(2,2,3)
plt.plot(a_G,a_sipos,'r',a_G,a_sineg,'m',[3,3],[-1,1],'k--',[3,0],[1,1],'
plt.xlabel('G')
plt.ylabel('$\omega/\omega_0$')

# Again note that the values of Ra/Rb are negative for G<1, which is unre

plt.subplot(2,2,2)
plt.plot(a_RR,a_srpos,'b',a_RR,a_srneg,'g',[-1,2],[0,0],'k--',[2,2],[-2.6
#plt.plot(a_RR,a_srpos[99:1001],'b',a_RR,a_srneg[99:1001],'g',[0,2],[0,0]
plt.xlabel('$R_a/R_b$')
plt.ylabel('$\sigma/\omega_0$')

plt.subplot(2,2,4)
plt.plot(a_RR,a_sipos,'r',a_RR,a_sineg,'m',[2,2],[-1,1],'k--',[2,-1],[1,1
#plt.plot(a_RR,a_sipos[99:1001],'r',a_RR,a_sineg[99:1001],'m',[2,2],[-1,1
plt.xlabel('$R_a/R_b$')
plt.ylabel('$\omega/\omega_0$')

plt.subplots_adjust(top=1.92, bottom=0.08, left=0.10, right=2.0, hspace=0
                    wspace=0.25)

plt.show()
```



4 Phase plane interpretation of linear behavior

4.1 Conversion of the dynamics to a system of first-order differential equations

Consider again the second-order time-domain differential equation for the output voltage v_o. Note: for simplicity, we will drop the subscript o and simply use v to denote the output voltage.

$$\frac{d^2v}{dt^2} - \left(\frac{R_a}{R_b} - 2\right)\omega_0 \frac{dv}{dt} + \omega_0^2 v = 0.$$

Let us re-write this equation in scaled form, defining $\epsilon \equiv R_a/R_b - 2$ and $\tilde{t} \equiv \omega_0 t$.

$$\frac{d^2v}{d\tilde{t}^2} - \epsilon\frac{dv}{d\tilde{t}} + v = 0.$$

In preparation for developing nonlinear models, we will convert this into a dynamical system of coupled first-order equations. There are various ways to do this. The simplest approach is to define a new variable $w = dv/d\tilde{t} = (1/\omega_0)dv/dt$. Then an equivalent system of equations is:

$$\frac{dv}{d\tilde{t}} = w$$
$$\frac{dw}{d\tilde{t}} = -v + \epsilon w.$$

4.2 Numerical solution of the linear dynamical model

Numerically integrating this system from a specific initial condition and plotting the trajetory (v(t), w(t)) provides a "phase portrait" of the dynamics. One can try different initial conditions to see how the phase portraits vary.

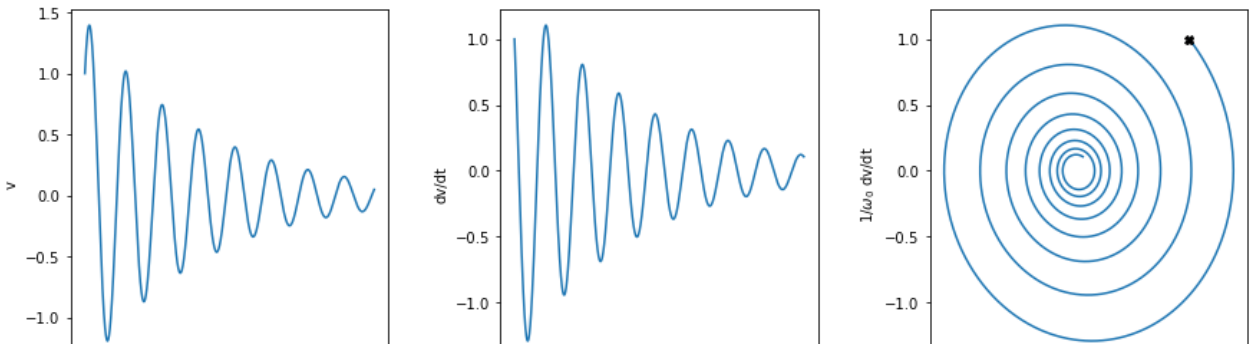
We see three distinct cases in these phase portraits computed and plotted in the code below. For $\epsilon < 0$ the trajectory spirals inwards to (u,v) = (0,0) from any initial condition. For $\epsilon = 0$ the trajectory is a closed cycle, just like that obtained for a frictionless pendlum plotting angular velocity versus angle. Different closed cycles are obtained for different initial conditions. For $\epsilon > 0$ the trajectory spirals outward, towards infinity in the linear model.

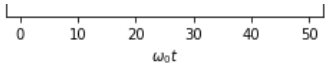
Of course, a real system will eventually be amplitude limited and our goal later in this document is to find ways to rationally design a predictable amplitude into the dynamics using nonlinear behavior. We will also find that after nonlinearity is introduced, the system will select (for $\epsilon > 0$) a unique orbit – a "limit cycle" – for a broad range of initial conditions.

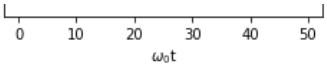
In [4]:
```
% matplotlib inline
# Code to numerically integrate the actual dynamical system
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math
#
# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt,epsilon):
    ff1 = xx[1]
    ff2 = -xx[0]+epsilon*xx[1]
    return [ff1,ff2]
#
timestep = 0.01
timerange = 50
Nstep = int(timerange/timestep)

for m in range(1,4):
    # Initial conditions
    v_0 = 1.0
    w_0 = 1.0
    #
    epsilon = -0.1 + (m-1)* 0.1 # specify epsilon directly
    print('epsilon = ',epsilon)
    #
    a_tt = np.arange(0,timerange,timestep) # time array in dimensional u
    a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))
    vv = a_sol[0:Nstep,0]
    ww = a_sol[0:Nstep,1]
    if m == 2:  # for epsilon = 0, compute multiple solutions from diffe
        v_0 = .75
        w_0 = .75
        a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))
```

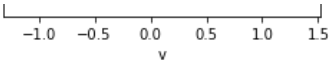
```
        vv75 = a_sol[0:Nstep,0]
        ww75 = a_sol[0:Nstep,1]
        v_0 = .5
        w_0 = .5
        a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))
        vv50 = a_sol[0:Nstep,0]
        ww50 = a_sol[0:Nstep,1]
        v_0 = .25
        w_0 = .25
        a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))
        vv25 = a_sol[0:Nstep,0]
        ww25 = a_sol[0:Nstep,1]


    plt.figure(m)
    plt.subplot(1,3,1)
    plt.plot(a_tt,vv)
    plt.xlabel('$\omega_0t$')
    plt.ylabel('v')

    plt.subplot(1,3,2)
    plt.plot(a_tt,ww)
    plt.xlabel('$\omega_0$t')
    plt.ylabel('dv/dt')

    #plt.annotate('eps = {0:4.2f}'.format(epsilon),xy=(5,0.75))
    #
    plt.subplot(1,3,3)
    if m == 2:
        plt.plot(vv,ww,vv75,ww75,vv50,ww50,vv25,ww25,[1.],[1.],'kX',[.75
    else:
        plt.plot(vv,ww,[v_0],[w_0],'kX')
    plt.xlabel('v')
    plt.ylabel('$1/\omega_0$ dv/dt')
    #plt.annotate('eps = {0:4.2f}'.format(epsilon),xy=(5,0.75))

    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspa
                        wspace=0.35)
    plt.show()
```

```
epsilon =  -0.1
```

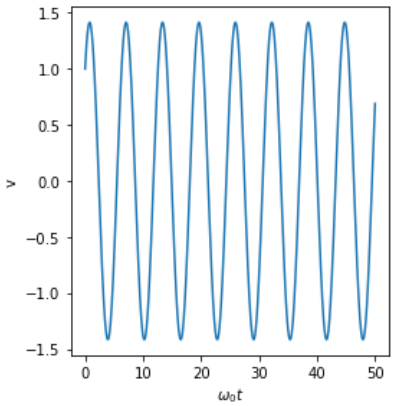


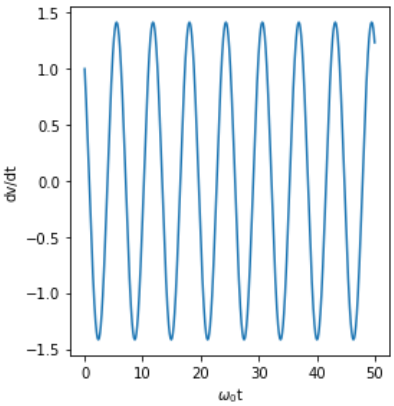


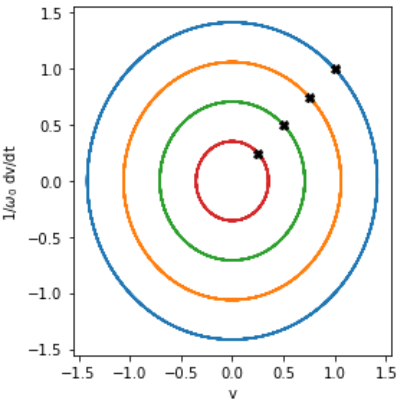




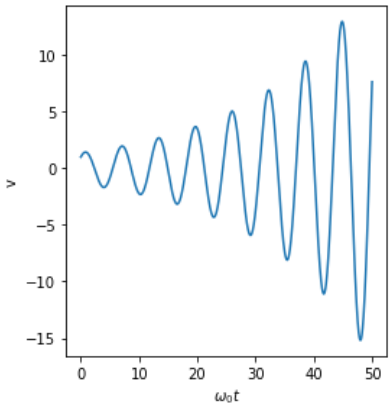
epsilon = 0.0

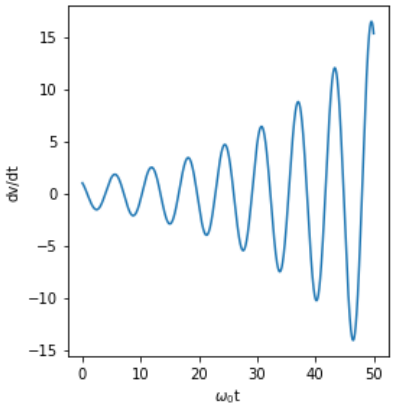


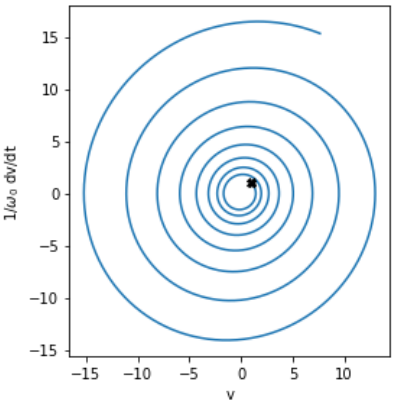




epsilon = 0.1







4.3 Analytical solution of the linear dynamical model

Recall

$$\frac{dv}{d\tilde{t}} = w$$
$$\frac{dw}{d\tilde{t}} = -v + \epsilon w.$$

In matrix form, this becomes

$$\frac{d}{d\tilde{t}}\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & \epsilon \end{pmatrix}\begin{pmatrix} v \\ w \end{pmatrix}$$

Trying a solution

$$\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} \widehat{v} \\ \widehat{w} \end{pmatrix} e^{\tilde{s}\tilde{t}}$$

gives the eigenvalue equation

$$\begin{pmatrix} 0 & 1 \\ -1 & \epsilon \end{pmatrix} \begin{pmatrix} \widehat{v} \\ \widehat{w} \end{pmatrix} = \widetilde{s} \begin{pmatrix} \widehat{v} \\ \widehat{w} \end{pmatrix}.$$

The eigenvalue is obtained by solving the determinant equation:

$$\begin{vmatrix} -\widetilde{s} & 1 \\ -1 & \epsilon - \widetilde{s} \end{vmatrix} = 0.$$

which leads to the characteristic equation:

$$\widetilde{s}^2 - \epsilon\widetilde{s} + 1 = 0.$$

Solutions are

$$\widetilde{s}_{\pm} = \frac{\epsilon \pm \sqrt{\epsilon^2 - 4}}{2}.$$

Let's focus on the range $-2 < \epsilon < 2$. Then we can write

$$\widetilde{s}_{\pm} = \frac{\epsilon \pm i\sqrt{4 - \epsilon^2}}{2}.$$

The solution in actual time units is

$$\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} v_0 \\ w_0 \end{pmatrix} e^{st}.$$

Given that $\tilde{t} = \omega_o t$, we must have

$$e^{st} = e^{s\tilde{t}/\omega_0} = e^{\widetilde{s}\widetilde{t}}.$$

Consistency requires that

$$\widetilde{s} = \frac{s}{\omega_0},$$

and

$$s_{\pm} = \widetilde{s}_{\pm}\omega_0 = \frac{\epsilon \pm i\sqrt{4 - \epsilon^2}}{2}\omega_0.$$

Let's define

$$s_{\pm} \equiv \sigma \pm i\omega$$

.

Then

$$\sigma = \frac{\epsilon}{2}\omega_0$$
$$\omega = \frac{\sqrt{4-\epsilon^2}}{2}\omega_0 .$$

We must also identify the eigenvectors. They are given by:

$$\begin{pmatrix} \widehat{v}_\pm \\ \widehat{w}_\pm \end{pmatrix} = V \begin{pmatrix} 1 \\ \frac{\sigma}{\omega_0} \pm i\frac{\omega}{\omega_0} \end{pmatrix},$$

where V is some arbitrary voltage magnitude.

We can express any real initial condition as a linear combination of these eigenvectors. Since the eigenvectors are complex conjugates, the complex coefficients must also be complex conjugates in order to give a real result.

$$\begin{pmatrix} V_0 \\ W_0 \end{pmatrix} = (a+ib)\begin{pmatrix} 1 \\ \frac{\sigma}{\omega_0} + i\frac{\omega}{\omega_0} \end{pmatrix} + (a-ib)\begin{pmatrix} 1 \\ \frac{\sigma}{\omega_0} - i\frac{\omega}{\omega_0} \end{pmatrix}.$$

This leads to

$$a = \frac{V_0}{2}$$
$$b = \frac{V_0\sigma - W_0\omega_0}{2\omega} .$$

Then at a later time t:

$$\begin{pmatrix} v \\ w \end{pmatrix} = \left(\frac{V_0}{2} + i\frac{V_0\sigma - W_0\omega_0}{2\omega}\right)\begin{pmatrix} 1 \\ \frac{\sigma}{\omega_0} + i\frac{\omega}{\omega_0} \end{pmatrix} e^{\sigma t} e^{i\omega t} + \left(\frac{V_0}{2} - i\frac{V_0\sigma - W_0\omega_0}{2\omega}\right)\begin{pmatrix} \\ \frac{\sigma}{\omega_0} \end{pmatrix} .$$

Sorting and combining terms gives:

$$\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} V_0\cos(\omega t) - \left[V_0\frac{\sigma}{\omega} - W_0\frac{\omega_0}{\omega}\right]\sin(\omega t) \\ W_0\cos(\omega t) - \left\{V_0\frac{\omega}{\omega_0}\left[1+(\frac{\sigma}{\omega})^2\right] - W_0\frac{\sigma}{\omega}\right\}\sin(\omega t) \end{pmatrix} e^{\sigma t}$$

4.4 Comparison of numerical and analytical solutions

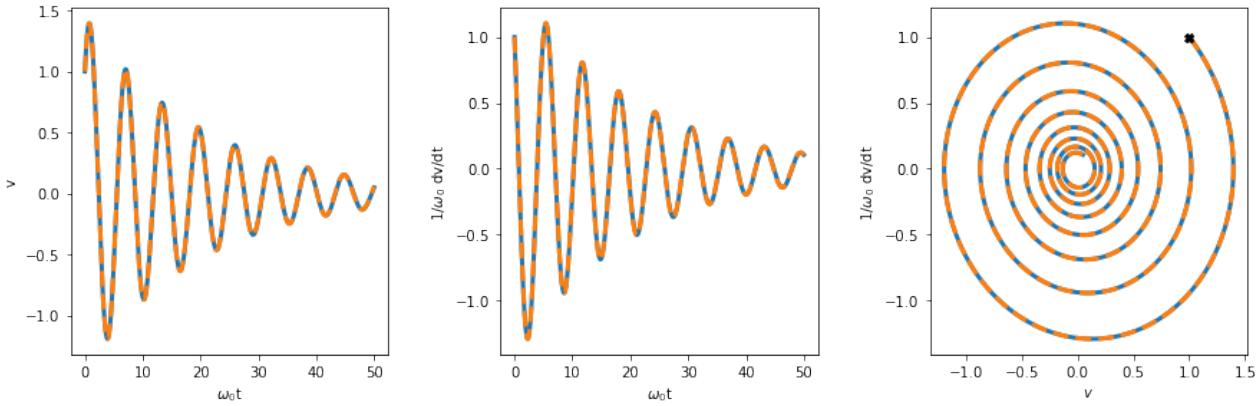
Here we plot both numerical and analytical solutions on the same axes, with a dashed curve for the analytical solution. We see that the plots overlie one another, indicating agreement.

In [5]:
```
% matplotlib inline
# Code to compate numerical and analytical solutions
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math
#
```

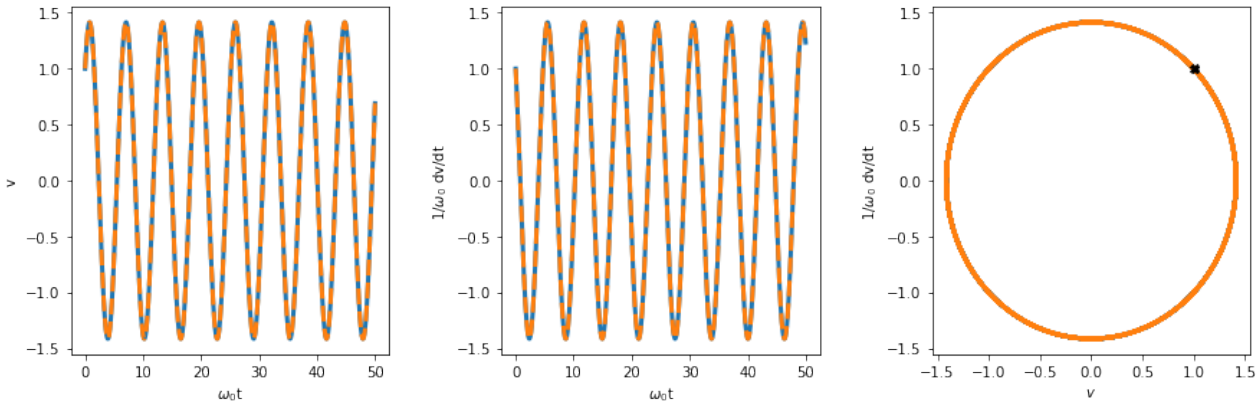
```
# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt,epsilon):
    ff1 = xx[1]
    ff2 = -xx[0]+epsilon*xx[1]
    return [ff1,ff2]
#
timestep = 0.01
timerange = 50
Nstep = int(timerange/timestep)

omega0 = 1.0

for m in range(1,4):
    # Initial conditions
    v_0 = 1.0
    w_0 = 1.0
    #
    epsilon = -0.1 + (m-1)* 0.1
    print('epsilon = ',epsilon)
    gamma = 1. # not actually used in this particular calculation.
    #
    a_tt = np.arange(0,timerange,timestep) # time array
    a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))
    vv = a_sol[0:Nstep,0]
    ww = a_sol[0:Nstep,1]

    sigma = 0.5*epsilon #in units of omega_0
    omega = 0.5*math.sqrt(4-epsilon**2) #in units of omega_0
    sigom = sigma/omega
    print('sigma = ',sigma)
    print('omega = ',omega)
    print('sigma/omega = ',sigom)


    vvAnalytic = v_0*np.cos(a_tt)-(v_0*sigom-w_0/omega)*np.sin(a_tt)
    wwAnalytic = w_0*np.cos(a_tt)-(v_0*omega*(1+sigom**2)-w_0*sigom)*np.s

    vvAnalytic = vvAnalytic*np.exp(sigma*a_tt)
    wwAnalytic = wwAnalytic*np.exp(sigma*a_tt)

    plt.figure(m)
    plt.subplot(1,3,1)
    #plt.plot(a_tt,vv)
    plt.plot(a_tt,vv,a_tt,vvAnalytic,'--',linewidth=3)
    plt.xlabel('$\omega_0$t')
    plt.ylabel('v')

    plt.subplot(1,3,2)
    #plt.plot(a_tt,ww)
    plt.plot(a_tt,ww,a_tt,wwAnalytic,'--',linewidth=3)
    plt.xlabel('$\omega_0$t')
    plt.ylabel('$1/\omega_0$ dv/dt')

    plt.subplot(1,3,3)
```

```
    plt.plot(vv,ww,vvAnalytic,wwAnalytic,'--',v_0,w_0,'kX',linewidth=3)
    #plt.plot(vv,ww)
    #plt.plot(vvAnalytic,wwAnalytic)
    plt.xlabel('$v$')
    plt.ylabel('$1/\omega_0$ dv/dt')

    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspa
                        wspace=0.35)
    plt.show()
```

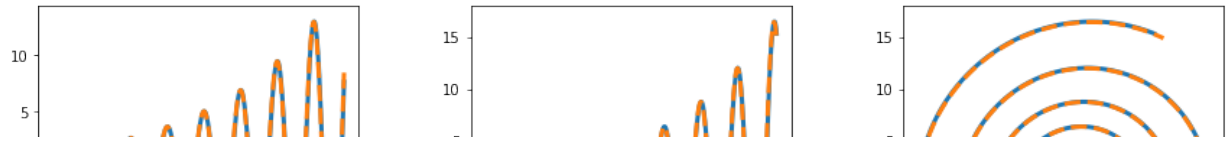
```
epsilon =  -0.1
sigma =  -0.05
omega =  0.998749217771909
sigma/omega =  -0.05006261743217589
```

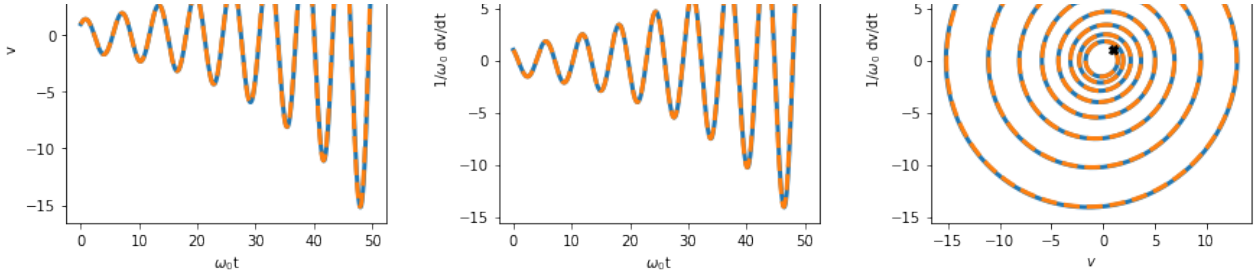



```
epsilon =  0.0
sigma =  0.0
omega =  1.0
sigma/omega =  0.0
```




```
epsilon =  0.1
sigma =  0.05
omega =  0.998749217771909
sigma/omega =  0.05006261743217589
```





5 Energy considerations

For frictionless pendulum motion, kinetic energy and potential energy are exchanged with one another twice per period. The same is true for an oscillating mass-spring system or for the exchange of energy between capacitor and inductor in an LC circuit. So can we find similar ideas of exchange of energy in the Wien bridge oscillator?

It turns out that the situation is more subtle, but at least we find that the energy stored in the capacitors is shifted in phase from capactior C_1 to capacitor C_2. The shift is by $\pi/2$ rather than π and the peak amount of energy stored in each is not the same. This will be seen in plots made at the end of the following series of derivations.

Another key result to be shown below is that the average power supplied by the amplifier to maintain oscillations is - assuming a periodic output voltage v_0 - simply

$$\overline{P_{\text{amp}}} = \frac{1}{G}\frac{\overline{v_0^2}}{R}.$$

If the output is sinusoidal with amplitude V_0, this then becomes

$$\overline{P_{\text{amp}}} = \frac{1}{G}\frac{1}{2}\frac{V_0^2}{R}.$$

Note also that the gain G at oscillation onset is 3. This power does not include power expended in any external load driven by the output or power dissipated in the gain network formed by resistors R_a and R_b.

NOTE: the rest of this section may be skipped on first reading. Later it might be fruitful to consider energy exchanges, particularly in thinking about possible design adaptations.

5.1 Stored energy

What is the energy stored in each capactor?

$$
\begin{aligned}
U_1 &= \frac{1}{2} C_1 v_+^2 \\
&= \frac{1}{G^2} \frac{1}{2} C v_o^2
\end{aligned}
$$

$$
\begin{aligned}
U_2 &= \frac{1}{2} C [v_o - (v_+ + i_2 R_2)]^2 \\
&= \frac{1}{2} C \left\{ v_o - [v_+ + (i_{R_1} + i_{C_1}) R_2] \right\}^2 \\
&= \frac{1}{2} C \left\{ v_o - \left[v_+ + \left(\frac{v_+}{R_1} + C_1 \frac{dv_+}{dt} \right) R_2 \right] \right\}^2 \\
&= \frac{1}{2} C \left[(G-2) v_+ - \frac{1}{\omega_0} \frac{dv_+}{dt} \right]^2 \\
&= \frac{1}{G^2} \frac{1}{2} C \left[(G-2) v_o - \frac{1}{\omega_0} \frac{dv_o}{dt} \right]^2
\end{aligned}
$$

Along the way, we assume that $R_1 = R_2 \equiv R$, $C_1 = C_2 \equiv C$, $\omega_0 \equiv \frac{1}{RC}$, and $v_o = G v_+$.

The total stored energy is

$$
\begin{aligned}
U &= \frac{1}{G^2} \frac{1}{2} C \left\{ v_o^2 + \left[(G-2) v_o - \frac{1}{\omega_0} \frac{dv_o}{dt} \right]^2 \right\} \\
&= \frac{1}{G^2} \frac{1}{2} C \left\{ \left[(G-2)^2 + 1 \right] v_o^2 + 2(G-2) \frac{1}{\omega_0} v_o \frac{dv_o}{dt} + \frac{1}{\omega_0^2} \left(\frac{dv_o}{dt} \right)^2 \right\}
\end{aligned}
$$

5.2 Rate of change of stored energy

The rate of change of energy of each capacitor is

$$
\begin{aligned}
\frac{dU_1}{dt} &= \frac{1}{G^2} C v_o \frac{dv_0}{dt} \\
&= \frac{1}{G^2}\frac{1}{R}\frac{1}{\omega_0} v_o \frac{dv_0}{dt}. \\
\frac{dU_2}{dt} &= \frac{1}{G^2} C \left[(G-2)v_o - \frac{1}{\omega_0}\frac{dv_o}{dt}\right]\left[(G-2)\frac{dv_o}{dt} - \frac{1}{\omega_0}\frac{d^2v_o}{dt^2}\right] \\
&= \frac{1}{G^2} C \left[(G-2)v_o - \frac{1}{\omega_0}\frac{dv_o}{dt}\right]\left[(G-3)\frac{dv_o}{dt} - \frac{1}{\omega_0}\frac{d^2v_o}{dt^2} + \frac{dv_o}{dt}\right] \\
&= \frac{1}{G^2} C \left[(G-2)v_o - \frac{1}{\omega_0}\frac{dv_o}{dt}\right]\left[\omega_0 v_0 + \frac{dv_o}{dt}\right] \\
&= \frac{1}{G^2} C \left[(G-2)v_0\left(\omega_0 v_0 + \frac{dv_0}{dt}\right) - \frac{1}{\omega_0}\left(\frac{dv_o}{dt}\right)^2 - v_0\frac{dv_0}{dt}\right] \\
&= \frac{1}{G^2}\frac{1}{R}\left[(G-2)v_0\left(v_0 + \frac{1}{\omega_0}\frac{dv_0}{dt}\right) - \frac{1}{\omega_0^2}\left(\frac{dv_o}{dt}\right)^2 - \frac{1}{\omega_0}v_0\frac{dv_0}{dt}\right].
\end{aligned}
$$

We used $C = RC/R = (1/R)(1/\omega_0)$. We also used the fact that the dynamical equation

$$
\frac{d^2v_o}{dt^2} + (3-G)\omega_o\frac{dv_o}{dt} + \omega_0^2 v_o = 0.
$$

can be re-arranged to give

$$
(G-3)\frac{dv_o}{dt} - \frac{1}{\omega_0}\frac{d^2v_o}{dt^2} = \omega_0 v_0.
$$

The rate of change of total stored energy is

$$
\begin{aligned}
\frac{dU}{dt} &= \frac{dU_1}{dt} + \frac{dU_2}{dt} \\
&= \frac{1}{G^2}\frac{1}{R}\left[(G-2)v_0\left(v_0 + \frac{1}{\omega_0}\frac{dv_0}{dt}\right) - \frac{1}{\omega_0^2}\left(\frac{dv_o}{dt}\right)^2\right].
\end{aligned}
$$

5.3 Dissipated power

$$
\begin{aligned}
P_{\text{diss1}} &= i_{R1}^2 R_1 \\
&= \left(\frac{v_+}{R_1}\right)^2 R_1 \\
&= \frac{1}{G^2}\frac{1}{R} v_o^2 \\
P_{\text{diss2}} &= i_{R2}^2 R_2 \\
&= \left(\frac{v_+}{R_1} + C_1 \frac{dv_+}{dt}\right)^2 R_2 \\
&= \frac{1}{G^2}\frac{1}{R}\left(v_o + \frac{1}{\omega_0}\frac{dv_o}{dt}\right)^2 \\
P_{\text{diss}} &= P_{\text{diss1}} + P_{\text{diss2}} \\
&= \frac{1}{G^2}\frac{1}{R}\left[v_o^2 + \left(v_o + \frac{1}{\omega_0}\frac{dv_o}{dt}\right)^2\right] \\
&= \frac{1}{G^2}\frac{1}{R}\left[2v_o\left(v_0 + \frac{1}{\omega_0}\frac{dv_o}{dt}\right) + \frac{1}{\omega_0^2}\left(\frac{dv_o}{dt}\right)^2\right].
\end{aligned}
$$

5.4 Amplifier power

The total power provided by the amplifier (above the power dissapated in the gain feedback network) must be the sum of these two.

$$
\begin{aligned}
P_{\text{amp}} &= \frac{dU}{dt} + P_{\text{diss}} \\
&= \frac{1}{G^2}\frac{1}{R}\left[(G-2)v_0\left(v_0 + \frac{1}{\omega_0}\frac{dv_0}{dt}\right) - \frac{1}{\omega_0^2}\left(\frac{dv_o}{dt}\right)^2\right] + \frac{1}{G^2}\frac{1}{R}\left[2v_o\left(v_0 + \frac{1}{\omega_0}\frac{d}{d}\right.\right. \\
&= \frac{1}{G}\frac{1}{R}\left(v_o^2 + \frac{1}{\omega_0} v_o \frac{dv_o}{dt}\right) \\
&= \frac{1}{G}\left(\frac{v_o}{R} + C\frac{dv_o}{dt}\right) v_o \\
&= \left(\frac{v_+}{R} + C\frac{dv_+}{dt}\right) v_o \\
&= i_f v_o,
\end{aligned}
$$

where $i_f = i_2 = i_{R_1} + i_{C_1}$ is the feedback current coming from the amplifier output.

We interpret this simple result as stating that the power provided by the amplifier (above the power dissipated in the gain feedback network and with no external load) is indeed the product of the current i_f fed back by the amplfier into the feedback network and the driving voltage v_o at the amplifier output. This result, derived as a sum of the rate of change of stored energy and power dissipated, is also a good check on the preceding calculations.

6 Energy and power assuming sinusoidal oscillation

Let's suppose that we have $G = 3$ and a resulting steady oscillation that is very nearly sinusoidal.

$$v_o \approx V_o \cos \omega_o t.$$

We'll see below with nonlinear equations that this situation can be achieved.

6.1 Evaluating the energy terms

The stored energies are:

$$\begin{aligned}
U_1 &= \frac{1}{G^2}\frac{1}{2}C_1 v_o^2 \\
&= \frac{1}{9}\frac{1}{2}CV_o^2 \cos^2 \omega_o t \\
&= \frac{1}{9}\frac{1}{2}CV_o^2 \left[\frac{1}{2} + \frac{1}{2}\cos 2\omega_0 t\right] \\
&= \frac{1}{2}CV_o^2 \frac{1}{18}[1 + \cos 2\omega_0 t] \\
&= \frac{1}{2}CV_o^2 \frac{1}{18}\left[1 + \cos 4\pi \frac{t}{T_0}\right] \\
U_2 &= \frac{1}{G^2}\frac{1}{2}C\left[(G-2)v_o - \frac{1}{\omega_0}\frac{dv_o}{dt}\right]^2 \\
&= \frac{1}{9}\frac{1}{2}C\left[V_o \cos \omega_o t - \frac{1}{\omega_0}\frac{d}{dt}(V_o \cos \omega_o t)\right]^2 \\
&= \frac{1}{9}\frac{1}{2}CV_o^2[\cos \omega_o t + \sin \omega_o t]^2 \\
&= \frac{1}{9}\frac{1}{2}CV_o^2[1 + 2\cos \omega_o t \sin \omega_o t] \\
&= \frac{1}{9}\frac{1}{2}CV_o^2[1 + \sin 2\omega_o t] \\
&= \frac{1}{2}CV_o^2 \frac{1}{9}\left[1 + \cos 4\pi\left(\frac{t}{T_0} - \frac{1}{8}\right)\right].
\end{aligned}$$

$$\begin{aligned}
U &= U_1 + U_2 \\
&= \frac{1}{2}CV_o^2\left\{\frac{1}{18}[1 + \cos(2\omega_0 t)] + \frac{1}{9}[1 + \sin(2\omega_o t)]\right\} \\
&= \frac{1}{2}CV_o^2 \frac{1}{18}[3 + \cos(2\omega_0 t) + 2\sin(2\omega_o t)] \\
&= \frac{1}{2}CV_o^2 \frac{1}{18}\left[3 + \sqrt{5}\cos 4\pi\left(\frac{t}{T_0} - 0.0881\right)\right]
\end{aligned}$$

The rates of change of stored energy are:

$$\begin{aligned}
\frac{dU_1}{dt} &= \frac{1}{G^2}\frac{1}{R}\left[\frac{1}{\omega_0}v_o\frac{dv_0}{dt}\right] \\
&= \frac{1}{9}\frac{V_0^2}{R}(-\cos\omega_0 t\sin\omega_0 t) \\
&= -\frac{V_0^2}{2R}\frac{1}{9}\sin 2\omega_0 t \\
&= \frac{V_0^2}{2R}\frac{1}{9}\cos 4\pi\left(\frac{t}{T_0}+\frac{1}{8}\right) \\
\frac{dU_2}{dt} &= \frac{1}{G^2}\frac{1}{R}\left[(G-2)v_0\left(v_0+\frac{1}{\omega_0}\frac{dv_0}{dt}\right)-\frac{1}{\omega_0^2}\left(\frac{dv_o}{dt}\right)^2-\frac{1}{\omega_0}v_0\frac{dv_0}{dt}\right] \\
&= \frac{1}{9}\frac{V_0^2}{R}(\cos^2\omega_0 t-\sin^2\omega_0 t) \\
&= \frac{V_0^2}{2R}\frac{2}{9}\cos 2\omega_0 t \\
&= \frac{V_0^2}{2R}\frac{2}{9}\cos 4\pi\left(\frac{t}{T_0}\right). \\
\frac{dU}{dt} &= \frac{dU_1}{dt}+\frac{dU_2}{dt} \\
&= \frac{V_0^2}{2R}\frac{1}{9}[-\sin 2\omega_0 t+2\cos 2\omega_o t] \\
&= \frac{V_0^2}{2R}\frac{\sqrt{5}}{9}\cos 4\pi\left(\frac{t}{T_0}-0.0369\right).
\end{aligned}$$

The dissipated powers are:

$$
\begin{aligned}
P_{\text{diss1}} &= \frac{1}{G^2}\frac{1}{R}v_o^2 \\
&= \frac{1}{9}\frac{1}{R}V_0^2\cos^2\omega_0 t \\
&= \frac{V_o^2}{2R}\frac{1}{9}(1+\cos 2\omega_0 t).
\end{aligned}
$$

$$
\begin{aligned}
P_{\text{diss2}} &= \frac{1}{G^2}\frac{1}{R}\left(v_o + \frac{1}{\omega_0}\frac{dv_o}{dt}\right)^2 \\
&= \frac{1}{9}\frac{1}{R}V_0^2(\cos\omega_0 t - \sin\omega_0 t)^2 \\
&= \frac{1}{9}\frac{1}{R}V_0^2(1-2\cos\omega_0 t\sin\omega_0 t) \\
&= \frac{V_0^2}{2R}\frac{1}{9}(2-2\sin 2\omega_0 t).
\end{aligned}
$$

$$
\begin{aligned}
P_{\text{diss}} &= \frac{1}{9}\frac{1}{2R}(3+\cos 2\omega_0 t - 2\sin 2\omega_0 t) \\
&= \frac{1}{2R}\frac{1}{9}\left[3+\sqrt{5}\cos 4\pi\left(\frac{t}{T_0}+0.0881\right)\right]
\end{aligned}
$$

The total power provided by the amplifier is

$$
\begin{aligned}
P_{\text{amp}} &= \frac{dU}{dt} + P_{\text{diss}} \\
&= \frac{1}{9}\frac{V_o^2}{2R}[(2\cos 2\omega_o t - \sin 2\omega_0 t) + (3+\cos 2\omega_0 t - 2\sin 2\omega_0 t)] \\
&= \frac{V_o^2}{2R}\frac{1}{3}(1+\cos 2\omega_0 t - \sin 2\omega_0 t).
\end{aligned}
$$

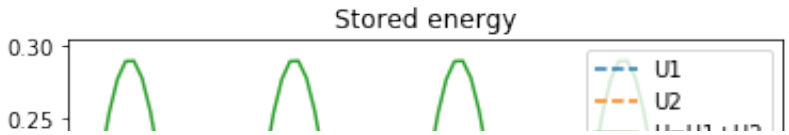
$$
\begin{aligned}
\text{Alternatively, } P_{\text{amp}} &= \frac{1}{G}\frac{1}{R}\left(v_o^2 + \frac{1}{\omega_0}v_o\frac{dv_o}{dt}\right) \\
&= \frac{1}{3}\frac{1}{R}\left[V_o^2\cos^2\omega_0 t - V_o^2\cos\omega_0 t\sin\omega_0 t\right] \\
&= \frac{V_o^2}{2R}\frac{1}{3}(1+\cos 2\omega_0 t - \sin 2\omega_0 t) \\
&= \frac{V_o^2}{2R}\frac{1}{3}\left[1+\sqrt{2}\cos 4\pi\left(\frac{t}{T}+\frac{1}{16}\right)\right].
\end{aligned}
$$

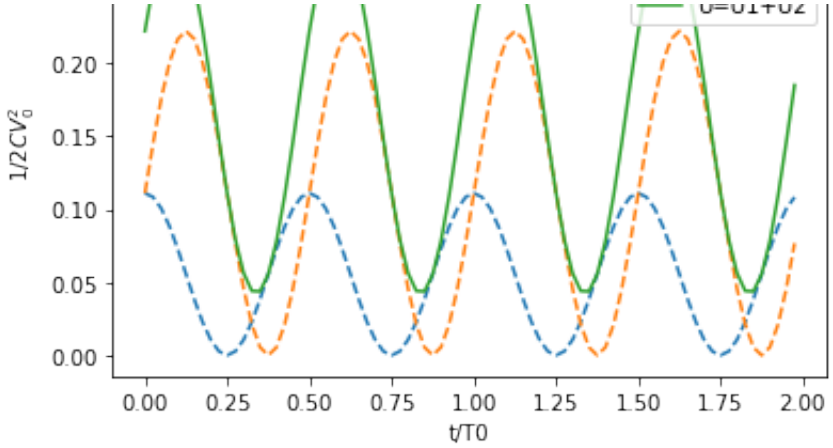
6.2 Code to compute and plot the energy and power expressions

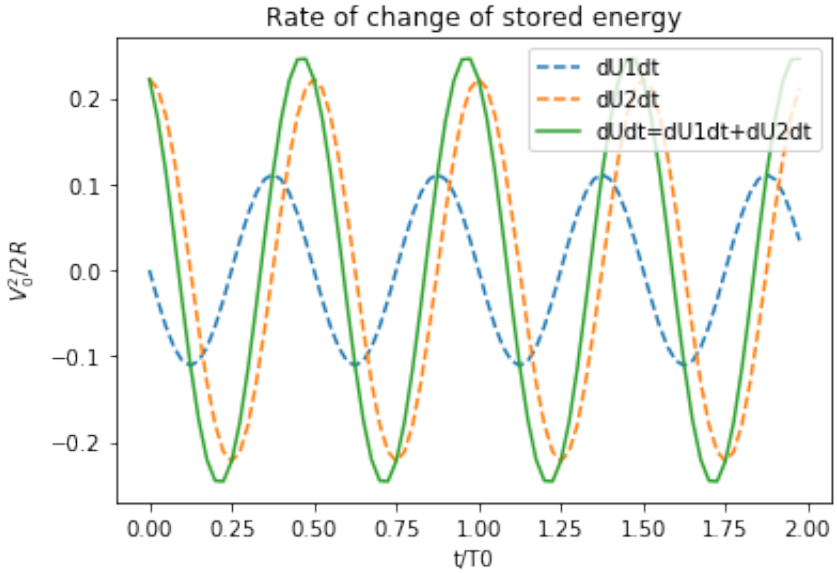
In [6]:
```
import numpy as np
import math
import matplotlib.pyplot as plt

def U1(omegat):
    U1 = (1/18.)*(1+math.cos(2*omegat))
```

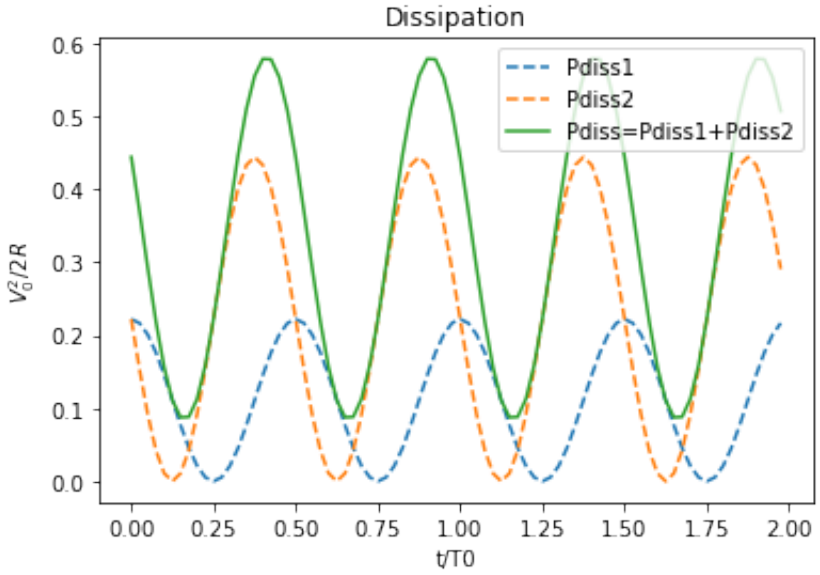
```
    return U1 #in units of 1/2CV^2

def U2(omegat):
    U2 = (1/9.)*(1+math.sin(2*omegat))
    return U2 #in units of 1/2CV^2

def U(omegat):
    U = (1/18.)*(3+math.cos(2*omegat)+2*math.sin(2*omegat))
    return U #in units of 1/2CV^2

def dU1dt(omegat):
    dU1dt = (-1/9.)*math.sin(2*omegat)
    return dU1dt #in units of V^2/(2R)

def dU2dt(omegat):
    dU2dt = (2/9.)*math.cos(2*omegat)
    return dU2dt #in units of V^2/(2R)

def dUdt(omegat):
    dUdt = (1/9.)*(2*math.cos(2*omegat)-math.sin(2*omegat))
    return dUdt #in units of V^2/(2R)

def Pdiss1(omegat):
    Pdiss1 = (1/9.)*(1+math.cos(2*omegat))
    return Pdiss1 #in units of V^2/(2R)

def Pdiss2(omegat):
    Pdiss2 = (1/9.)*(2-2*math.sin(2*omegat))
    return Pdiss2 #in units of V^2/(2R)

def Pdiss(omegat):
    Pdiss = (1/9.)*(3+math.cos(2*omegat)-2*math.sin(2*omegat))
    return Pdiss #in units of V^2/(2R)

def Pamp(omegat):
    Pamp = (1/3.)*(1+math.cos(2*omegat)-math.sin(2*omegat))
    return Pamp #in units of V^2/(2R)


a_t = []
a_U1 = []
a_U2 = []
a_U =[]
a_dU1dt = []
a_dU2dt = []
a_dUdt = []
a_Pdiss1 = []
a_Pdiss2 = []
a_Pdiss = []
a_PampSum = []
a_Pamp = []

for i in range(0,80):   #t is in units of T_0, the period
    t=i/40
```

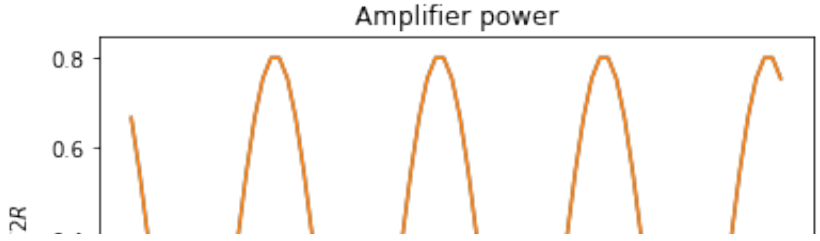
```
    t=1/40.
    omegat=2*math.pi*t
    a_t = np.append(a_t,t)
    a_U1 = np.append(a_U1,U1(omegat))
    a_U2 = np.append(a_U2,U2(omegat))
    a_U = np.append(a_U,U(omegat))
    a_dU1dt = np.append(a_dU1dt,dU1dt(omegat))
    a_dU2dt = np.append(a_dU2dt,dU2dt(omegat))
    a_dUdt = np.append(a_dUdt,dUdt(omegat))
    a_Pdiss1 = np.append(a_Pdiss1,Pdiss1(omegat))
    a_Pdiss2 = np.append(a_Pdiss2,Pdiss2(omegat))
    a_Pdiss = np.append(a_Pdiss,Pdiss(omegat))
    a_PampSum = np.append(a_PampSum,1.0*(dUdt(omegat)+Pdiss(omegat)))
    a_Pamp = np.append(a_Pamp,Pamp(omegat))


plt.figure(0)
plt.plot(a_t,a_U1,'--',label='U1')
plt.plot(a_t,a_U2,'--',label='U2')
plt.plot(a_t,a_U,label='U=U1+U2')
plt.legend(loc='upper right')
plt.title('Stored energy')
plt.xlabel('t/T0')
plt.ylabel(' $1/2 CV_0^2$')

plt.figure(1)
plt.plot(a_t,a_dU1dt,'--',label='dU1dt')
plt.plot(a_t,a_dU2dt,'--',label='dU2dt')
plt.plot(a_t,a_dUdt,label='dUdt=dU1dt+dU2dt')
plt.legend(loc='upper right')
plt.title('Rate of change of stored energy')
plt.xlabel('t/T0')
plt.ylabel(' $V_0^2/2R$')

plt.figure(2)
plt.plot(a_t,a_Pdiss1,'--',label='Pdiss1')
plt.plot(a_t,a_Pdiss2,'--',label='Pdiss2')
plt.plot(a_t,a_Pdiss,label='Pdiss=Pdiss1+Pdiss2')
plt.legend(loc='upper right')
plt.title('Dissipation')
plt.xlabel('t/T0')
plt.ylabel('$V_0^2/2R$')

plt.figure(3)
plt.plot(a_t,a_PampSum,a_t,a_Pamp)
plt.title('Amplifier power')
plt.xlabel('t/T0')
plt.ylabel('Pamp  $V_0^2/2R$')

plt.show()
```

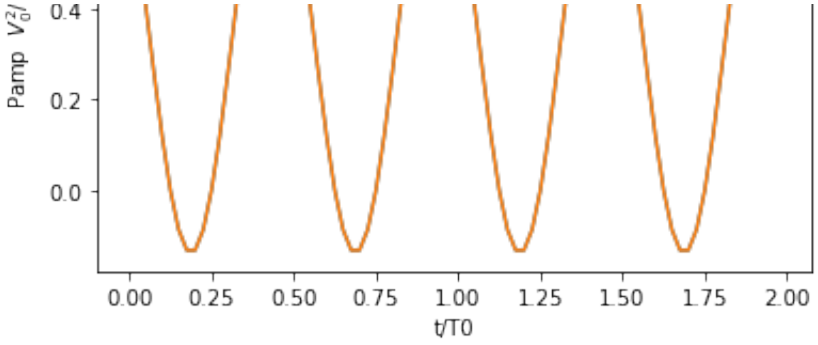



0.20
0.15
0.10
0.05
0.00
1/2 CV$_0^2$
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
t/T0


Rate of change of stored energy
dU1dt
dU2dt
dUdt=dU1dt+dU2dt
0.2
0.1
0.0
−0.1
−0.2
V$_0^2$/2R
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
t/T0


Dissipation
Pdiss1
Pdiss2
Pdiss=Pdiss1+Pdiss2
0.6
0.5
0.4
0.3
0.2
0.1
0.0
V$_0^2$/2R
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
t/T0


Amplifier power
0.8
0.6
2R



7 Creating a limit cycle by introducing nonlinearity: the Hopf oscillator

The linearized dynamics provides fixed cycles only for a special value of the gain parameter. Also, the amplitude depends on the initial condition. In order to obtain more robust oscillations, we intoduce nonlinearity.

To obtain insight into the transition to self-sustained oscillations and the formation of limit cycles, we first look at an idealized system of equations. The features revealed in the following special model – particularly with respect to the way the limit cycle amplitude scales with the bifurcation parameter ϵ – apply near onset to a more general class of oscillators. These oscillators make a continuous transition from steady state to limit-cycle oscillation when $\epsilon = 0$ and the transition is called a *Hopf bifurcation*. Both the linear and the nonlinear theory of the Hopf oscillator are particularly elegant and provide an ideal starting point for understanding limit cycles. In some sense, the Hopf system creates a target for designing practical oscillators to achieve.

7.1 Setting up the Hopf dynamical system

Consider this system of equations:

$$\frac{1}{\omega_0}\frac{dv}{dt} = \epsilon v + w - \zeta v(v^2 + w^2).$$
$$\frac{1}{\omega_0}\frac{dw}{dt} = -v + \epsilon w - \zeta w(v^2 + w^2).$$

The linearized version for small v and w is

$$\frac{1}{\omega_0}\frac{d}{dt}\begin{pmatrix} v \\ w \end{pmatrix} \approx \begin{pmatrix} \epsilon & 1 \\ -1 & \epsilon \end{pmatrix}\begin{pmatrix} v \\ w \end{pmatrix}$$

Trying

$$\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} \widehat{v} \\ \widehat{w} \end{pmatrix} e^{st}$$

leads to a characteristic equation for eigenvalues

$$\begin{vmatrix} \epsilon - \frac{s}{\omega_0} & 1 \\ -1 & \epsilon - \frac{s}{\omega_0} \end{vmatrix} = 0.$$

$$\left(\epsilon - \frac{s}{\omega_0}\right)^2 + 1 = 0.$$

$$s = (\epsilon \pm i)\omega_0 .$$

Thus a system with exponential growth for $\epsilon > 0$ of oscillations with frequence ω_0 is obtained in a particularly straightforward way.

Now return to the full nonlinear equation in order to lean how the amplitude of the oscillation stabilizes, here in an elegant and straightforward way.

First, make a change of variables.

$$v = A\cos\theta.$$
$$w = A\sin\theta.$$

In this notation, A represents an amplitude and $theta$ a phase: both could in principle be time dependent. The inverse relations are:

$$A = v^2 + w^2.$$
$$\tan\theta = \frac{w}{v}.$$

In these polar coordinates the full nonlinear dynamical system becomes remarkably simple:

$$\frac{dA}{dt} = \epsilon A - \zeta A^3.$$
$$\frac{d\theta}{dt} = -\omega_0 .$$

From these results, we see that for the Hopf model a steady-state amplitude may be reached for the limit cycle when $dA/dt = 0$, which occurs for

$$A = \sqrt{\frac{\epsilon}{\zeta}},$$

while for any ϵ the frequency remains constant:

$$\omega \equiv \left|\frac{d\theta}{dt}\right| = \omega_0 .$$

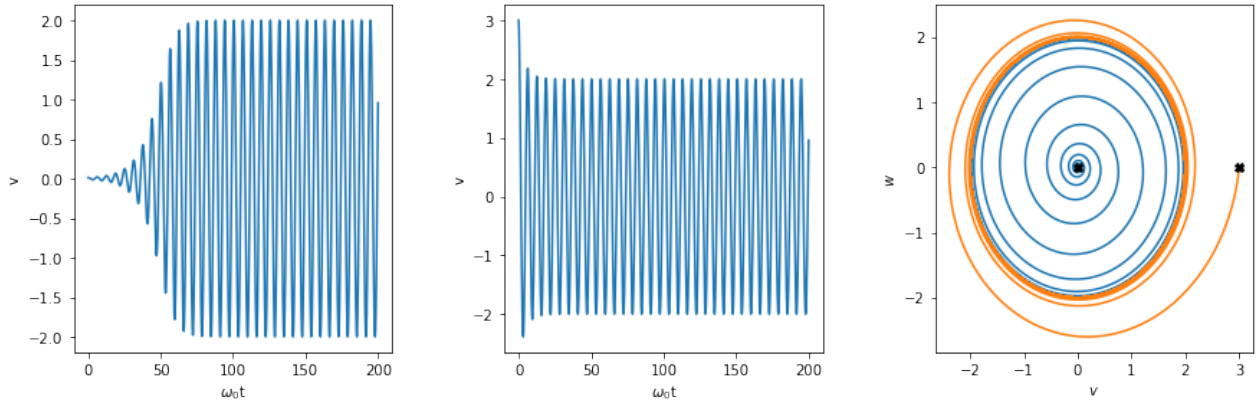
7.2 Numerical simulation of the Hopf dynamical system

The following code shows how initial conditions inside and outside the predicted radius of steady amplitude oscillation converge onto the same orbit...hence the idea of a "limit cycle". The radius of the limit cycle is indeed equal to $\sqrt{\epsilon/\zeta}$.

In [1]:
```
% matplotlib inline
# Code to numerically integrate the Hopf dynamical system
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt,epsilon,zeta):
    r2 = xx[0]*xx[0]+xx[1]*xx[1]
    ff1 = epsilon*xx[0]+xx[1]-zeta*xx[0]*r2
    ff2 = -xx[0]+epsilon*xx[1]-zeta*xx[1]*r2
#    print(ff1,ff2)
    return [ff1,ff2]

epsilon = 0.1 # specify epsilon directly
zeta = 0.025
print('epsilon = ',epsilon,'zeta = ',zeta,'A = ',math.sqrt(epsilon/zeta)

timestep = 0.01
timerange = 200
Nstep = int(timerange/timestep)
a_tt = np.arange(0,timerange,timestep) # time array

# Initial conditions inside the limit cycle
v0_in = 0.01
w0_in = 0.0

a_sol = integrate.odeint(dynsys,[v0_in,w0_in],a_tt,args=(epsilon,zeta))
vva = a_sol[0:Nstep,0]
wwa = a_sol[0:Nstep,1]

# Initial conditions outside the limit cycle
v0_out = 1.5*math.sqrt(epsilon/zeta)
w0_out = 0.0

a_sol = integrate.odeint(dynsys,[v0_out,w0_out],a_tt,args=(epsilon,zeta)
vvb = a_sol[0:Nstep,0]
wwb = a_sol[0:Nstep,1]

plt.figure(1)
plt.subplot(1,3,1)
plt.plot(a_tt,vva)
plt.xlabel('$\omega_0$t')
plt.ylabel('v')
```

```
plt.subplot(1,3,2)
plt.plot(a_tt,vvb)
plt.xlabel('$\omega_0$t')
plt.ylabel('v')

plt.subplot(1,3,3)
plt.plot(vva,wwa,vvb,wwb,[v0_in],[w0_in],'kX',[v0_out],[w0_out],'kX')
plt.xlabel('$v$')
plt.ylabel('$w$')
#plt.annotate('eps = {0:4.2f}'.format(epsilon),xy=(5,0.75))

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspace=(

plt.show()
```

```
epsilon =  0.1 zeta =  0.025 A =  2.0
```



8 The van der Pol equation: steps towards a practical model for nonlinear oscillations

Now we'll move towards some more practical versions of nonlinear equations. Here the word *practical* refers to types of dynamics that might better describe actual constructed systems. The first step is to take the harmonic oscillator equation that we used above and introduce nonlinearity into the damping term. The goal is to use the growing amplitude of oscillation to re-stabilize the oscillations that otherwise "explode" when the gain exceeds its critical value.

8.1 Constructing the generalized van der Pol equation

Recall our dynamical equation using dimensonless time

$$\tilde{t} \equiv \omega_0 t$$

where $\omega_0 \equiv \frac{1}{RC}$. We will continue to drop the subscript and use v as a simple notation for the output voltage v_o and define $w \equiv dv/d\tilde{t} = (1/\omega_0)dv/dt$.

The second-order equation is then written as:

$$\frac{d^2v}{d\tilde{t}^2} - \epsilon\frac{dv}{d\tilde{t}} + v = 0.$$

The equivalent first-order system of two equations was:

$$\frac{dv}{d\tilde{t}} = w$$
$$\frac{dv}{d\tilde{t}} = -v + \epsilon w.$$

This equation has negative dampling when $\epsilon > 0$ and we know that this produces dynamics that spirals outwards to infinity in the (u, w) phase-plane. What if we modified the damping term by a function that, for $\epsilon > 0$, brought the damping back to "zero" in response to the growing amplifier power that is proportional to v^2 ? We will explore below how to do this with actual circuit components, but first let's explore a mathematical model that subtracts $-\zeta v^2$ from ϵ :

$$\frac{d^2v}{d\tilde{t}^2} - (\epsilon - \zeta v^2)\frac{dv}{d\tilde{t}} + v = 0.$$

We'll define a voltage scale $v_* \equiv \frac{1}{\sqrt{\zeta}}$ so that we can write

$$\frac{d^2v}{d\tilde{t}^2} - (\epsilon - \frac{v^2}{v_*^2})\frac{dv}{d\tilde{t}} + v = 0.$$

We can now also rescale voltage so that $\widetilde{v} = \sqrt{\zeta}v = \frac{v}{v_*}$.

$$\frac{d^2\widetilde{v}}{d\tilde{t}^2} - (\epsilon - \widetilde{v}^2)\frac{d\widetilde{v}}{d\tilde{t}} + \widetilde{v} = 0.$$

As a system of two equations:

$$\frac{d\widetilde{v}}{d\tilde{t}} = \widetilde{w}$$
$$\frac{d\widetilde{v}}{d\tilde{t}} = -\widetilde{v} + (\epsilon - \widetilde{v}^2)\widetilde{w}.$$

We call this the *generalized van der Pol equation*. It is named after Balthasar van der Pol (https://en.wikipedia.org/wiki/Balthasar_van_der_Pol), the Dutch physicist who worked at Phillips Research Labs in the first half of the 20th Century. His work included theory of electronic oscllators...which at the time were made using vacuum tubes.

The scaled version of this equation allows a lot of different experimental conditions to be captured into a single computation of $\widetilde{v}(\tilde{t})$. The actual voltage is then obtained from:

$$v_o(t) = v_*\widetilde{v}(\omega_o t).$$

8.2 Alternate rescaling to obtain the van der Pol equation

In order to connect with some of the literature, it is useful to see a different scaling that depends on ϵ. This leads to the more common version of the van der Pol equation. Unfortunately, the bifurcation behvaior is masked because the scaling becomes singular at $\epsilon = 0$.

Start with the fully dimensioned equation:

$$\frac{d^2v}{dt^2} - (\epsilon - \zeta v^2)\omega_0 \frac{dv}{dt} + \omega_0^2 v = 0.$$

Provided that $\epsilon \neq 0$ we can write:

$$\frac{d^2v}{dt^2} - \epsilon(1 - \frac{\zeta}{\epsilon}v^2)\omega_0 \frac{dv}{dt} + \omega_0^2 v = 0.$$

Rescale time $\tilde{t} \equiv \omega_0 t$ and $\tilde{v}' \equiv \sqrt{\frac{\zeta}{\epsilon}} v$. Then the equation becomes:

$$\frac{d^2\tilde{v}'}{d\tilde{t}^2} - \epsilon(1 - \tilde{v}'^2)\frac{d\tilde{v}'}{d\tilde{t}} + \tilde{v}' = 0.$$

This is the standard form of the van der Pol equation. Notice, however, that the scale factor $\sqrt{\frac{\zeta}{\epsilon}}$ becomes singular at $\epsilon = 0$ at the same point that the actual output voltage amplitude is just about to rise from a value of zero.

8.3 Numerically integrating the van der Pol equation

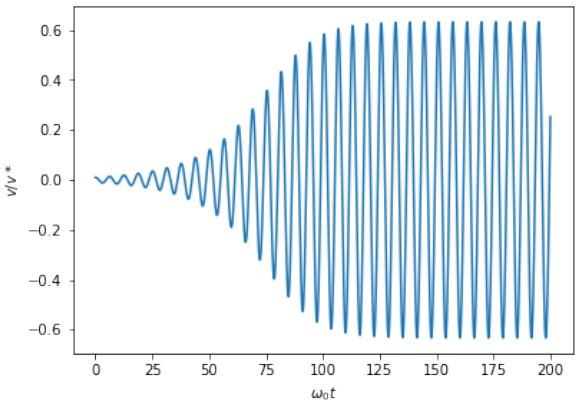
The following produces a time series and phase portrait two values of ϵ. The dashed circle is the trajectoruy predicted from the averaging method (see below). Notice how harmonic distortion increases, as seen in the shape of the waveform, the non-circular form of the phase portrait, and in the power spectrum. The lowest-order averaging result from the following section does not show this, but still obtains very good estimates of the peak amplitude of the voltage v.

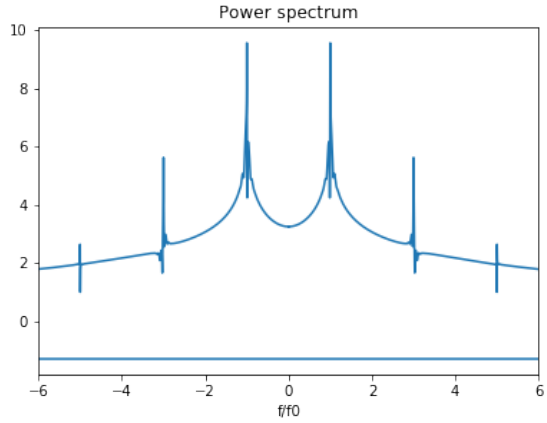
In [36]:
```
% matplotlib inline
# Code to numerically integrate the actual dynamical system
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt,epsilon):
    ff1 = xx[1]
    ff2 = -xx[0]+(epsilon-xx[0]**2)*xx[1]
#    print(ff1,ff2)
    return [ff1,ff2]
```

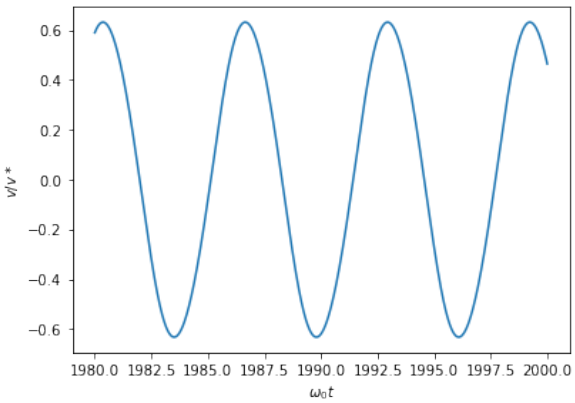
```
a_epsilon = [0.1,0.5] # specify an initial epsilon directly (alt value 0

# Initial conditions
v_0 = 0.01
w_0 = 0.0

timestep = 0.01
timerange = 2000
Nstep = int(timerange/timestep)
a_tt = np.arange(0,timerange,timestep) # time array

for i in range(0,2):

    epsilon = a_epsilon[i]
    print('epsilon = ',epsilon)

    # Regarding numerical integration:
    # See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/sc
    # Also drr https://nathantypanski.com/blog/2014-08-23-ode-solver-py.
    a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))

    vv = a_sol[0:Nstep,0]
    ww = a_sol[0:Nstep,1]

    A_avg = 2*math.sqrt(epsilon)
    dtheta = 2*math.pi/100
    vvcir = []
    wwcir = []
    for j in range(0,101):
        theta = j*dtheta
        vvcir = np.append(vvcir,A_avg*np.cos(theta))
        wwcir = np.append(wwcir,A_avg*np.sin(theta))

    #Now compute the spectrum
    #sp = np.fft.fft(vv[Nstep-4096:Nstep])
    #freq = np.fft.fftfreq(4096, d=timestep)
    sp = np.fft.fft(vv)
    freq = np.fft.fftfreq(Nstep, d=timestep)

    plt.figure(0)

    Ntransient = int(Nstep/10)
    plt.subplot(1,2,1)
    plt.plot(a_tt[0:Ntransient],vv[0:Ntransient])
    plt.xlabel('$\omega_0 t$')
    plt.ylabel('$v/v*$')

    plt.subplot(1,2,2)
    plt.plot(2*math.pi*freq, np.log10(sp.real**2+sp.imag**2))
    plt.xlim(-6,6)
    plt.xlabel('f/f0')
    #plt.ylabel('V^2/Hz')
    plt.title('Power spectrum')
    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspa
```

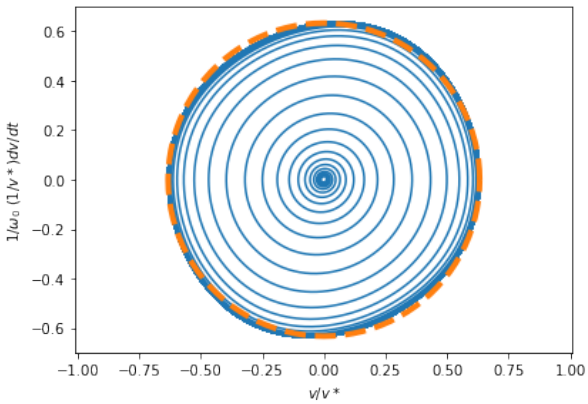
```
                    wspace=0.35)

    plt.figure(1)

    plt.subplot(1,2,1)
    plt.plot(a_tt[Nstep-2000:Nstep],vv[Nstep-2000:Nstep])
    plt.xlabel('$\omega_0 t$')
    plt.ylabel('$v/v*$')
    #plt.annotate('eps = {0:4.2f}'.format(epsilon),xy=(5,0.75))

    plt.subplot(1,2,2)
    plt.plot(vv,ww)
    plt.plot(vvcir,wwcir,'--',linewidth=4)
    plt.xlabel('$v/v*$')
    plt.ylabel('$1/\omega_0$ $(1/v*)dv/dt$')
    plt.axis('equal')
    #plt.annotate('eps = {0:4.2f}'.format(epsilon),xy=(5,0.75))

    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspa
                    wspace=0.35)
    plt.show()
```

epsilon = 0.1

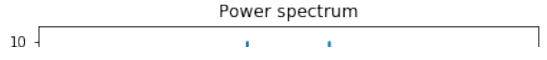


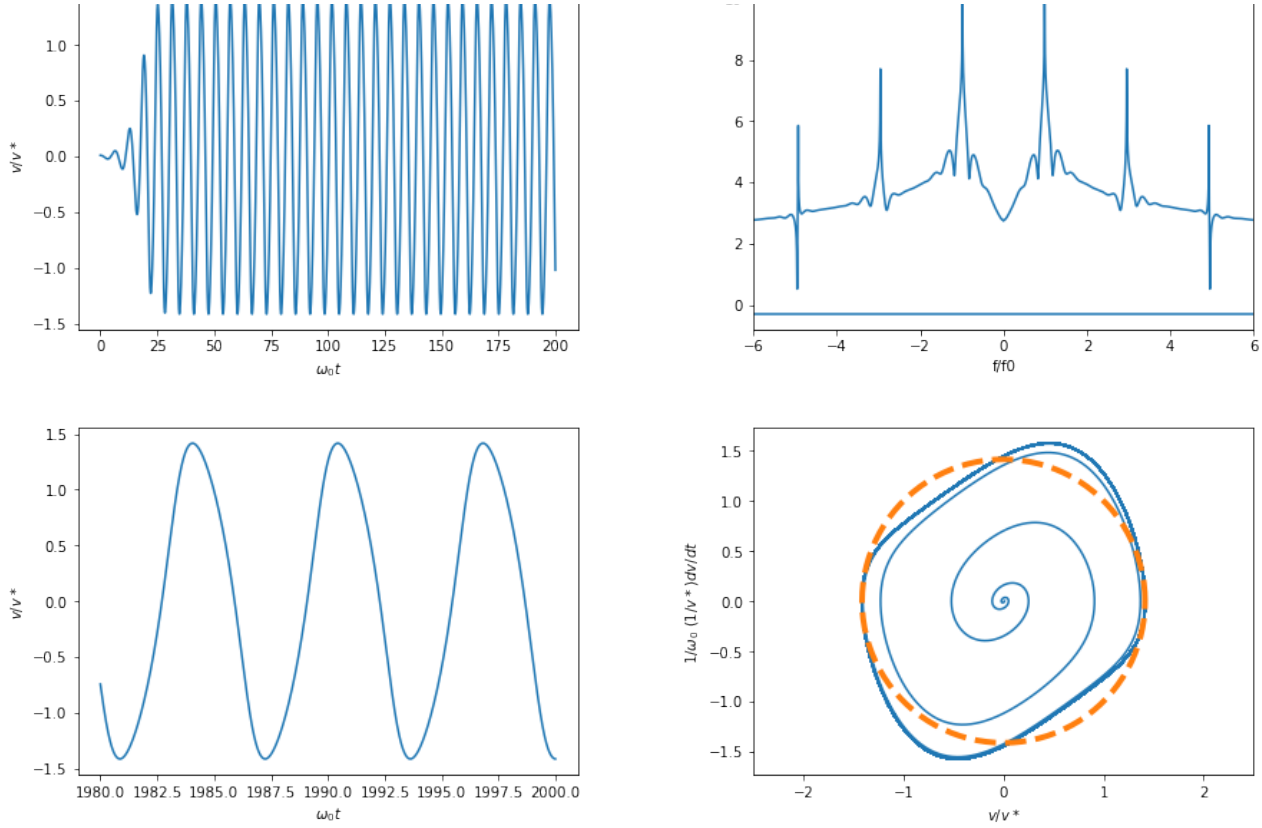






epsilon = 0.5





8.4 Method of Averaging Applied to the Van der Pol equation

Consider the single scaled nonlinear equation in v.

$$\frac{d^2\widetilde{v}}{dt^2} - (\epsilon - \widetilde{v}^2)\frac{d\widetilde{v}}{dt} + \widetilde{v} = 0.$$

Recall that $\widetilde{t} \equiv \omega_0 t$ and $\widetilde{v} \equiv \frac{v}{v_*}$.

NOTE: For notational clarity, we will drop the tilde's in the following but assume the equations are still in scaled form.

The dynamics can be re-written as two first-order equations

$$\frac{dv}{dt} = w$$
$$\frac{dv}{dt} = -v + (\epsilon - v^2)w.$$

We are going to assume that the the solution will evolve from the simple sinsoidal solution when the nonlinear term is turned on $\delta > 0$. Thus we think of time variations in amplitude and phase around the limit cycle such that the phase-space trajectory can be written in the form:

$$v(t) = A(t)\cos[t + \phi(t)].$$

$$w(t) = -A(t)\sin[t + \phi(t)].$$

Our goal is to find equations for the amplitude function $A(t)$ and the phase function $\phi(t)$. Inserting the expressions for v and w into the dynamical equations gives

$$\frac{dA(t)}{dt}\cos[t+\phi(t)] - A(t)\left[1+\frac{d\phi(t)}{dt}\right]\sin[t+\phi(t)] = -A(t)\sin[t+\phi(t)]$$
$$-\frac{dA(t)}{dt}\sin[t+\phi(t)] - A(t)\left[1+\frac{d\phi(t)}{dt}\right]\cos[t+\phi(t)] = -A(t)\cos[t+\phi(t)] - \left\{\epsilon - A^2(t\right.$$

This simplifies to

$$\frac{dA(t)}{dt}\cos[t+\phi(t)] - A(t)\frac{d\phi(t)}{dt}\sin[t+\phi(t)] = 0.$$
$$-\frac{dA(t)}{dt}\sin[t+\phi(t)] - A(t)\frac{d\phi(t)}{dt}\cos[t+\phi(t)] = -\left\{\epsilon - A^2(t)\cos^2[t+\phi(t)]\right\}A(t)\sin[t-$$

Use the first of the above pair of equations to write

$$\frac{d\phi(t)}{dt} = \frac{\frac{dA(t)}{dt}}{A(t)}\frac{\cos[t+\phi(t)]}{\sin[t+\phi(t)]}.$$

Substitute this into the second of the above pair of equations and multiply through by $\sin[t+\phi(t)]$:

$$-\frac{dA(t)}{dt}\sin^2[t+\phi(t)] - \frac{dA(t)}{dt}\cos^2[t+\phi(t)] = -\left\{\epsilon - A^2(t)\cos^2[t+\phi(t)]\right\}A(t)\sin^2[t+$$

Collect terms and use the fact that $\cos^2[t+\phi(t)] + \sin^2[t+\phi(t)] = 1$ to obtain

$$\frac{dA(t)}{dt} = \left\{\epsilon - A^2(t)\cos^2[t+\phi(t)]\right\}A(t)\sin^2[t+\phi(t)].$$

We can see also that

$$\frac{d\phi(t)}{dt} = \left\{\epsilon - A^2(t)\cos^2[t+\phi(t)]\right\}\cos[t+\phi(t)]\sin[t+\phi(t)].$$

So far this has been an exact change of coordinates. We now make an approximation. Anticipating that $A^2(t) \sim O(\epsilon)$, we take both time derivatives $\frac{dA(t)}{dt}$ and $\frac{d\phi(t)}{dt}$ to likewise be $O(\epsilon)$ and thus the amplitude and phase are slowly varying deviations from the circular limit cycle. We will then treat them as effectively constant over one cycle and replace the right-hand side with averages. Taking $\theta \equiv t + \phi(t)$,

$$\begin{aligned}\frac{dA(t)}{dt} &\approx \frac{1}{2\pi}\int_0^{2\pi}\left[\epsilon - A^2(t)\cos^2\theta\right]A(t)\sin^2\theta d\theta \\ &\approx \epsilon A(t)\frac{1}{2\pi}\int_0^{2\pi}\sin^2\theta d\theta + A^3(t)\frac{1}{2\pi}\int_0^{2\pi}\cos^2\theta\sin^2\theta d\theta \\ &= \frac{1}{2}\epsilon A(t) - \frac{1}{8}A^3(t).\end{aligned}$$

So we have

$$\frac{dA(t)}{dt} \approx \frac{1}{8}\left[4\epsilon - A^2(t)\right] A(t).$$

This is an example of an *amplitude equation*. It may be formally integrated starting with

$$8\frac{dA}{\left[4\epsilon - A^2\right] A} = dt$$

to arrive at

$$\epsilon\frac{A + 2\sqrt{\epsilon}}{A^2(2\sqrt{\epsilon} - A)} = e^{\epsilon t}.$$

At long times

$$A \to 2\sqrt{\epsilon}$$

(justifying thie assumption that $A^2 \sim O(\epsilon)$), while the next order correction is

$$A \approx \sqrt{\epsilon}\left(2 - e^{-\epsilon t}\right).$$

Returning to dimensioned coordinates,

$$v \approx \left(2 - e^{-\epsilon\omega_0 t}\right) v_* \sqrt{\epsilon}\cos(\omega_0 t),$$

and a predicted final steady oscillation is given by

$$v \to 2v_* \sqrt{\epsilon}\cos(\omega_0 t).$$

8.5 Comparing the averaging result for amplitude to the maxima of the full numerical integration

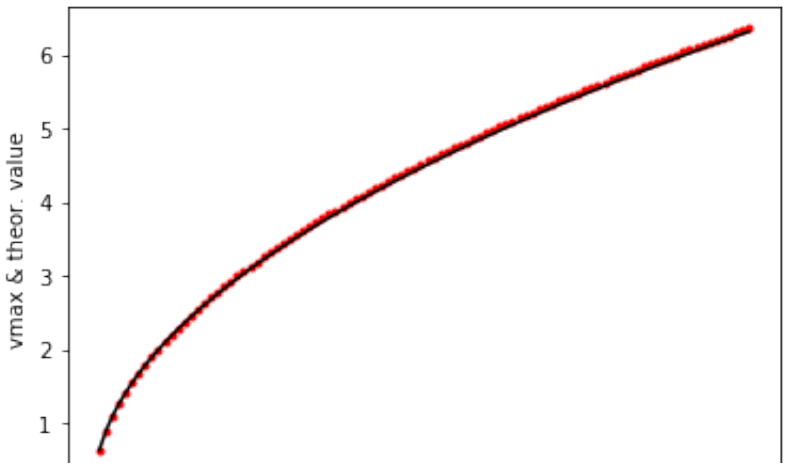
In [24]:
```
% matplotlib inline
# Code to compare the averaging result tothe numerically integrated dynar
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt,epsilon):
    ff1 = xx[1]
    ff2 = -xx[0]+(epsilon-xx[0]**2)*xx[1]
#    print(ff1,ff2)
    return [ff1,ff2]

# Initial conditions
x_0 = 0.01
```

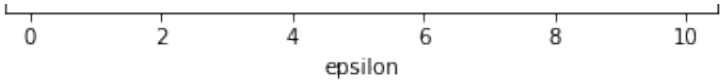
```
v_0 = 0.01
w_0 = 0.0

timestep = 0.01
timerange = 2000
Nstep = int(timerange/timestep)
a_tt = np.arange(0,timerange,timestep) # time array

# Regarding numerical integration:
# See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy..
# Also drr https://nathantypanski.com/blog/2014-08-23-ode-solver-py.html
a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))

vv = a_sol[0:Nstep,0]
ww = a_sol[0:Nstep,1]

# Now compute and plot epsilon dependence of final amplitude
# along with theoretical value 2*sqrt(epsilon)
timestep = 0.01
timerange = 5000 # need a longer time range for smaller epsilon
Nstep = int(timerange/timestep)
a_tt = np.arange(0,timerange,timestep) # time array
a_eps = []
a_vmax = []
a_thval = []
epsilon = 0.0 # re-set the initial epsilon
print('epsilon  vmax  A_avg')

for i in range(0,100):
  epsilon = epsilon + 0.1
  a_eps = np.append(a_eps,epsilon)
  a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))
  vv = a_sol[0:Nstep,0]
  vmax = np.amax(vv)
  a_vmax = np.append(a_vmax,vmax)
  thval = 2*math.sqrt(epsilon)
  a_thval = np.append(a_thval,thval)
  print('{0:0.4f} {1:3.4f} {2:3.4f}'.format(epsilon,vmax,thval))

plt.figure(0)
plt.plot(a_eps,a_vmax,'r.',a_eps,a_thval,'k')
plt.xlabel('epsilon')
plt.ylabel('vmax & theor. value')
plt.show()
```

```
epsilon  vmax  A_avg
0.1000 0.6325 0.6325
0.2000 0.8946 0.8944
0.3000 1.0960 1.0954
0.4000 1.2659 1.2649
0.5000 1.4160 1.4142
0.6000 1.5519 1.5492
0.7000 1.6772 1.6733
```

0.8000 1.7941 1.7889
0.9000 1.9042 1.8974
1.0000 2.0086 2.0000
1.1000 2.1081 2.0976
1.2000 2.2034 2.1909
1.3000 2.2949 2.2804
1.4000 2.3830 2.3664
1.5000 2.4681 2.4495
1.6000 2.5505 2.5298
1.7000 2.6303 2.6077
1.8000 2.7079 2.6833
1.9000 2.7832 2.7568
2.0000 2.8566 2.8284
2.1000 2.9280 2.8983
2.2000 2.9978 2.9665
2.3000 3.0659 3.0332
2.4000 3.1325 3.0984
2.5000 3.1976 3.1623
2.6000 3.2614 3.2249
2.7000 3.3239 3.2863
2.8000 3.3852 3.3466
2.9000 3.4454 3.4059
3.0000 3.5045 3.4641
3.1000 3.5625 3.5214
3.2000 3.6196 3.5777
3.3000 3.6757 3.6332
3.4000 3.7310 3.6878
3.5000 3.7854 3.7417
3.6000 3.8390 3.7947
3.7000 3.8918 3.8471
3.8000 3.9439 3.8987
3.9000 3.9952 3.9497
4.0000 4.0459 4.0000
4.1000 4.0959 4.0497
4.2000 4.1453 4.0988
4.3000 4.1941 4.1473
4.4000 4.2423 4.1952
4.5000 4.2899 4.2426
4.6000 4.3370 4.2895
4.7000 4.3836 4.3359
4.8000 4.4296 4.3818
4.9000 4.4752 4.4272
5.0000 4.5202 4.4721
5.1000 4.5648 4.5166
5.2000 4.6090 4.5607
5.3000 4.6527 4.6043
5.4000 4.6960 4.6476
5.5000 4.7389 4.6904
5.6000 4.7814 4.7329
5.7000 4.8235 4.7749
5.8000 4.8652 4.8166
5.9000 4.9066 4.8580
6.0000 4.9476 4.8990

```
6.1000 4.9882 4.9396
6.2000 5.0285 4.9800
6.3000 5.0685 5.0200
6.4000 5.1081 5.0596
6.5000 5.1475 5.0990
6.6000 5.1865 5.1381
6.7000 5.2252 5.1769
6.8000 5.2637 5.2154
6.9000 5.3018 5.2536
7.0000 5.3397 5.2915
7.1000 5.3773 5.3292
7.2000 5.4146 5.3666
7.3000 5.4517 5.4037
7.4000 5.4885 5.4406
7.5000 5.5250 5.4772
7.6000 5.5614 5.5136
7.7000 5.5974 5.5498
7.8000 5.6333 5.5857
7.9000 5.6689 5.6214
8.0000 5.7042 5.6569
8.1000 5.7394 5.6921
8.2000 5.7743 5.7271
8.3000 5.8090 5.7619
8.4000 5.8435 5.7966
8.5000 5.8778 5.8310
8.6000 5.9119 5.8652
8.7000 5.9458 5.8992
8.8000 5.9795 5.9330
8.9000 6.0130 5.9666
9.0000 6.0463 6.0000
9.1000 6.0795 6.0332
9.2000 6.1124 6.0663
9.3000 6.1452 6.0992
9.4000 6.1778 6.1319
9.5000 6.2102 6.1644
9.6000 6.2424 6.1968
9.7000 6.2745 6.2290
9.8000 6.3064 6.2610
9.9000 6.3381 6.2929
10.0000 6.3697 6.3246
```





9 The Rayleigh equation: an alternate type of nonlinear dammping

The van der Pol equation supposed that the damping contained a nonlinear term $-\zeta v^2$. Let us instead introduce nonlinearity via $-\zeta' \frac{1}{\omega_0^2}\left(\frac{dv}{dt}\right)^2 = -\zeta'\left(\frac{dv}{d\tilde{t}}\right)^2 = -\zeta' w^2$. Physically this is motivated by the idea that the linear dependence of damping on the rate of change of v is corrected by a term proportional to the cube of this rate of change. (A quadratic correction is excluded by symmetry considerations: the damping must oppose the rate of change and thus must be of opposite sign.)

Speculation: this might be introduced via nonlinear behavior of the resistors in the Wien bridge. CHECK THIS!!

9.1 Constructing the Rayleigh equation

$$\frac{d^2v}{d\tilde{t}^2} - \left[\epsilon - \zeta'\left(\frac{dv}{d\tilde{t}}\right)^2\right]\frac{dv}{d\tilde{t}} + v = 0.$$

We'll define the new voltage scale $v'_* \equiv \frac{1}{\sqrt{\zeta'}}$ (recalling that \widetilde t is dimensionless) so that we can write

$$\frac{d^2v}{d\tilde{t}^2} - \left[\epsilon - \frac{1}{v_*^2}\left(\frac{dv}{d\tilde{t}}\right)^2\right]\frac{dv}{d\tilde{t}} + v = 0.$$

We can now also rescale voltage so that $\widetilde{v} = \sqrt{\zeta'}v = \frac{v}{v'_*}$.

$$\frac{d^2\widetilde{v}}{d\tilde{t}^2} - \left[\epsilon - \left(\frac{d\widetilde{v}}{d\tilde{t}}\right)^2\right]\frac{d\widetilde{v}}{d\tilde{t}} + \widetilde{v} = 0.$$

As a system of two equations:

$$\frac{d\widetilde{v}}{d\tilde{t}} = \widetilde{w}$$
$$\frac{d\widetilde{v}}{d\tilde{t}} = -\widetilde{v} + (\epsilon - \widetilde{w}^2)\widetilde{w}.$$

We call this the *Rayleigh equation*. It is named after Lord Rayleigh (https://en.wikipedia.org/wiki/John_William_Strutt,_3rd_Baron_Rayleigh), the nineteenth-century Britisch physicist.

The scaled version of this equation allows a lot of different experimental conditions to be captured into a single computation of $\tilde{v}(\tilde{t})$. The actual voltage is then obtained from:

$$v_o(t) = v'_* \tilde{v}(\omega_o t).$$

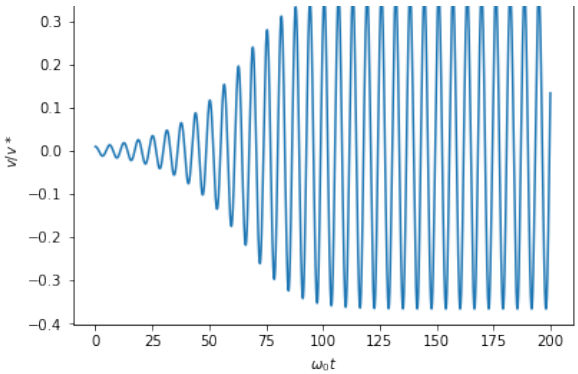
9.2 Numerically integrating the Rayleigh equation.

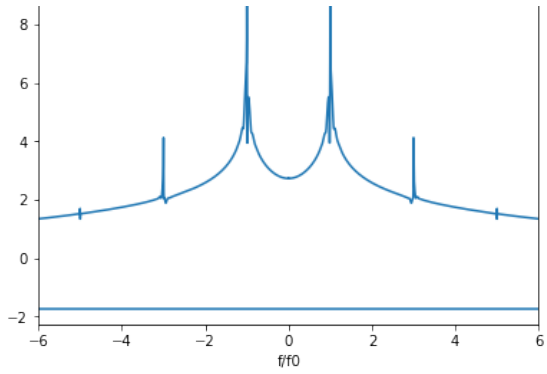
In the following, it appears empiraclly tha the amplitude scales as $7/6\sqrt{\epsilon}$. This needs to be CHECKED!! via a formal averaging mehtod similar to that used for the van der Pol equation.

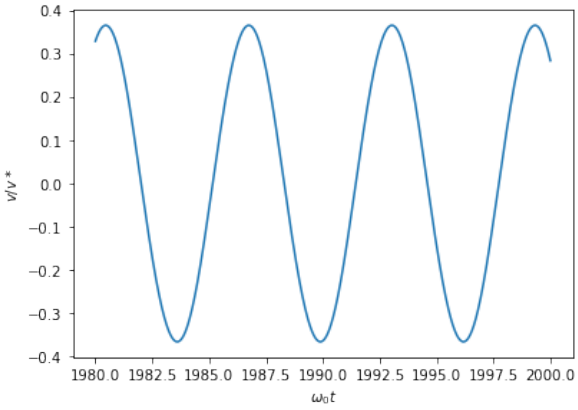
In [7]:
```
% matplotlib inline
# Code to numerically integrate the actual dynamical system
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt,epsilon):
    ff1 = xx[1]
    ff2 = -xx[0]+(epsilon-xx[1]**2)*xx[1]
#    print(ff1,ff2)
    return [ff1,ff2]

a_epsilon = [0.1,0.5] # specify an initial epsilon directly (alt value 0

# Initial conditions
v_0 = 0.01
w_0 = 0.0

timestep = 0.01
timerange = 2000
Nstep = int(timerange/timestep)
a_tt = np.arange(0,timerange,timestep) # time array

for i in range(0,2):

    epsilon = a_epsilon[i]
    print('epsilon = ',epsilon)

    # Regarding numerical integration:
    # See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/sc
    # Also drr https://nathantypanski.com/blog/2014-08-23-ode-solver-py.
    a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt,args=(epsilon,))

    vv = a_sol[0:Nstep,0]
    ww = a_sol[0:Nstep,1]

    A_avg = (7./6.)*math.sqrt(epsilon)
    dtheta = 2*math.pi/100
    vvcir = []
```

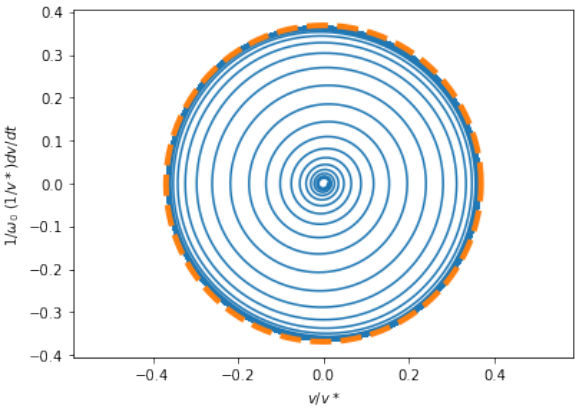
```
    wwcir = []
    for j in range(0,101):
        theta = j*dtheta
        vvcir = np.append(vvcir,A_avg*np.cos(theta))
        wwcir = np.append(wwcir,A_avg*np.sin(theta))

    #Now compute the spectrum
    #sp = np.fft.fft(vv[Nstep-4096:Nstep])
    #freq = np.fft.fftfreq(4096, d=timestep)
    sp = np.fft.fft(vv)
    freq = np.fft.fftfreq(Nstep, d=timestep)

    plt.figure(0)

    Ntransient = int(Nstep/10)
    plt.subplot(1,2,1)
    plt.plot(a_tt[0:Ntransient],vv[0:Ntransient])
    plt.xlabel('$\omega_0 t$')
    plt.ylabel('$v/v*$')

    plt.subplot(1,2,2)
    plt.plot(2*math.pi*freq, np.log10(sp.real**2+sp.imag**2))
    plt.xlim(-6,6)
    plt.xlabel('f/f0')
    #plt.ylabel('V^2/Hz')
    plt.title('Power spectrum')
    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspa
                        wspace=0.35)

    plt.figure(1)

    plt.subplot(1,2,1)
    plt.plot(a_tt[Nstep-2000:Nstep],vv[Nstep-2000:Nstep])
    plt.xlabel('$\omega_0 t$')
    plt.ylabel('$v/v*$')
    #plt.annotate('eps = {0:4.2f}'.format(epsilon),xy=(5,0.75))

    plt.subplot(1,2,2)
    plt.plot(vv,ww)
    plt.plot(vvcir,wwcir,'--',linewidth=4)
    plt.xlabel('$v/v*$')
    plt.ylabel('$1/\omega_0$ $(1/v*)dv/dt$')
    plt.axis('equal')
    #plt.annotate('eps = {0:4.2f}'.format(epsilon),xy=(5,0.75))

    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspa
                        wspace=0.35)
    plt.show()
```

```
epsilon =  0.1
```

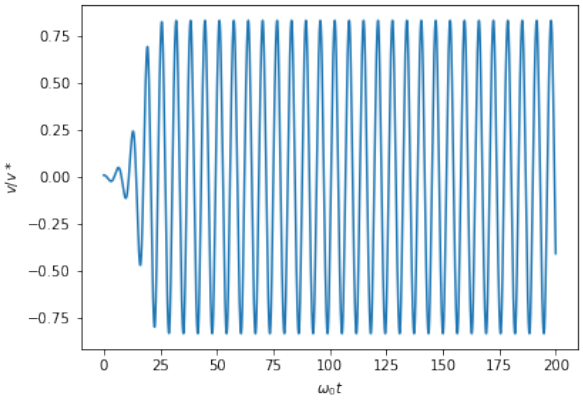


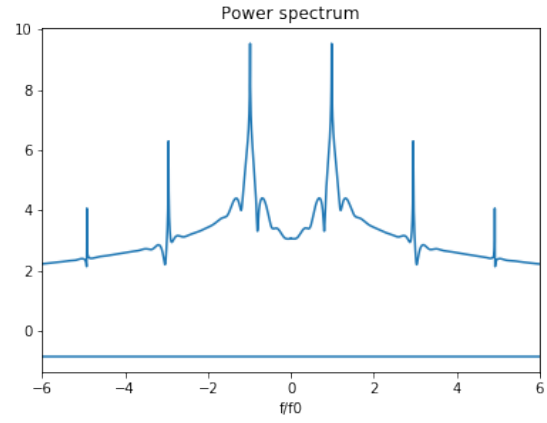


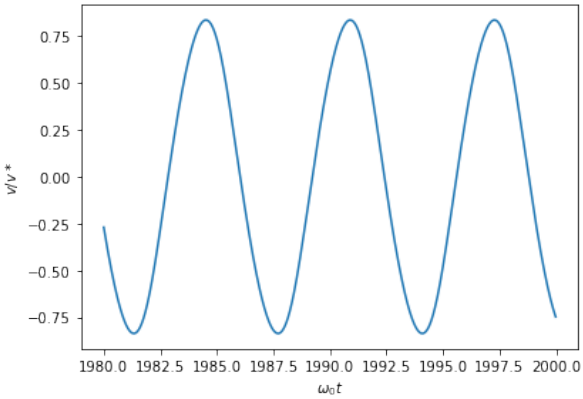


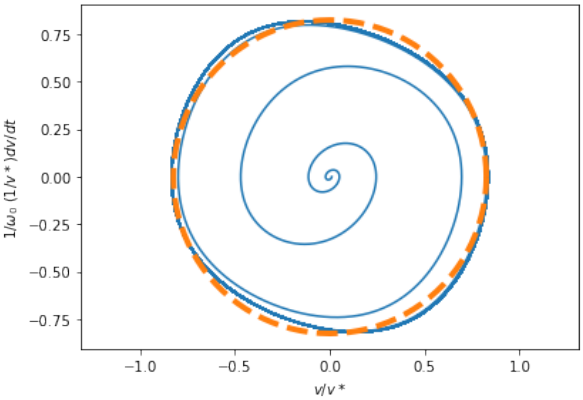


epsilon = 0.5









10 Actual implementation of the amplitude-dependent gain in the Wien-bridge oscillator circuit

gain in the Wien bridge oscillator circuit

Now we must see how to use some actual electronic devices to achieve the goal of amplitude stabilization.

10.1 Using a tungsten-filament light bulb for the gain resistor R_3

We will now show how to achieve a power-dependent adjustment to the gain of the form

$$\frac{d^2v}{dt^2} - \left[\left(\frac{R_a}{R_b} - 2\right) - f(v^2)\right]\omega_0 \frac{dv}{dt} + \omega_0^2 v = 0,$$

where $f(v^2)$ is some monotonically increasing function of the square of the output voltage. (Note: for simplicity, we drop the subscript on v_o and just let v stand for output voltage.)

A clever way to achieve this - one that goes back to the oscillator that was the first product made by Hewlett-Packard - is to use a lightbulb as the feedback resistor R_3 . As more voltage appears across this resistor, it heats up and the resistance is an increasing function of temperature $R_b(T)$.

$$\frac{d^2v}{dt^2} - \left[\frac{R_a}{R_b(T)} - 2\right]\omega_0 \frac{dv}{dt} + \omega_0^2 v = 0,$$

Let $R_{b0} \equiv R_b(T_0)$, where T_0 is the ambient temperature, i.e. the temperature of the lightbulb before any self-heating occurs. Also define $\Delta R_b \equiv R_b(T) - R_{b0}$. Then formally we can write

$$\frac{d^2v}{dt^2} - \left[\left(\frac{R_a}{R_{b0}} - 2\right) - \left(\frac{R_a}{R_{b0}} - \frac{R_a}{R_{b0} + \Delta R_b(v_0^2)}\right)\right]\omega_0 \frac{dv}{dt} + \omega_0^2 v = 0.$$

Thus

$$f(v^2) = \frac{R_a}{R_{b0}} - \frac{R_a}{R_{b0} + \Delta R_b(v^2)}.$$

So we now need to model how the resistance R_b changes with temperature and how the higher temperature $T > T_0$ occurs as a result of a non-zero output voltage v. We will see that this is a dynamic process and thus the overall dynamical system will include both electrical and thermal behaviors.

When the tungsten filament of a lightbulb experiences voltages for which the lightbulb is designed, the filament can heat to incandescence and the temperature rises from room temperature to around two thousand Kelvin. In typical circuit operation of the Wien bridge oscillator, the incandescent state may not be reached but still the temperature might change a large amount. Thus we will express the change in resistance to second order in the temperature difference from ambient:

$$R_b(T) = R_{b0}[1 + \alpha(T - T_0) + \beta(T - T_0)^2].$$

The differential equation for the output voltage becomes

$$\frac{d^2v}{dt^2} - \left[\frac{R_a}{R_{b0}[1+\alpha(T-T_0)+\beta(T-T_0)^2]} - 2\right]\omega_0\frac{dv}{dt} + \omega_0^2 v = 0.$$

There is now a new variable, the temperature of the filament. If the filament has heat capacity $\mathbf{C}_f$, the temperature is governed by a heat transfer equation:

$$\mathbf{C}_f\frac{d(T-T_0)}{dt} = i_b v_b - K(T-T_0) - S(T^4 - T_0^4).$$

Here i_b is the current passing through the filament resistance and v_b is the potential drop across the resistance. The product is the power dissipated in the resistance. This power is a source of thermal energy that acts to increase the filament temperature. Counteracting this are various forms of heat transfer away from the filament. The term $K(T-T_0)$ represents heat transfer by conduction and possibly convection (in the case there is some gas is inside the lightbulb): this form of heat transfer is assumed to be proportional to the temperature difference $T-T_0$. The term $S(T^4 - T_0^4)$ is due to radiative transport. The coefficient S is related to filament area A and emissibity ϵ_R by $S = \sigma\epsilon_R A$ where $\sigma = 5.670\times10^{-8}\mathrm{Wm^{-2}K^{-4}}$ is the Stefan-Boltzman constant.

The current i_b is given by

$$i_b = \frac{v}{R_a + R_b}.$$

Then the potential drop v_b is given by

$$v_b = \frac{R_b}{R_a + R_b}v.$$

Thus the dissipated power is

$$i_b v_b = \frac{R_b}{(R_a + R_b)^2}v^2.$$

Finally recall that the temperature dependence of R_b is given by

$$R_b(T) = R_{b0}[1+\alpha(T-T_0)+\beta(T-T_0)^2]$$

and divide through the heat transfer equation by the filament's heat capacity. This results in a differential equation for the filament temperature:

$$\frac{d(T-T_0)}{dt} = \frac{1}{\mathbf{C}_f}\left[\frac{R_{b0}[1+\alpha(T-T_0)+\beta(T-T_0)^2]}{\left[R_a + R_{b0}[1+\alpha(T-T_0)+\beta(T-T_0)^2]\right]^2}v_0^2 - \mathrm{K}(T-T_0) - \mathrm{S}(T^4 - \right.$$

Combine this equation for the temperature with the set of equations describing the voltage dynamics (writing the second order equation for output voltage as two first order equations).

$$\frac{dv}{dt} = \omega_0 w.$$
$$\frac{dw}{dt} = -\omega_0^2 v + \left(\frac{R_a}{R_{b0}\left[1 + \alpha(T - T_0) + \beta(T - T_0)^2\right]} - 2 \right) \omega_0 w.$$

The result is now a three-dimensional dynamical system for v, w, and T and we want to see if this can still produce stable oscillations, that is, solutions that are still limit cycles.

10.2 Obtaining thermal characteristics for the tungsten filament

Three aspects of the tungsten filament must be determined:

1. the temperature coefficents of resistance of the filament α and β;
2. the heat transfer coefficients K and S for the particular light bulb;
3. the heat capacity $\mathbf{C}_f$ of the filament.

This discussion is based upon the use of a Radio Shack model 272-1141 12 volt 25 mA small incandescent bulb for R_3 . This bulb is particularly convenient both because its operating voltage matches the common power supply level for many op amp circuits and because the bulb uses small current. Note that at full operation the current will exceed the source capability of many op amps, leading to current-limited behavior for large-amplitude oscillation. However, the operating point of most circuits can be designed so that this regime is avoided. An interesting discussion of the use of this bulb in analog circuit design is by Ronald Quan "Oh how I miss my neighborhood Radio Shack store" (posted 6/26/2015): https://www.planetanalog.com/author.asp?section_id=3161&doc_id=563980 (https://www.planetanalog.com/author.asp?section_id=3161&doc_id=563980) (accessed 12/10/2017).

10.3 Temperature coefficients for tungsten resistivity

Tungsten resistivity data and parameterization is provided in the publication P.D. Desai et al., "Electrical resistivity of selected elements," J. Phys. Chem. Ref. Data, v.13, no.4 (1984), pp. 1069-1096. Here we fit tabulated data of resistivity versus temperature to the model

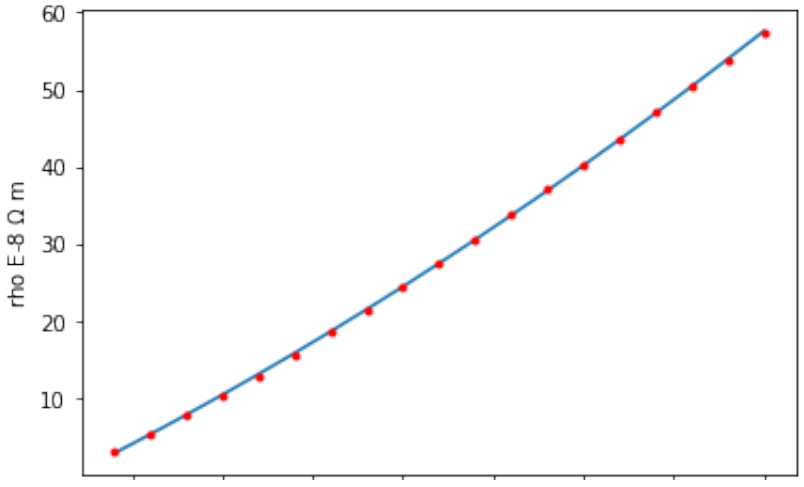
$$\rho = \rho_0[1 + \alpha(T - T_0) + \beta(T - T_0)^2]$$

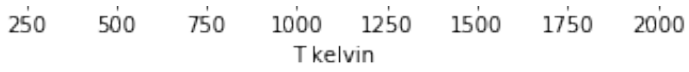
to obtain parameters α and β given a specified value of ρ_0 at ambient temperature T_0. These will be used below as initial estimates in fitting actual filament resistance data below.

In [9]:
```
% matplotlib inline
# Code to fit tungsten filament resistivity to temperature according to
#     rho = rho0*(1+alpha*(T-T0)+beta*(T-T0)**1),
# obtaining parameters alpha and beta assuming rho0 and T0 are specified

# The scipy.optimize.curve_fit module is used in this code. See
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.cu
```

```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.optimize import curve_fit

#  The following tungsten temperature and resitivity data are from P.D. 
# J. Phys. Chem. Ref. Data, v. 13, no. 4, (1984), pp. 1069-1096. See in 
Tdata=[200,300,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500,160
rhodata=[3.18,5.44,7.83,10.35,13.00,15.76,18.61,21.53,24.51,27.57,30.68,

T0=300
rho0 = 5.44

alphaguess = 4.5E-3
betaguess = 1.E-6

def rhofunc(TT,alpha1,beta1):
    DT = TT-T0
    rho = rho0*(1+alpha1*DT+beta1*DT**2)
    return rho

poptrho, pcovrho = curve_fit(rhofunc,Tdata,rhodata,p0=[alphaguess,betagu
alpha=poptrho[0]
beta = poptrho[1]

print("alpha = {0:.3e} K^-1".format(alpha))
print("beta = {0:.3e} K^-2".format(beta))

DTdata = np.subtract(Tdata,T0)
rhofit = rho0*(1+alpha*DTdata+beta*DTdata**2)

plt.plot(Tdata,rhofit,Tdata,rhodata,'r.')
plt.xlabel('T kelvin')
plt.ylabel('rho E-8 $\Omega$ m')
plt.show()
```

```
alpha = 4.578e-03 K^-1
beta = 6.215e-07 K^-2
```





10.4 Obtaining the temperature from the lightbulb resistance

Given the temperature coefficients α and β for resistivity and assuming the filament dimensions don't change significantly with temperature, these same coefficients apply to the filament resistance itself.

$$R_b = R_{b0}[1 + \alpha(T - T_0) + \beta(T - T_0)^2].$$

Now we can invert this to find temperature from the resistance:

$$\beta(T - T_0)^2 + \alpha(T - T_0) - (\frac{R_b}{R_{b0}} - 1) = 0.$$

$$T - T_0 = \frac{\alpha}{2\beta}\left[-1 + \sqrt{1 + \frac{4\beta}{\alpha^2}(\frac{R_b}{R_{b0}} - 1)}\right].$$

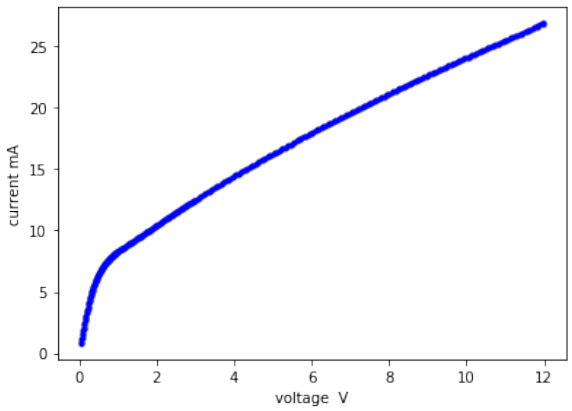
10.5 Characterizing the filament current-voltage behavior

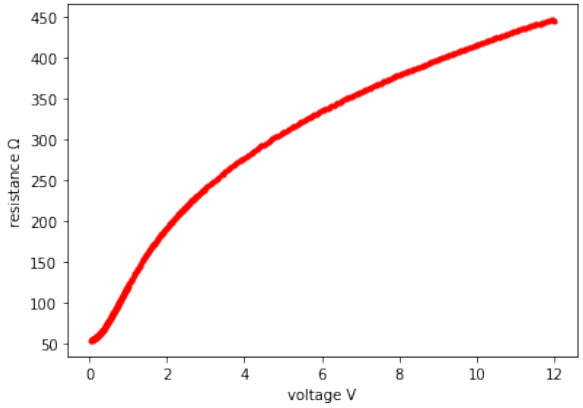
In preparation for finding the heat-transfer coefficients, a first step is to measure current versus voltage for the light bulb at steady-state operation where - for a given applied voltage across the bulb - the filament reaches a dynamic equilibrium where heat losses by conduction, convection, and radiation match the power dissipated and the temperature is constant ($dT/dt = 0.$)

10.5.1 Inspecting actual data

Care must be taken particularly to measure precise values at low voltages before radiative transport becomes significant. The following code reads in and plots data for current versus voltage. This data was taken in October 2017 by University of Colorado Denver student Courtney Fleming for the Radio Shack model 272-1141. From current-voltage data, resistance is calculated by dividing voltage by current. Then resistance is also plotted versus voltage.

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import csv

#Reading actual data to superimpose on the plot - NOT YET FINISHED
mydata = np.loadtxt('ivdata_RadioShack_272_11bulb.txt',delimiter=',')
#print(mydata)
vdata = mydata[:,0] # measured voltages (V)
idata = mydata[:,1] # measured currents (mA)
rdata = np.multiply(np.divide(vdata,idata),1000) # the factor of 1000 is

#print(vdata[0:5],idata[0:5])

plt.figure(1)
plt.subplot(1,2,1)
plt.plot(vdata,idata,'b.')
plt.xlabel('voltage  V')
plt.ylabel('current mA')

plt.subplot(1,2,2)
plt.plot(vdata,rdata,'r.')
plt.xlabel('voltage V')
plt.ylabel('resistance $\Omega$')

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspace=(
plt.show()
```





10.5.2 Relating measured voltage and resistance data

Suppose we apply a voltage v directly to the ligtbulb filament. If the filament has heat capacity C, then

$$C\frac{dT}{dt} = \frac{v^2}{R_b} - \mathrm{K}(T - T_0) - \mathrm{S}(T^4 - T_0^4).$$

In steady state, the time derivative vanishes because the power dissipation is matched by heat transfer:

$$0 = \frac{v^2}{R_b} - \mathrm{K}(T - T_0) - \mathrm{S}(T^4 - T_0^4).$$

We can re-write:

$$T^4 - T_0^4 = (T - T_0)^4 + 4T_0(T - T_0)^3 + 6T_0^2(T - T_0)^2 + 4T_0^3(T - T_0).$$

For now, we'll neglect the variation of conductance/convection parameter K and radiation parameter S (primarily through the emissivity ϵ_R contained within it) do not vary with temperature. For the wide range where the filament goes from ambient to incandescent, *a better fit to data will probably require correcting for this temperature variation of the parameters.*

Emperical data for a lightbulb filament provides v versus R_b. We will fit this as follows.

$$v^2(R_b) = R_b \left\{\mathrm{K}(T - T_0) + \mathrm{S}[(T - T_0)^4 + 4T_0(T - T_0)^3 + 6T_0^2(T - T_0)^2 + 4T_0^3(T - T_0)]\right.$$

Now insert the function

$$y(R_b) \equiv \frac{\alpha}{2\beta}\left[-1 + \sqrt{1 + \frac{4\beta}{\alpha^2}(\frac{R_b}{R_{b0}} - 1)}\right]$$

for $T - T_0$.

We have data of current I versus voltage V for a particular bulb (Radio Shack 12V 25mA mini lamp part number 272-1141). We can convert this to a data set $R_b \equiv V/I$ versus V. The goal is to fit this data with a model. Ideally we would vary parameters K and S to obtain the best fit for v^2 to $y(R_b)$:

$$v^2 = R_b \left\{\mathrm{K}y(R_b) + \mathrm{S}[(y^4(R_b) + 4T_0y^3(R_b) + 6T_0^2y^2(R_b) + 4T_0^3y(R_b)]\right\}.$$

10.5.3 Initial guess for the fitting parameters

Actual data gives an operating current of 27 mA at 12 volts. The full-on resistance at 12 volts is $R_{b\max}$ =12/0.027=444 Ω.

Experimental data for the light bulb gives a resistance at zero power dissipation R_{b0} = 54.3 Ω. Thus the ratio $R_{b\max}/R_{b0}$ = 8.177.

This implies a maximum operating temperature given by

$$T_{max} = T_0 + \frac{\alpha}{2\beta}\left[-1 + \sqrt{1 + \frac{4\beta}{\alpha^2}(\frac{R_{b\max}}{R_0} - 1)}\right]$$

Using $T_0 = 300$ K, $\alpha = 4.578 \times 10^{-3}$ K^{-1}, and $\beta = 6.215 \times 10^{-7}$ K^{-2}, this gives a value $T_{max} = 1629.5$ K.

Now we make a guess at the conductive/convective transport parameter K. Using the code below we can play around with different filament geometries and tungsten data to arrive at an estimate. Also, we can try to obtain a best fit of low temperature data using only conductive/convective transport - at least a trial-and-error "eyeball fit" to the plots (see second code to follow this cell). With this trial and error approach, we choose 3×10^{-5} W/K as an initial guess.

The total power dissipation at full operation is given by $P = iv$ = 0.027 A x 12 volt = 0.324 W = 324 mW. The resistance-temperature fit was used above to deduced a temperature of 1629K. Thus the conducted power is estimated to be $3 \times 10^{-5} \times (1629 - 300) = 39.9$ mW. This is 12% of the total. Subtract this from the total power to give 284 mW (0.284W) that must be transported radiatively.

$$iv - \mathrm{K}(T_{max} - T_0) = \mathrm{S}(T_{max}^4 - T_0^4)$$

.

This gives and estimate for the parameter S

$$\mathrm{S} = \frac{iv - \mathrm{K}(T_{max} - T_0)}{(T_{max}^4 - T_0^4)} = \frac{0.027 \times 12 - 3.0 \times 10^{-5} \times (1629 - 300)}{(1629^4 - 300^4)} = 4.034 \times 10^{-14}\ \mathrm{WK}$$

.

In [12]:
```
# This is an intermediate set of computations used to explore possible va
# Randall Tagg 12/11/2017

import math

#tungsten data
alpha = 4.578E-3 # K^(-1) temperature coefficient of resistance for tungs
beta = 6.215E-7 # K^(-2)
rho = 5.44E-8 # ohm-m resistivity of tungsten at ambient temperature T0
rhomass = 19.25E3 # Kg/m^3 density of tungsten
```

```
specificheat = 0.134 # J/gK tungsten
thermcond = 175.0 # W /(m K)) thermal conductivity of tungsten https://w

# filament data
R0 = 54.3 # room temperature resistance ohms - Radio Shack 12V 25mA (act
Rmax = 12/.027 # 444 ohm maximum resistance based on 12 mA current measu

print("Measured cold resistance = {0:.1f} ohms".format(R0))
print("Resistance at 12 volts = {0:.1f} ohms".format(Rmax))
print('  ')
print('Temperature coefficients: alpha = {0:.3e} K^-1, beta = {1:.3e} K^
#print("alpha = {0:.3e} K^-1".format(alpha))
#print("beta = {0:.3e} K^-2".format(beta))

T0=300
Tmax=T0+(0.5*alpha/beta)*(-1+math.sqrt(1+4*beta/alpha**2*(Rmax/R0-1)))
print('Maximum temperature {0:0.1f} K'.format(Tmax))
print('  ')

# filament calculations
r_fil = 2.5e-5 # estimate of filament radius in meters ...try 25 micron,
print('Chosen filament radius {0:0.3e} m'.format(r_fil))
A_fil = math.pi*r_fil**2
print('Filament cross-secional area {0:0.3e} m^2'.format(A_fil))
L_fil = R0*A_fil/rho
print('Filament length {0:0.3e} m'.format(L_fil))
V_fil = L_fil*A_fil # filament volume
m_fil = rhomass*V_fil*1000 # g mass of filament
print('Filament mass {0:0.3e} g'.format(m_fil))
C_fil = specificheat*m_fil
print('Heat capacity {0:6.3e} J/K'.format(C_fil))
L_path = 0.005*L_fil # estimated conduction path length to remove heat f
print('Estimated conduction path length {0:6.3e} m'.format(L_path))
K_est = thermcond*A_fil/L_path # W/K
print('Estimated thermal conductance {0:6.3e} W/(m K)'.format(K_est))
print('  ')


K = 3.0E-5 #chosen conduction/convection parameter
print('Chosen conduction/convection parameter K {0:.3e} W/K'.format(K))
S = (0.027*12-K*(Tmax-T0))/(Tmax**4-T0**4) # W/K^4 assume 12volt 27mA la
print('Radiation parameter S {0:.3e} W/K^4'.format(S))

T=T0+100
condpow = 1000*K*(T-T0)
radpow = 1000*S*(T**4-T0**4)
totpow = condpow+radpow
condpercent = 100*condpow/totpow
print('At 100K above ambient, conducted power is {0:.2f} mW,{1:.1f}% of

T=Tmax
condpow = 1000*K*(T-T0)
radpow = 1000*S*(T**4-T0**4)
totpow = condpow+radpow
```

```
condpercent = 100*condpow/totpow
print('At maximum temp {0:.0f} K, conducted power is {1:.1f} mW, {2:.1f}%
```

```
Measured cold resistance = 54.3 ohms
Resistance at 12 volts = 444.4 ohms

Temperature coefficients: alpha = 4.578e-03 K^-1, beta = 6.215e-07 K^-
2
Maximum temperature 1629.5 K

Chosen filament radius 2.500e-05 m
Filament cross-secional area 1.963e-09 m^2
Filament length 1.960e+00 m
Filament mass 7.408e-02 g
Heat capacity 9.927e-03 J/K
Estimated conduction path length 9.799e-03 m
Estimated thermal conductance 3.506e-05 W/(m K)

Chosen conduction/convection parameter K 3.000e-05 W/K
Radiation parameter S 4.034e-14 W/K^4
At 100K above ambient, conducted power is 3.00 mW,80.9% of 3.71 mW tot
al.
At maximum temp 1629 K, conducted power is 39.9 mW, 12.3% of 324 mW to
tal.
```

10.5.4 Nonlinear least squares fit of the voltage-resistance relation

The code below uses the scipy.optimize.curve_fit module to do a nonlinear least-squares fit of the transport model relating voltage υ to bulb resistance R_b, using the above coefficients α, β, K and S as initial guesses.

The fitting relation is:

$$\upsilon^2 = R_b \left\{ \mathrm{K} y(R_b) + \mathrm{S}[(y^4(R_b) + 4T_0 y^3(R_b) + 6T_0^2 y^2(R_b) + 4T_0^3 y(R_b)] \right\} .$$

given the defintion

$$y(R_b) \equiv \frac{\alpha}{2\beta}\left[-1 + \sqrt{1 + \frac{4\beta}{\alpha^2}(\frac{R_b}{R_{b0}} - 1)}\right] .$$

See the resulting plot below the code. The plot shows υ^2 versus R since this was the most straightforward way to employ the fitting parameters.

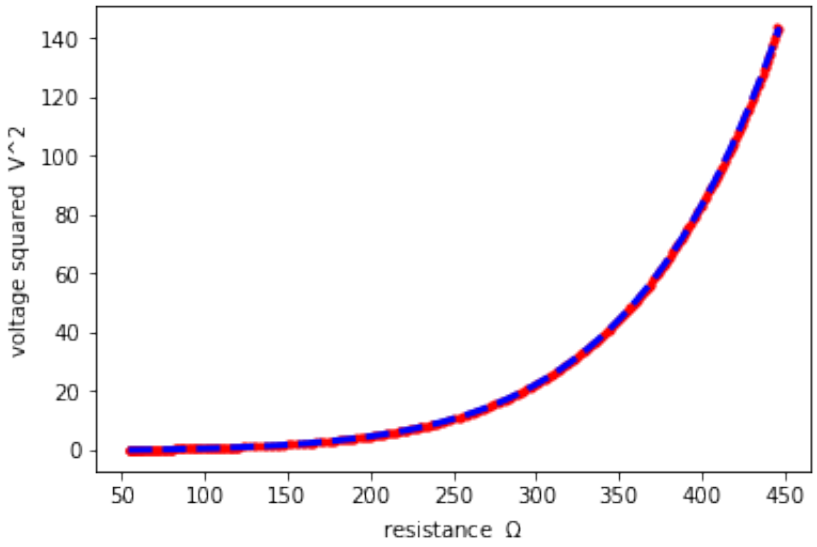
NOTE: this code reads a file called tungstendata.txt, which must be imported to the same directory as this Jupyter notebook. [Click here to access an online version. (https://sites.google.com/site/experimentalphysicsdecathlon/home/01-mechanics-and-nonlinear-dynamics/nonlinear-electronic-oscillator/tungstendata.txt?attredirects=0&d=1)](https://sites.google.com/site/experimentalphysicsdecathlon/home/01-mechanics-and-nonlinear-dynamics/nonlinear-electronic-oscillator/tungstendata.txt?attredirects=0&d=1)

In [13]:
```
% matplotlib inline
# This fits square of voltage to resistance to obtain tempeture coeffici
```

```
# as well as conductive and radiative transfer parameters K and S.
# Randall Tagg 3/12/2018

# The scipy.optimize.curve_fit module is used in this code. See
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.cu

import numpy as np
import matplotlib.pyplot as plt
import math
import csv
from scipy.optimize import curve_fit

alphaguess=4.578E-3 # linear temperature coefficient for resistance in K
betaguess = 6.215E-7 # quadratic temperature coefficient for resistance
print("alphaguess = {0:.3e} K^-1".format(alphaguess))
print("betaguess = {0:.3e} K^-2".format(betaguess))

T0 = 300 # ambient temperature in kelvin
R0 = 54.3 # limiting resistance in ohms at 0 volts
vmax = 12.0 # maximum operating voltage (volts)
imax = 0.027 # measured current at maximum operating voltage (amps)
Rmax = vmax/imax
Tmax=T0+(0.5*alphaguess/betaguess)*(-1+math.sqrt(1+4*betaguess/alphaguess
Kguess = 3.0E-5 # estimated conductive/convective transport coefficient
Sguess = (imax*vmax-Kguess*(Tmax-T0))/(Tmax**4-T0**4) # radiative transp
print('Guessed conduction parameter K {0:.3e} W/K'.format(Kguess))
print('Guessed radiation parameter S {0:.3e} W/K^4'.format(Sguess))
print()

def vfunc(R,K,S,alpha,beta):
    y = (0.5*alpha/beta)*(-1+np.sqrt(1+4*beta/alpha**2*(R/R0-1)))
    v2 = R*(K*y+S*(y**4+4*T0*y**3+6*T0**2*y**2+4*T0**3*y))
    #v = np.sqrt(v2)
    return v2

#Reading data
mydata = np.loadtxt('tungstendata.txt',delimiter=',')
vdata = mydata[:,0]
Rdata = mydata[:,1]

v2data = vdata**2

popt, pcov = curve_fit(vfunc,Rdata,v2data,p0=[Kguess,Sguess,alphaguess,be
Kfit = popt[0]
Sfit = popt[1]
alphafit = popt[2]
betafit = popt[3]
print('Fitted linear temperature coefficient alpha {0:.3e} /K'.format(al
print('Fitted linear temperature coefficient beta {0:.3e} /K^2'.format(be
print('Fitted conduction parameter K {0:.3e} W/K'.format(Kfit))
print('Fitted radiation parameter S {0:.3e} W/K^4'.format(Sfit))

v2 = vfunc(Rdata,Kfit,Sfit,alphafit,betafit)
```

```
plt.plot(Rdata,v2data,'r.')
plt.plot(Rdata,v2,'b--',linewidth=3)
plt.xlabel('resistance  $\Omega$')
plt.ylabel('voltage squared  V^2')

plt.show()
```

```
alphaguess = 4.578e-03 K^-1
betaguess = 6.215e-07 K^-2
Guessed conduction parameter K 3.000e-05 W/K
Guessed radiation parameter S 4.034e-14 W/K^4

Fitted linear temperature coefficient alpha 3.126e-03 /K
Fitted linear temperature coefficient beta -9.291e-08 /K^2
Fitted conduction parameter K 1.610e-05 W/K
Fitted radiation parameter S 4.607e-15 W/K^4
```



10.5.5 Re-plotting fitted voltage-resistance data

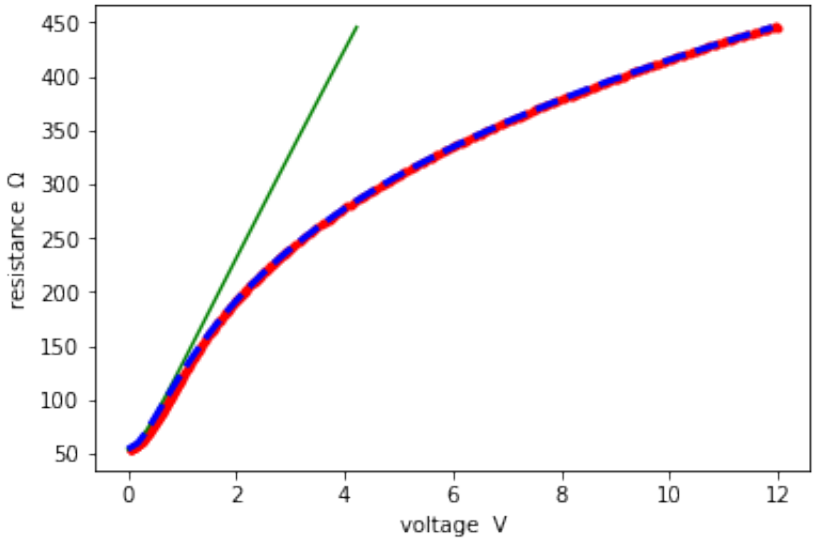
Now let's return to a view corresponding to the way the data was originally presented as resistance versus voltage

In [14]:
```
% matplotlib inline
# This calculates and plots the modeled resistance versus voltage and co
# Randall Tagg 12/11/2017

# Plotting the field
import numpy as np
import matplotlib.pyplot as plt
import math
import csv

R0=54.3 # limiting resistance at 0 volts
vmax = 12.0 # maximum operating voltage
```

```
imax = 0.027 # measured current at maximum operating voltage
Rmax = vmax/imax

T0 = 300 # ambient temperature
#alpha = 4.578E-3 #from direct fit to NBS resistivity table
#beta = 6.215E-7 #from direct fit to NBS resistivity table
alpha = 3.126E-3
beta = -9.291E-8
print("alpha = {0:.3e} K^-1".format(alpha))
print("beta = {0:.3e} K^-2".format(beta))

Tmax=T0+(0.5*alpha/beta)*(-1+math.sqrt(1+4*beta/alpha**2*(Rmax/R0-1)))
print('Tmax = {0:.1f}'.format(Tmax))

#K = 3.0E-5 # estimated conductive/convective transport coefficient
#S = (0.027*12-K*(Tmax-T0))/(Tmax**4-T0**4) # radiative transport coeffi
K = 1.610E-5
S = 4.607E-15
print('Conduction parameter K {0:.3e} W/K'.format(K))
print('Radiation parameter S {0:.3e} W/K^4'.format(S))
print()

rratio = np.arange(1.0,8.21,0.1) # resistor ratio array
y = (0.5*alpha/beta)*(-1+np.sqrt(1+4*beta/alpha**2*(rratio-1)))

R=R0*(1+alpha*y+beta*y**2)

v2 = R*(K*y+S*(y**4+4*T0*y**3+6*T0**2*y**2+4*T0**3*y))
v = np.sqrt(v2)   #voltage obtained from the complete transport model fo
vK2 = R*K*y
vK = np.sqrt(vK2) #this is the voltage that would appear for a given res
                  # was conductive

#print('v       vK     R      T')
#for i in range(0,73):
#    print('{0:.2f}   {1:.2f}   {2:.1f}   {3:.1f}'.format(v[i],vK[i],R[i

#Reading actual data to superimpose on the plot - NOT YET FINISHED
mydata = np.loadtxt('tungstendata.txt',delimiter=',')
vdata = mydata[:,0]
Rdata = mydata[:,1]
#print(vdata[0:5],Rdata[0:5])
plt.plot(vdata,Rdata,'r.')

#Now plot the model
#plt.plot(v,R)
plt.plot(vK,R,'g')
plt.plot(v,R,'b--',linewidth=3)
plt.xlabel('voltage  V')
plt.ylabel('resistance  $\Omega$')


#print(mydata)
#with open('tungstendata2.csv',newline='') as csvfile:
```

```
#with open('tungstendata2.csv',newline='') as csvfile:
#    reader=csv.reader(csvfile)
#    for row in reader:
#        print(row)

plt.show()
```

```
alpha = 3.126e-03 K^-1
beta = -9.291e-08 K^-2
Tmax = 2781.5
Conduction parameter K 1.610e-05 W/K
Radiation parameter S 4.607e-15 W/K^4
```



10.5.6 Discussion of the fitted resistance-voltage plot

The thick red cure (actually red dots) is actual data. The dashed blue curve is the model. The green curve above shows the resistance-voltage curve obtained if only conductive/convective transport were present using the assumed value of the parameter K. We see that the conductive portion is perhaps good to about 1 volt. If conduction were the only manner of heat transfer, then the temperature (and thus resistance) would rise much faster with voltage. Over the whole range of data up to the lamp's rated voltage (12V), the fit of the model that includes both conductive/convective and radiative transport is now very good, indicated by the overlap of the dashed blue curve (the model) with the data (closely packed red dots). We will use the fitted parameters below in the numerical simulations of the full electronic/filament dynamical system.

10.6 Light bulb heat capacity

THIS SECTION IS UNDER DEVELOPMENT. It will discuss how to use measured step response of the light bulb filament to estimate the heat capacity.

Recall that the time dependence of the filament temperature is given by

$$C\frac{dT}{dt} = \frac{v^2}{R_{b0}[1+\alpha(T-T_0)+\beta(T-T_0)^2]} - \mathrm{K}(T-T_0) - \mathrm{S}(T^4 - T_0^4).$$

We will estimate $\mathbf{C}_f$ from the step response of the temperature to a small change in voltage from 0 volts, thus assuming $T - T_0$ is small and keeping only linear order:

$$C\frac{dT}{dt} = \frac{v^2}{R_{b0}}[1-\alpha(T-T_0)] - \mathrm{K}(T-T_0) - \mathrm{S}4T_0^3(T-T_0).$$

*More to come... *

11 Numerical integration of the full dynamical system

Now we can explore the full dynamical system. This system includes the effects of self-heating on one of the feedback resistors (R_b) and consists of equations for the time derivatives of the variables $v \equiv v_o$, $w \equiv \frac{1}{\omega_0}\frac{dv}{dt}$, and T.

$$\frac{dv}{dt} = \omega_0 w.$$

$$\frac{dw}{dt} = -\omega_0 v + \left(\frac{R_a}{R_{b0}\left[1+\alpha(T-T_0)+\beta(T-T_0)^2\right]} - 2\right)\omega_0 w.$$

$$\frac{d(T-T_0)}{dt} = \frac{1}{\mathbf{C}_f}\left[\frac{R_{b0}[1+\alpha(T-T_0)+\beta(T-T_0)^2]}{\left\{R_a + R_{b0}[1+\alpha(T-T_0)+\beta(T-T_0)^2]\right\}^2}v^2 - \mathrm{K}(T-T_0) - \mathrm{S}(T^4 -\right.$$

The system of equations admits a steady state solution for all values of the adjustable feedback resistor R_a:

$$v = 0.$$
$$w = \frac{1}{\omega_0}\frac{dv}{dt} = 0.$$
$$T = T_0.$$

When oscillation appears, this solution becomes unstable and a new limit-cycle solution is said to "bifurcate" from the steady-state solution.

11.1 Insight into the time scales and voltage scales of the dynamics

(THIS SECTION IS STILL UNDER DEVELOPMENT.)

Added to the two equations for the circuit dynamics of the voltage and its time derivative – governed by a time scale $1/\omega_0$ – is a third equation that describes the effect of the dynamics of heat transfer. This last equation is governed by a thermal time constant $\tau_f \equiv \mathbf{C}_f / \left(\mathrm{K} + 4T_0^3 \mathrm{S}\right)$. This time constant reflects the time scale at which the filament temperature would relax in the absence of resistive self-heating, according to a linearized dependence of heat transfer on the difference between filament temperature and ambient temperature $T - T_0$. So we see that the full dynamical system has two characteristic time scales.

Note that in order to arrive at the thermal time scale, we have expanded the quartic dependence of the radiative transfer on the temperature difference $T - T_0$:

$$T^4 - T_0^4 = (T - T_0)^4 + 4T_0(T - T_0)^3 + 6T_0^2(T - T_0)^2 + 4T_0^3(T - T_0).$$

(POSSIBLY TO BE ADDED: rescaling of above three equations, including development of a voltage scale.)

11.2 Full System Simulation

The code below explores the full model. As of March 13, 2018 the filament time constant of 0.25 sec has been chosen to give a pattern of squegging that is in accord (perhaps with 20%) with observed behvaior of the circuit using the Radio Shack 272-1141 lamp and operating at epsilon of 0.02 (R_a adjusted about 11 ohms above threshold).

Key findings are the pulsation envelopes called "squegging" that appear to settle out in the absence of noise - but only after more than about 12 to 15 times the filament time constant. During the squegging pulses, the trajectories in phase space appear to cross one another (see the blue and orange plot): this is simply because the plots show the three-dimensional dynamical system (v(t),w(t),T(t) projected onto the two-dimensional (v,w) plane.

Future work will explore the effect of noise, both as spikes and as steady white noise. (Some of this was explored already: spikes appear to be capable of making the squegging persist.) Other work will examine situations that can cause the squegging to persist.

In [4]:
```
% matplotlib inline
# Code to numerically integrate the actual dynamical system
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy import integrate
import math
#
```

```
T0 = 300           # ambient temperature K
Rb0 = 54.3         # cold resistance ohms - Radio Shack 12V 25mA mini lam
alpha = 3.126E-3   # linear temperature coefficient for resistance in K^-
beta = -9.291E-8   # quadratic temperature coefficient for resistance in
K = 1.610E-5       # W/K estimated conductive/convective transport coeffi
S = 4.607E-15      # W/K^4 estimated radiative transport coefficient
tau_fil = 0.25      # seconds estimated thermal time constant of filament
#tau_fil = 5.0      # seconds estimated thermal time constant of filament

C_fil = tau_fil*(K+4*T0**3*S)
#print(C_fil,' J/K')

def Rb(T):
    Rb = Rb0*(1+alpha*(T-T0)+beta*(T-T0)*(T-T0))
    return Rb

#alpha=4.578E-3 # linear temperature coefficient for resistance in K^-1
#beta = 6.215E-7 # quadratic temperature coefficient for resistance in K
##K = 2.0E-5 # W/K estimated conductive/convective transport coefficient
##S = 2.491e-14 # W/K^4 estimated radiative transport coefficient
#K = 3.0E-5 # W/K estimated conductive/convective transport coefficient
#S = 4.823e-14 # W/K^4 estimated radiative transport coefficient

R = 6.7E3
C = 10.0E-9
omega0 = 1/(R*C)
f0 = omega0/(2*math.pi)
period = 1./f0
print('Oscillator period = {0:9.6f} sec'.format(period))

epsilon = 0.02
Ra = Rb0*(2.0+epsilon)

sigma = 0.5*epsilon*omega0
print('Growth time 1/sigma = {0:9.6f} sec'.format(1./sigma))
# tau = period/(math.pi*epsilon) # alternate way to calculate growth tim
# print('Growth time 1/sigma = {0:9.6f} sec'.format(tau))

vfixapprox = K*(Ra + Rb0)**2/Rb0

vspike = 0. # a voltage spike introduced every 1/60 second; set to zero

timestepfactor = 0.01
timerangefactor = 10000 #15000
#timerangefactor = 40000
timestep = timestepfactor*period
timerange = timerangefactor*period
Nstep = int(timerangefactor/timestepfactor)
print(Nstep,'time steps  ','{0:6.3f} sec total time'.format(timerange) )

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt):
    ff0 = omega0*xx[1]
    ff1 = (Ra/Rb(xx[2])-2)*omega0*xx[1]-omega0*xx[0]
```

```
#    omega = omega0*(1-0.2*xx[0]) # we assume that the frequency-determi
    ff2 = ((Rb(xx[2])/(Ra+Rb(xx[2])))**2)*xx[0]**2-K*(xx[2]-T0)-S*(xx[2]*
#    if (tt >= 0.01666666667) and (tt%0.01666666667) < timestep: # intro
#        ff0 = ff0 + vspike/timestep
    return [ff0,ff1,ff2]

a_tt = np.arange(0,timerange,timestep) # time array
# See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.
# Also https://nathantypanski.com/blog/2014-08-23-ode-solver-py.html

# Initial conditions
v_0 = 0.01
w_0 = 0.0
T_0 = T0

a_sol = integrate.odeint(dynsys,[v_0,w_0,T_0],a_tt)
vv = a_sol[0:Nstep,0]
ww = a_sol[0:Nstep,1]
TT = a_sol[0:Nstep,2]-T0

Ntenper = int(10*period/timestep)

# find the final steady filament temperature
TTavg = np.sum(TT[Nstep-Ntenper:Nstep])/(Ntenper)
print('<T-T0> = {0:6.3f} K'.format(TTavg))

# find the final steady amplitude measured by the signal maximum
# after squegging transients have settled
vmax = np.amax(vv[Nstep-Ntenper:Nstep])
print('vamplitude = {0:5.3f}'.format(vmax))

# Plot a long-time view of the dynamics, showing transient squegging
# and eventual oscillation at a steady amplitude
plt.figure(1)
plt.plot(a_tt,vv)
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('Long time view showing squegging')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {
            .format(epsilon,1000*period,1000*tau_fil,vspike))

# Plot the corresponding long-time view of the filament temperature
plt.figure(2)
plt.plot(a_tt,TT)
plt.xlabel('t (s)')
plt.ylabel('T-T0 (K)')
plt.title('Filament temperature variation')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {
            .format(epsilon,1000*period,1000*tau_fil,vspike))

# plot the first and start of the second squegging pulse
plt.figure(3)
Nshort = int(0.4*tau_fil/timestep)
plt.plot(a_tt[0:4*Nshort],vv[0:4*Nshort])
```

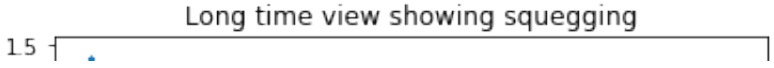
```
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('First and start of second  squegging pulse - volts')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {
            .format(epsilon,1000*period,1000*tau_fil,vspike))

# plot a magnified version of the squegging pulse to view the small
# residual oscillation that is still occuring between the pulses
plt.figure(4)
Nshort = int(0.4*tau_fil/timestep)
plt.plot(a_tt[0:4*Nshort],vv[0:4*Nshort])
plt.ylim(-0.0001,0.0001)
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('First and start of second squegging pulse - volts')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {
            .format(epsilon,1000*period,1000*tau_fil,vspike))

# plot the temperature during the first squegging pulse
plt.figure(5)
Nshort = int(0.4*tau_fil/timestep)
plt.plot(a_tt[0:Nshort],TT[0:Nshort])
plt.xlabel('t (s)')
plt.ylabel('T-T0 (K)')
plt.title('First and start of second  squegging pulse - temperature')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {
            .format(epsilon,1000*period,1000*tau_fil,vspike))

plt.figure(6)
Na = int(0.3*Nshort)
Nb = int(0.4*Nshort)
Nc = int(0.4*Nshort)
Nd = int(0.5*Nshort)
plt.plot(vv[Na:Nb],ww[Na:Nb],vv[Nc:Nd],ww[Nc:Nd],vv[Na:Na+1],ww[Na:Na+1]
plt.xlabel('v')
plt.ylabel('1/$\omega_0$ dv/dt')
plt.title('Apparent crossing of trajectories during squegging')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {
            .format(epsilon,1000*period,1000*tau_fil,vspike))
plt.axis('equal')

plt.figure(7)
Ntenper = int(10*period/timestep)
plt.plot(a_tt[Nstep-Ntenper:Nstep],vv[Nstep-Ntenper:Nstep])
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('Steady oscillation after many filament time-constants')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {
            .format(epsilon,1000*period,1000*tau_fil,vspike))

plt.figure(8)
plt.plot(vv[Nstep-Ntenper:Nstep],ww[Nstep-Ntenper:Nstep])
plt.xlabel('v')
plt.ylabel('1/$\omega_0$ dv/dt')
```

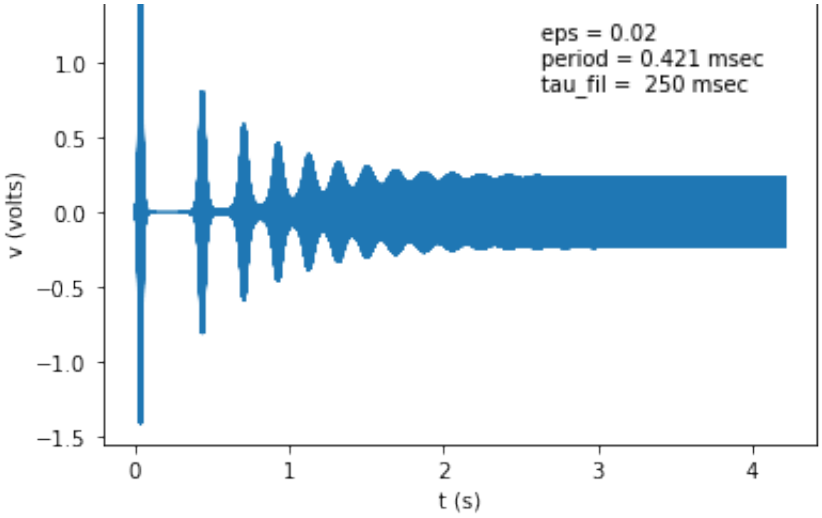
```
plt.ylabel('$d\omega_0/dt$')
plt.title('Final limit cycle')
plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3f} msec\ntau_fil = {2
            .format(epsilon,1000*period,1000*tau_fil,vspike))
plt.axis('equal')

# The following are several stages of the dynamics viewed in the
# full three-dimensional phase space
fig3D = plt.figure(9)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = 0
N3Db = int(0.3*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.title('Fist stage in building up nested tori for first two squegging
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3I
plt.savefig("figure8.pdf")

fig3D = plt.figure(10)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(0.3*Nshort)
N3Db = int(0.45*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3I
plt.savefig("figure9.pdf")

fig3D = plt.figure(11)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(0.45*Nshort)
N3Db = int(0.6*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3I
plt.savefig("figure10.pdf")

fig3D = plt.figure(12)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(0.6*Nshort)
N3Db = int(0.75*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3I
plt.savefig("figure11.pdf")

fig3D = plt.figure(13)
ax = fig3D.add_subplot(111, projection='3d')
```

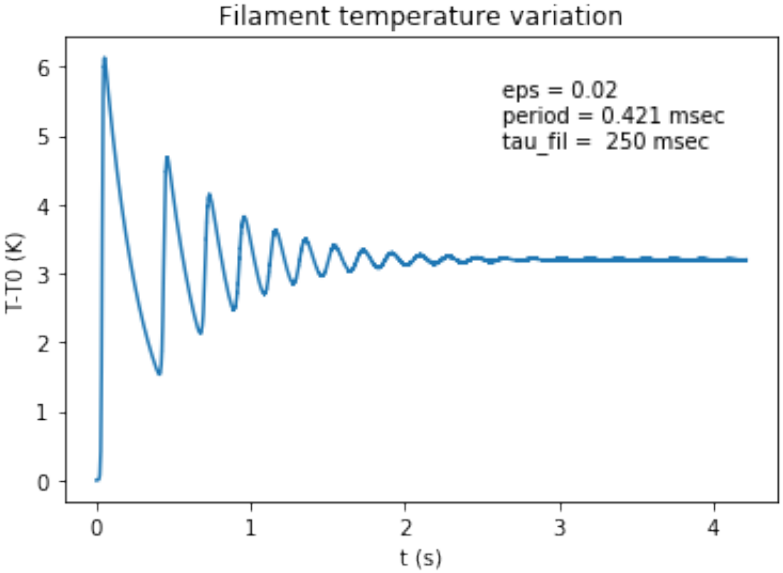
```
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(0.75*Nshort)
N3Db = int(4.3*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure12.pdf")

fig3D = plt.figure(14)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(4.3*Nshort)
N3Db = int(4.6*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure13.pdf")

fig3D = plt.figure(15)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(4.6*Nshort)
N3Db = int(4.75*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure14.pdf")

fig3D = plt.figure(16)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(4.75*Nshort)
N3Db = int(5.00*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure15.pdf")

fig3D = plt.figure(17)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(5.00*Nshort)
N3Db = int(7.00*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure16.pdf")

fig3D = plt.figure(18)
```

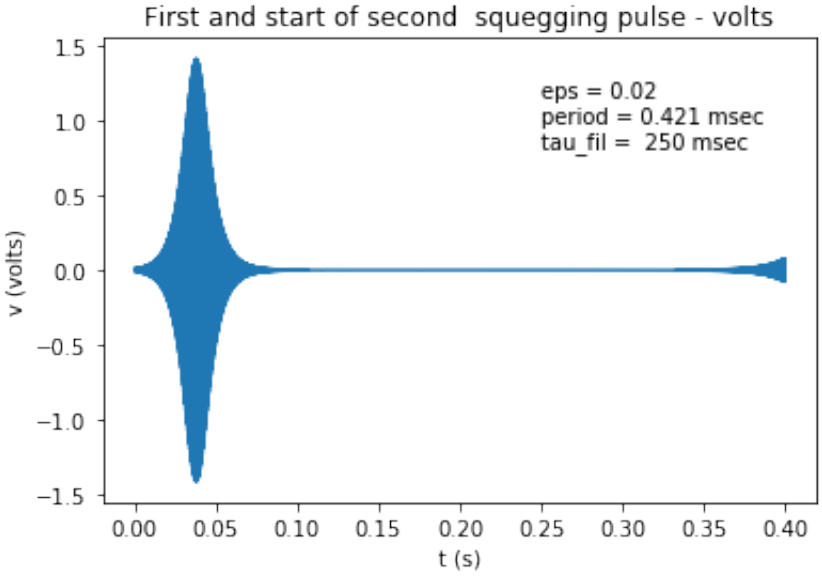
```
fig3D = plt.figure(18)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(7.00*Nshort)
N3Db = int(9.00*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure17.pdf")

# Plot nested tori corresponding to the first two sequegging pulses
fig3D = plt.figure(19)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = 0
N3Db = int(9.0*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.title('Nested tori from first two squegging pulses')
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure18.pdf")

# Plot the final limit cycle
fig3D = plt.figure(20)
ax = fig3D.add_subplot(111, projection='3d')
N3Da = int(40.0*Nshort)
N3Db = int(42.0*Nshort)
ax.plot(vv[N3Da:N3Db],ww[N3Da:N3Db],TT[N3Da:N3Db])
ax.set_xlim3d(-1.5,1.5)
ax.set_ylim3d(-1.5,1.5)
ax.set_zlim3d(0,6)
plt.title('Final limit cycle')
plt.figtext(0.9,0.7,'t = {0:6.3f} to {1:6.3f} s'.format(N3Da*timestep,N3
plt.savefig("figure19.pdf")

plt.show()

# The following code (commented out) is transferred to a later cell in t
# to generate the array vv.

# from IPython.display import Audio
# myrate=int(1/timestep)
# Audio(data=vv[0:1000000],rate=myrate)
```

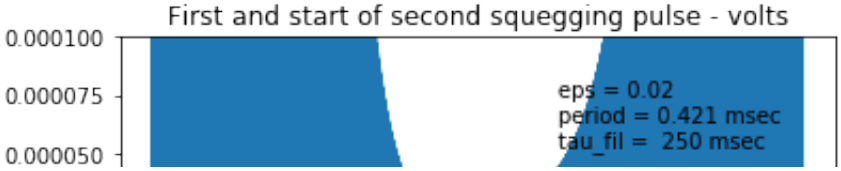
```
Oscillator period =  0.000421 sec
Growth time 1/sigma =  0.006700 sec
1000000 time steps    4.210 sec total time
<T-T0> =  3.198 K
vamplitude = 0.229
```

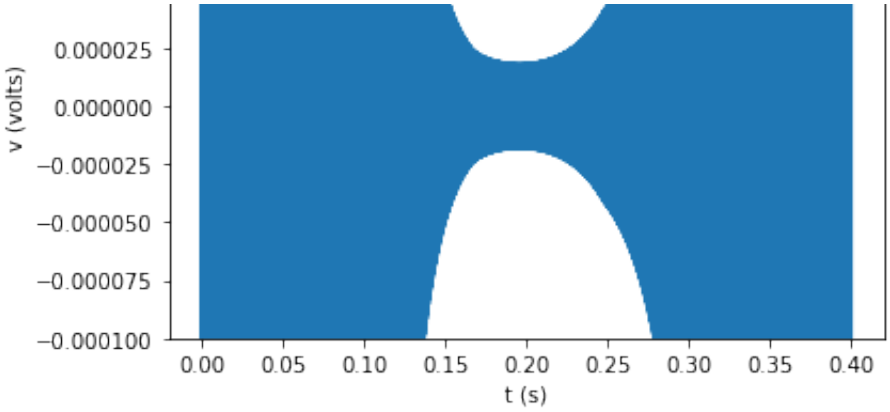


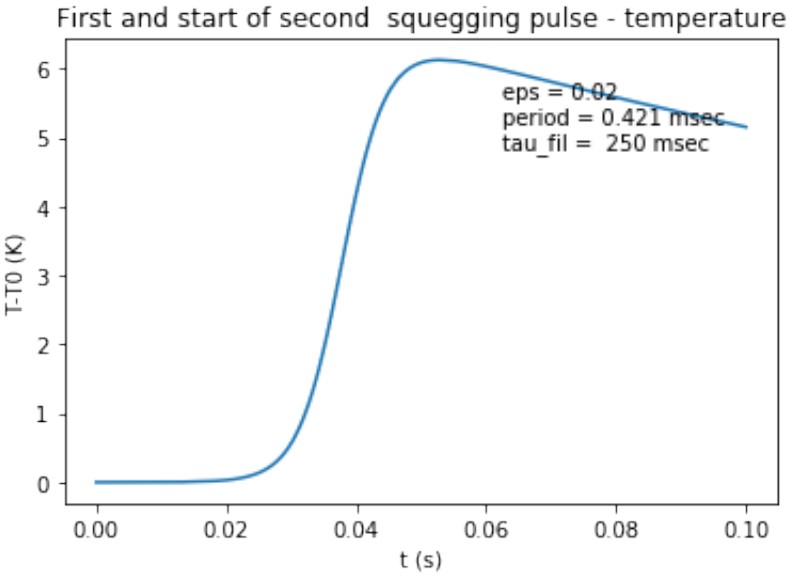

eps = 0.02
period = 0.421 msec
tau_fil = 250 msec
v (volts)
t (s)

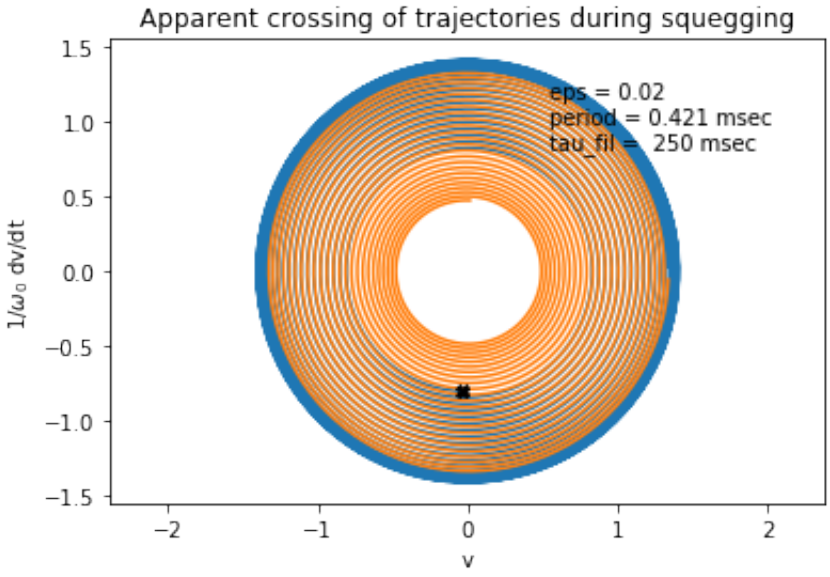

Filament temperature variation
eps = 0.02
period = 0.421 msec
tau_fil = 250 msec
T-T0 (K)
t (s)

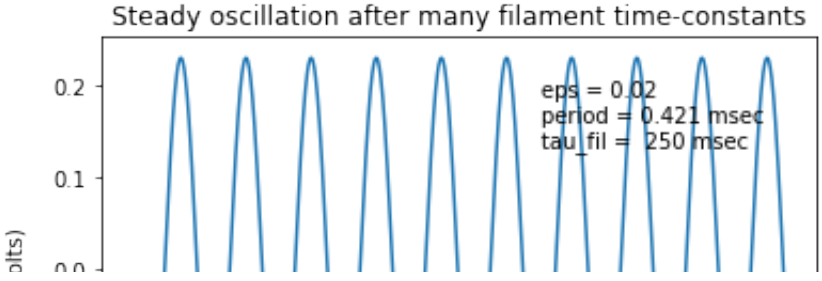

First and start of second squegging pulse - volts
eps = 0.02
period = 0.421 msec
tau_fil = 250 msec
v (volts)
t (s)

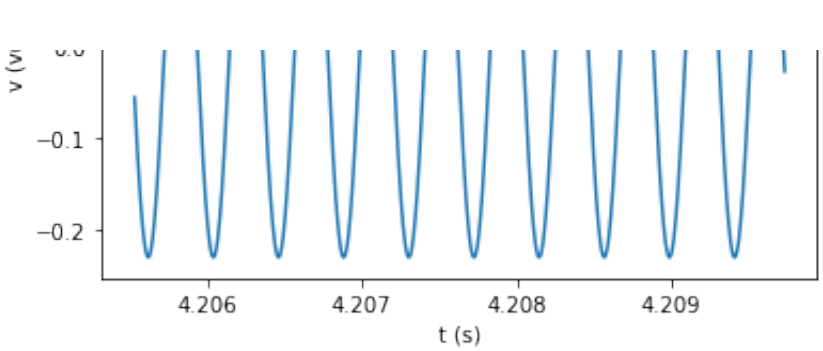

First and start of second squegging pulse - volts
eps = 0.02
period = 0.421 msec
tau_fil = 250 msec

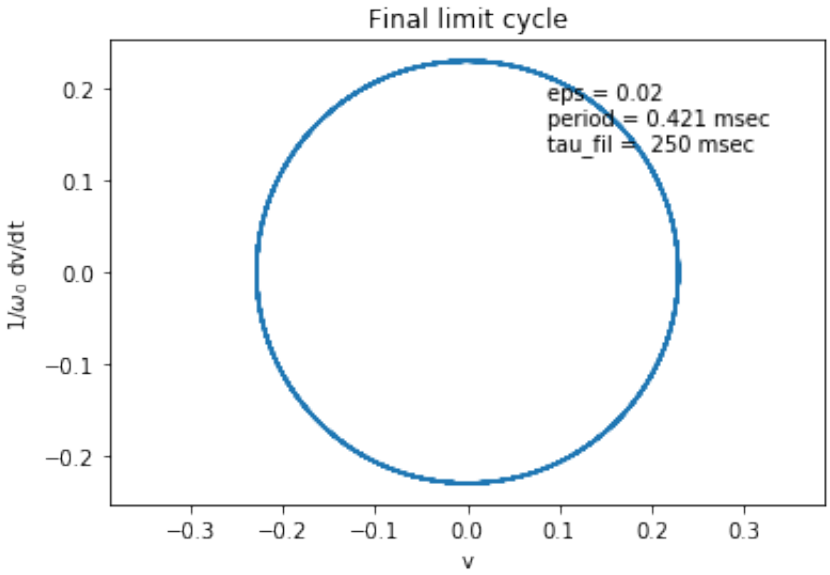

v (volts)
0.000025
0.000000
−0.000025
−0.000050
−0.000075
−0.000100
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
t (s)


First and start of second squegging pulse - temperature
eps = 0.02
period = 0.421 msec
tau_fil = 250 msec
T-T0 (K)
0
1
2
3
4
5
6
0.00
0.02
0.04
0.06
0.08
0.10
t (s)

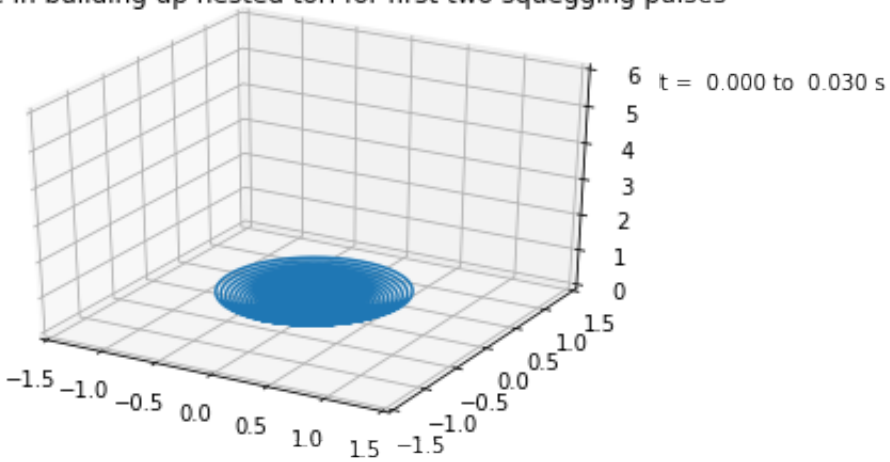

Apparent crossing of trajectories during squegging
eps = 0.02
period = 0.421 msec
tau_fil = 250 msec
1/ω₀ dv/dt
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2
v

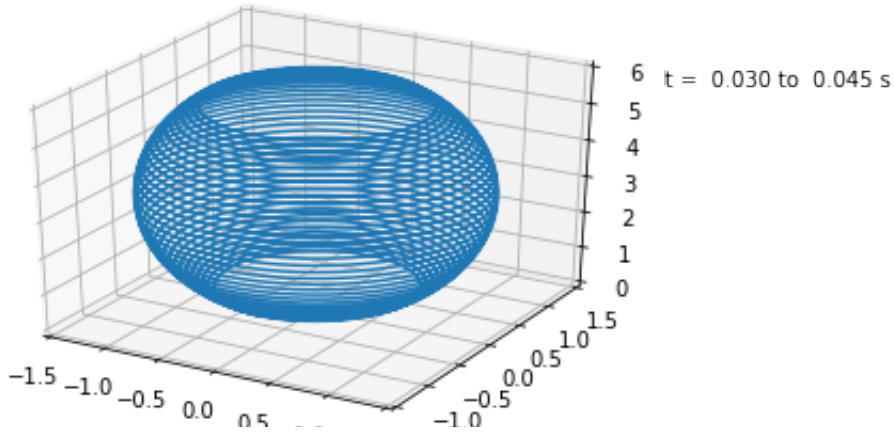

Steady oscillation after many filament time-constants
eps = 0.02
period = 0.421 msec
tau_fil = 250 msec
olts)
0.2
0.1
0.0

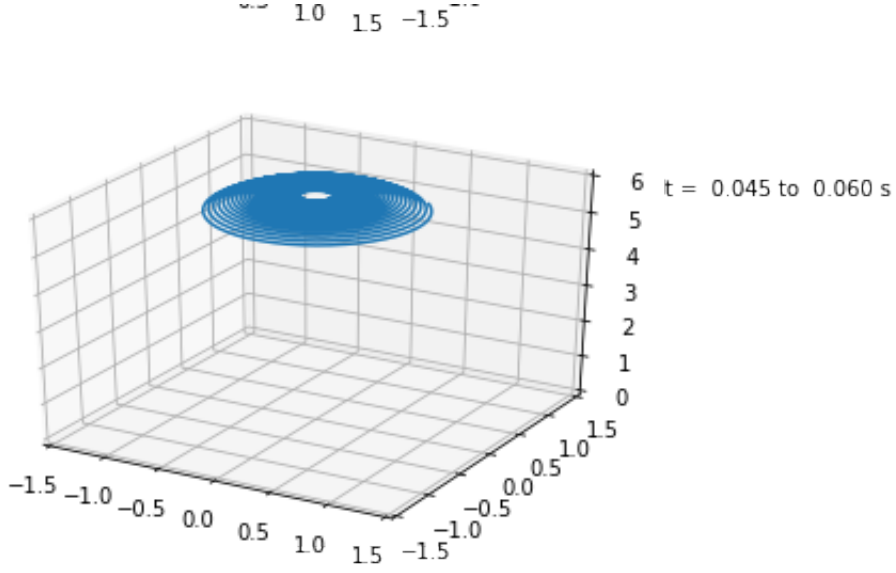


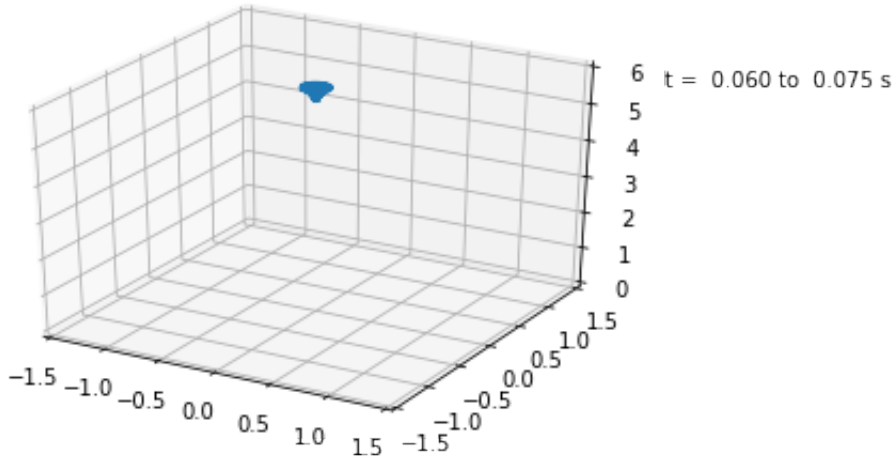


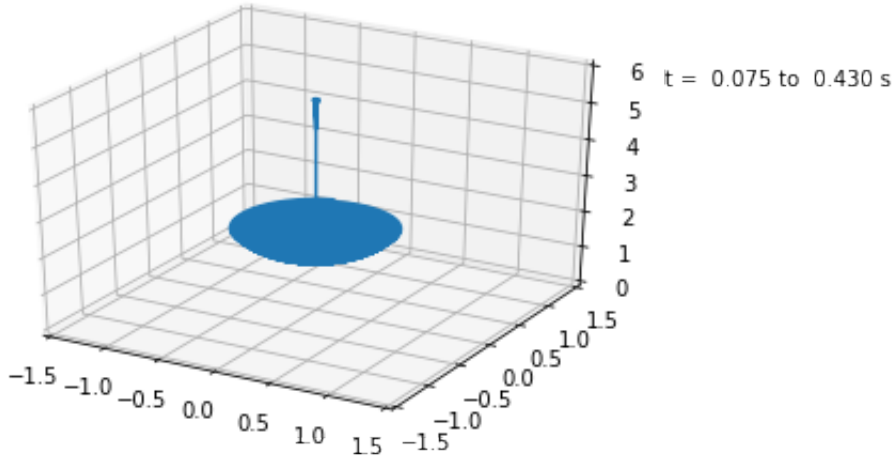
Fist stage in building up nested tori for first two squegging pulses

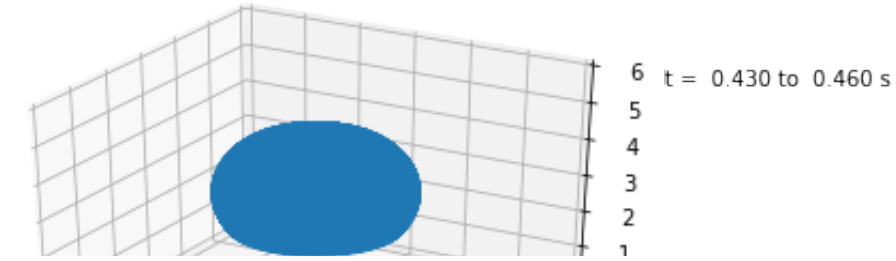


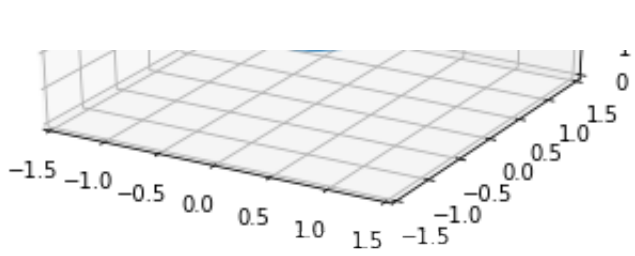


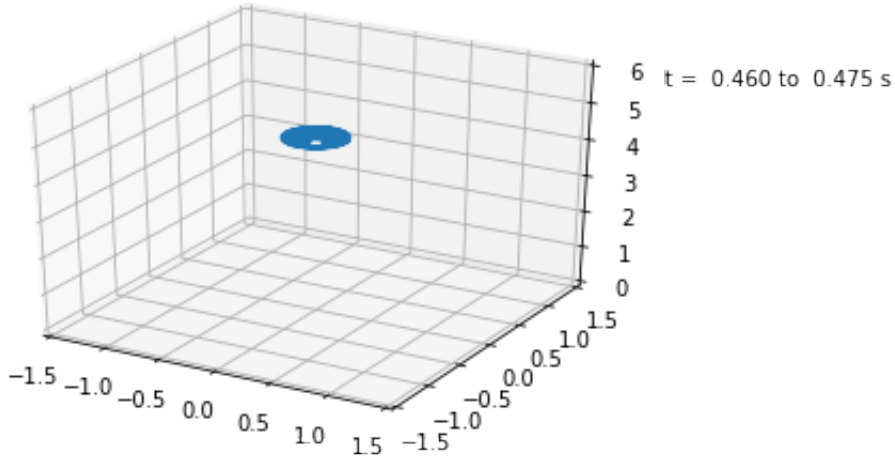

t = 0.045 to 0.060 s

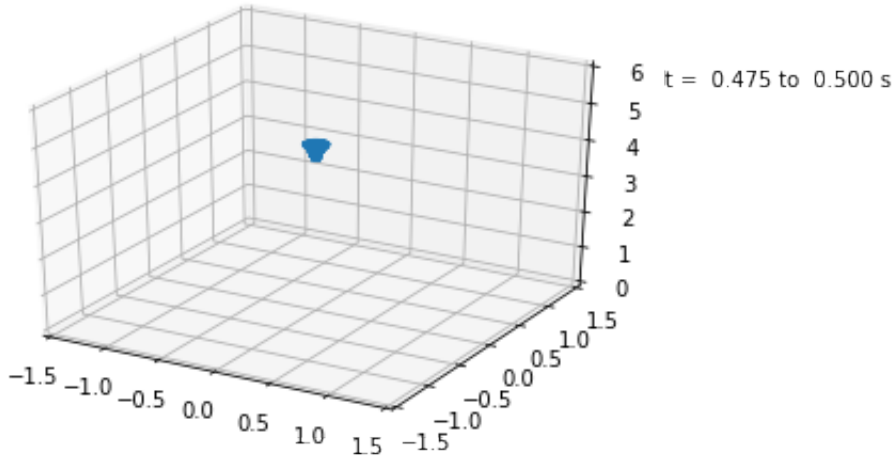

t = 0.060 to 0.075 s

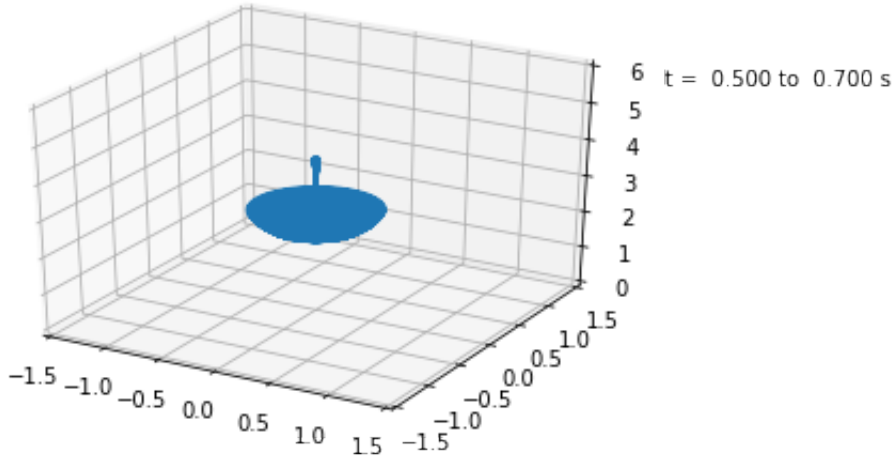

t = 0.075 to 0.430 s


t = 0.430 to 0.460 s

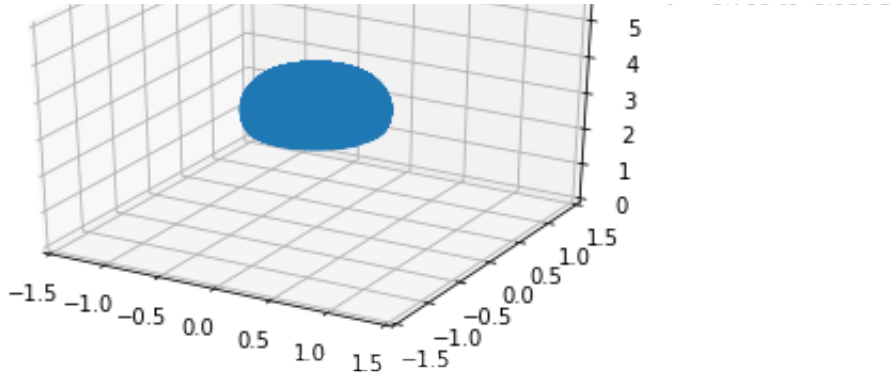

−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
0


6
5
4
3
2
1
0
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
t = 0.460 to 0.475 s

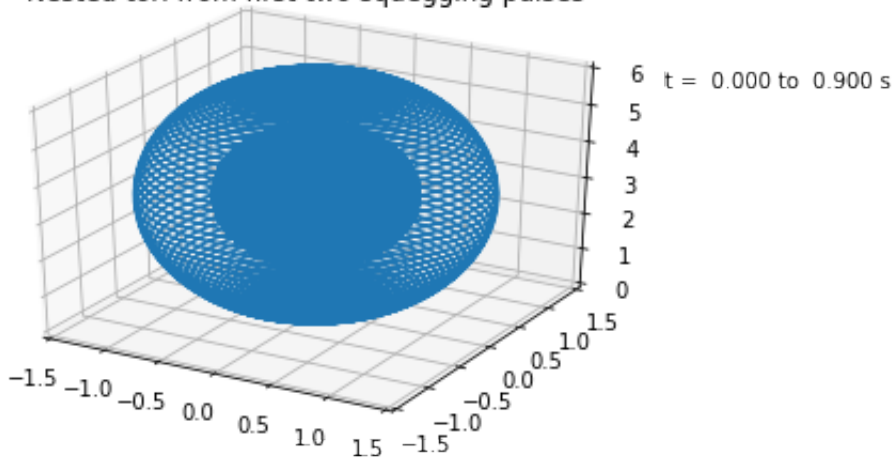

6
5
4
3
2
1
0
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
t = 0.475 to 0.500 s


6
5
4
3
2
1
0
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
t = 0.500 to 0.700 s

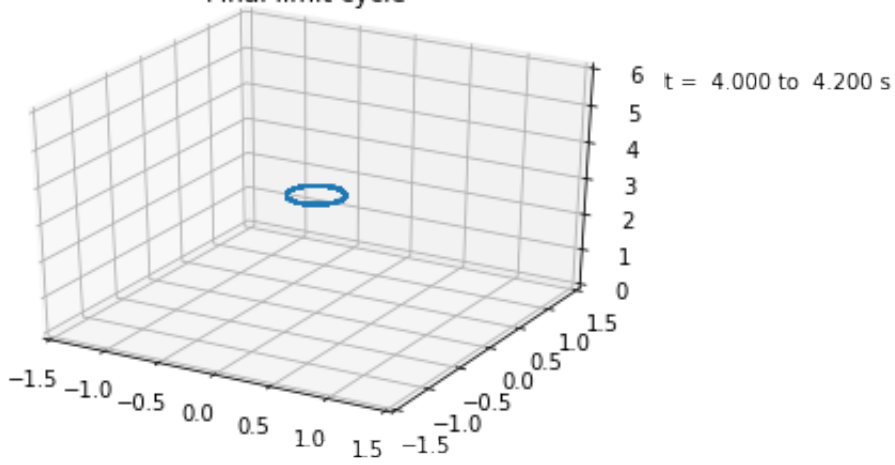

6
t = 0.700 to 0.900 s



Nested tori from first two squegging pulses



Final limit cycle



11.3 Audification of the time-series output

Run the previous code cell. Then run the cell below to create an audification of the time series array vv.

For comparison of the audio tone, link to

http://www.szynalski.com/tone-generator/ (http://www.szynalski.com/tone-generator/)

and insert the value 1/period into the frequency. (2375 Hz for 0.0004210 msec period)

In [17]:
```
from IPython.display import Audio
myrate=int(1/timestep)
Audio(data=vv[0:1000000],rate=myrate)
```

Out[17]:

11.4 Plotting amplitude versus birfucation parameter

The following computations - including linear least-squares fit to log-log data - reveal that the long-term dynamics settles to a fixed voltage amplitude that grows as $\sqrt{\epsilon}$ for an initial range $0 < \epsilon \lesssim 0.1$ and then begins to grow more quickly. The temperature rise appears to have a linear growth with ϵ even up to $\epsilon = 1$.

11.4.1 Narrow range of parameter

In [21]:
```
% matplotlib inline

# Code to numerically integrate the actual dynamical system
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from scipy import stats
import math

T0 = 300           # ambient temperature K
Rb0 = 54.3         # ohms - Radio Shack 12V 25mA mini lamp part number 27
alpha = 3.126E-3   # linear temperature coefficient for resistance in K^-
beta = -9.291E-8   # quadratic temperature coefficient for resistance in
K = 1.610E-5       # W/K estimated conductive/convective transport coeffi
S = 4.607E-15      # W/K^4 estimated radiative transport coefficient
tau_fil = 0.25     # seconds estimated thermal time constant of filament

C_fil = tau_fil*(K+4*T0**3*S)
#print(C_fil,' J/K')

def Rb(T):
    Rb = Rb0*(1+alpha*(T-T0)+beta*(T-T0)*(T-T0))
    return Rb

R = 6.7E3
C = 10.0E-9
omega0 = 1/(R*C)
f0 = omega0/(2*math.pi)
period = 1./f0
print('Oscillator period = {0:9.6f} sec'.format(period))

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt):
    ff0 = omega0*xx[1]
```

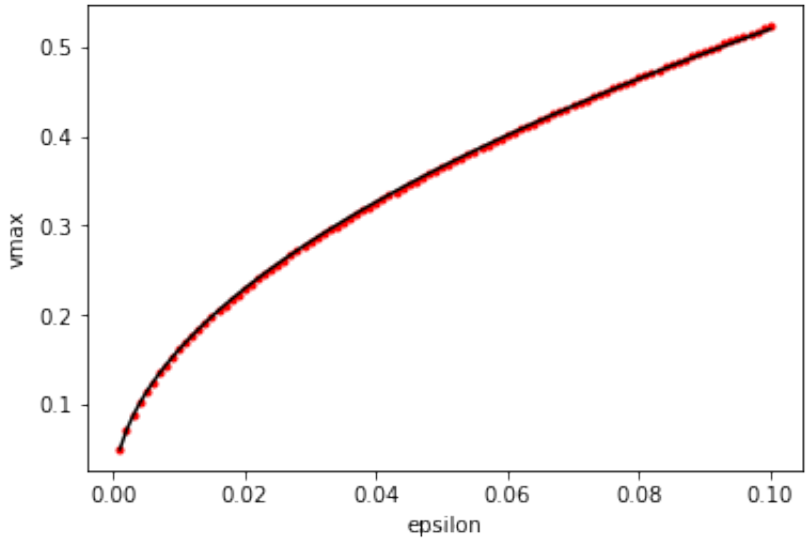
```
    ff1 = (Ra/Rb(xx[2])-2)*omega0*xx[1]-omega0*xx[0]
    ff2 = ((Rb(xx[2])/(Ra+Rb(xx[2])))**2)*xx[0]**2-K*(xx[2]-T0)-S*(xx[2]*
    return [ff0,ff1,ff2]

# Initial conditions
v_0 = 0.01
w_0 = 0.0
T_0 = T0

timestepfactor = 0.01
timerangefactor = 15000
timestep = timestepfactor*period
timerange = timerangefactor*period
Nstep = int(timerangefactor/timestepfactor)
print(Nstep,'time steps  ','{0:6.3f} sec total time'.format(timerange) )

# time array
a_tt = np.arange(0,timerange,timestep)

# See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.
# Also https://nathantypanski.com/blog/2014-08-23-ode-solver-py.html

a_eps = []
a_vmax = []
a_TTavg = []
epsilon = 0.0
#Nlimitsample = Nstep-int(Nstep/1000)
print('epsilon,vmax,<T-T0>,Rb(T),Ra/Rb(T)')

Ntenper = int(10*period/timestep)
for i in range(0,100):
    vmaxsum = 0.
    epsilon = epsilon + 0.001
    a_eps = np.append(a_eps,epsilon)
    Ra = Rb0*(2.0+epsilon)
    a_sol = integrate.odeint(dynsys,[v_0,w_0,T_0],a_tt)
    vv = a_sol[0:Nstep,0]
    TT = a_sol[0:Nstep,2]-T0
    jmax = 20
    for j in range(0,jmax): #average the maximum found over ten interval
        Nlimitupper = Nstep-j*Ntenper
        Nlimitlower = Nlimitupper - Ntenper
        vmax = np.amax(vv[Nlimitlower:Nlimitupper])
        vmaxsum = vmaxsum + vmax
    vmaxavg = vmaxsum/jmax
    a_vmax = np.append(a_vmax,vmaxavg)
    TTavg = np.sum(TT[Nstep-jmax*Ntenper:Nstep])/(jmax*Ntenper)
    a_TTavg = np.append(a_TTavg,TTavg)
    T_0 = T0 + TTavg # use longterm average temperature from this run as
    print('{0:0.4f},{1:7.4f},{2:7.3f},{3:7.3f},{4:6.3f}'.format(epsilon,v

# Compute logaritms of epsilon and vmax
logeps = np.log10(a_eps)
logvmax = np.log10(a_vmax)
```

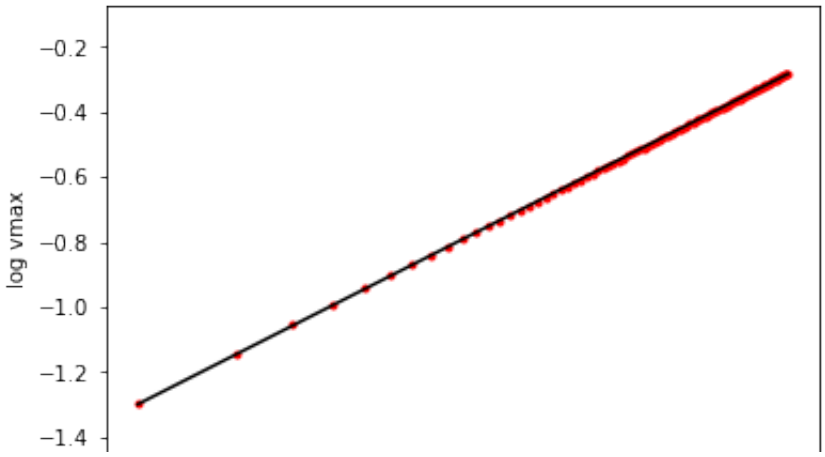
```
# Do a linear regression of the resulting logarithms
slope, intercept, r_value, p_value, std_err = stats.linregress(logeps,log
print('Logarithmic values slope={0:0.4f}, intercept={1:0.4f}'.format(slop

# Computer theoretical values
logvmaxtheorval = intercept + slope*logeps
a_vmaxtheorval = np.power(10,logvmaxtheorval)

plt.figure(0)
plt.plot(a_eps,a_vmax,'r.',a_eps,a_vmaxtheorval,'k')
plt.xlabel('epsilon')
plt.ylabel('vmax')

plt.figure(1)
plt.plot(logeps,logvmax,'r.',logeps,logvmaxtheorval,'k')
plt.xlabel('log epsilon')
plt.ylabel('log vmax')
plt.axis('equal')

plt.figure(2)
plt.plot(a_eps,a_TTavg,'r.')
plt.xlabel('epsilon')
plt.ylabel('<T-T0>')

plt.show()
```

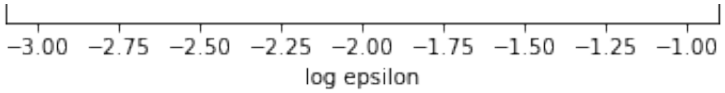
```
Oscillator period =  0.000421 sec
1500000 time steps    6.315 sec total time
epsilon,vmax,<T-T0>,Rb(T),Ra/Rb(T)
0.0010, 0.0509,  0.160, 54.327, 2.000
0.0020, 0.0721,  0.320, 54.354, 2.000
0.0030, 0.0883,  0.480, 54.381, 2.000
0.0040, 0.1020,  0.640, 54.409, 2.000
0.0050, 0.1140,  0.800, 54.436, 2.000
0.0060, 0.1249,  0.960, 54.463, 2.000
0.0070, 0.1350,  1.120, 54.490, 2.000
0.0080, 0.1443,  1.280, 54.517, 2.000
0.0090, 0.1531,  1.440, 54.544, 2.000
0.0100, 0.1615,  1.600, 54.572, 2.000
0.0110, 0.1694,  1.760, 54.599, 2.000
0.0120, 0.1770,  1.920, 54.626, 2.000
0.0130, 0.1843,  2.079, 54.653, 2.000
0.0140, 0.1913,  2.239, 54.680, 2.000
0.0150, 0.1981,  2.399, 54.707, 2.000
0.0160, 0.2046,  2.559, 54.734, 2.000
0.0170, 0.2110,  2.719, 54.762, 2.000
0.0180, 0.2171,  2.879, 54.789, 2.000
0.0190, 0.2231,  3.039, 54.816, 2.000
0.0200, 0.2290,  3.199, 54.843, 2.000
0.0210, 0.2347,  3.359, 54.870, 2.000
0.0220, 0.2402,  3.519, 54.897, 2.000
0.0230, 0.2457,  3.679, 54.924, 2.000
0.0240, 0.2510,  3.839, 54.952, 2.000
```

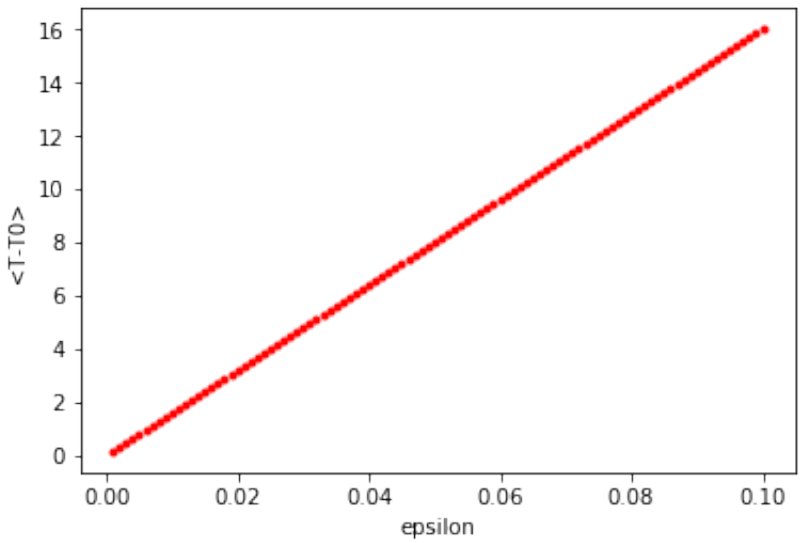
0.0250, 0.2563, 3.999, 54.979, 2.000
0.0260, 0.2614, 4.159, 55.006, 2.000
0.0270, 0.2665, 4.319, 55.033, 2.000
0.0280, 0.2715, 4.479, 55.060, 2.000
0.0290, 0.2764, 4.639, 55.087, 2.000
0.0300, 0.2812, 4.799, 55.115, 2.000
0.0310, 0.2859, 4.959, 55.142, 2.000
0.0320, 0.2906, 5.119, 55.169, 2.000
0.0330, 0.2952, 5.279, 55.196, 2.000
0.0340, 0.2997, 5.439, 55.223, 2.000
0.0350, 0.3041, 5.599, 55.250, 2.000
0.0360, 0.3085, 5.759, 55.277, 2.000
0.0370, 0.3128, 5.919, 55.305, 2.000
0.0380, 0.3171, 6.079, 55.332, 2.000
0.0390, 0.3213, 6.239, 55.359, 2.000
0.0400, 0.3254, 6.399, 55.386, 2.000
0.0410, 0.3295, 6.559, 55.413, 2.000
0.0420, 0.3337, 6.719, 55.440, 2.000
0.0430, 0.3377, 6.879, 55.467, 2.000
0.0440, 0.3417, 7.039, 55.495, 2.000
0.0450, 0.3457, 7.199, 55.522, 2.000
0.0460, 0.3496, 7.359, 55.549, 2.000
0.0470, 0.3535, 7.519, 55.576, 2.000
0.0480, 0.3574, 7.679, 55.603, 2.000
0.0490, 0.3612, 7.839, 55.630, 2.000
0.0500, 0.3649, 7.999, 55.658, 2.000
0.0510, 0.3686, 8.159, 55.685, 2.000
0.0520, 0.3723, 8.319, 55.712, 2.000
0.0530, 0.3760, 8.479, 55.739, 2.000
0.0540, 0.3796, 8.639, 55.766, 2.000
0.0550, 0.3831, 8.799, 55.793, 2.000
0.0560, 0.3867, 8.960, 55.820, 2.000
0.0570, 0.3902, 9.120, 55.848, 2.000
0.0580, 0.3937, 9.280, 55.875, 2.000
0.0590, 0.3972, 9.440, 55.902, 2.000
0.0600, 0.4007, 9.600, 55.929, 2.000
0.0610, 0.4042, 9.760, 55.956, 2.000
0.0620, 0.4076, 9.920, 55.983, 2.000
0.0630, 0.4110, 10.080, 56.010, 2.000
0.0640, 0.4144, 10.240, 56.038, 2.000
0.0650, 0.4177, 10.400, 56.065, 2.000
0.0660, 0.4210, 10.560, 56.092, 2.000
0.0670, 0.4243, 10.720, 56.119, 2.000
0.0680, 0.4275, 10.880, 56.146, 2.000
0.0690, 0.4308, 11.040, 56.173, 2.000
0.0700, 0.4340, 11.200, 56.201, 2.000
0.0710, 0.4371, 11.360, 56.228, 2.000
0.0720, 0.4403, 11.520, 56.255, 2.000
0.0730, 0.4434, 11.680, 56.282, 2.000
0.0740, 0.4465, 11.840, 56.309, 2.000
0.0750, 0.4497, 12.000, 56.336, 2.000
0.0760, 0.4528, 12.161, 56.363, 2.000
0.0770, 0.4559, 12.321, 56.391, 2.000

```
0.0780, 0.4590, 12.481, 56.418, 2.000
0.0790, 0.4621, 12.641, 56.445, 2.000
0.0800, 0.4652, 12.801, 56.472, 2.000
0.0810, 0.4682, 12.961, 56.499, 2.000
0.0820, 0.4712, 13.121, 56.526, 2.000
0.0830, 0.4742, 13.281, 56.553, 2.000
0.0840, 0.4771, 13.441, 56.581, 2.000
0.0850, 0.4801, 13.601, 56.608, 2.000
0.0860, 0.4830, 13.761, 56.635, 2.000
0.0870, 0.4859, 13.921, 56.662, 2.000
0.0880, 0.4887, 14.081, 56.689, 2.000
0.0890, 0.4916, 14.241, 56.716, 2.000
0.0900, 0.4944, 14.402, 56.744, 2.000
0.0910, 0.4973, 14.562, 56.771, 2.000
0.0920, 0.5002, 14.722, 56.798, 2.000
0.0930, 0.5031, 14.882, 56.825, 2.000
0.0940, 0.5060, 15.042, 56.852, 2.000
0.0950, 0.5088, 15.202, 56.879, 2.000
0.0960, 0.5116, 15.362, 56.906, 2.000
0.0970, 0.5144, 15.522, 56.934, 2.000
0.0980, 0.5172, 15.682, 56.961, 2.000
0.0990, 0.5200, 15.842, 56.988, 2.000
0.1000, 0.5227, 16.002, 57.015, 2.000
Logarithmic values slope=0.5073, intercept=0.2230
```









11.4.2 Broad range of parameter

In [22]:
```
% matplotlib inline
# Code to numerically integrate the actual dynamical system
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from scipy import stats
import math
#
T0 = 300           # ambient temperature K
Rb0 = 54.3         # ohms - Radio Shack 12V 25mA mini lamp part number 27
alpha = 3.126E-3   # linear temperature coefficient for resistance in K^-
beta = -9.291E-8   # quadratic temperature coefficient for resistance in
K = 1.610E-5       # W/K estimated conductive/convective transport coeffi
S = 4.607E-15      # W/K^4 estimated radiative transport coefficient
tau_fil = 0.25     # seconds estimated thermal time constant of filament

C_fil = tau_fil*(K+4*T0**3*S)
#print(C_fil,' J/K')

def Rb(T):
    Rb = Rb0*(1+alpha*(T-T0)+beta*(T-T0)*(T-T0))
    return Rb

R = 6.7E3
C = 10.0E-9
omega0 = 1/(R*C)
f0 = omega0/(2*math.pi)
period = 1./f0
print('Oscillator period = {0:9.6f} sec'.format(period))
```

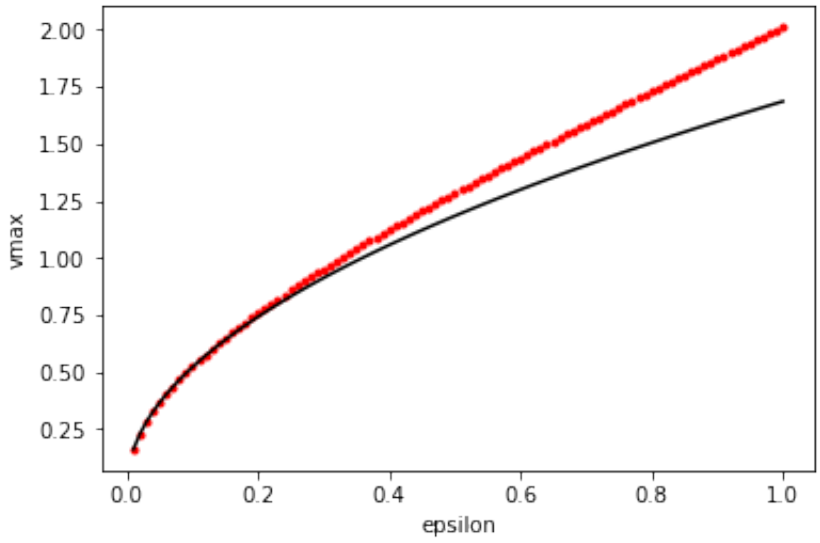
```
epsilon = 0.01
Ra = Rb0*(2.0+epsilon)

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt):
    ff0 = omega0*xx[1]
    ff1 = (Ra/Rb(xx[2])-2)*omega0*xx[1]-omega0*xx[0]
    ff2 = ((Rb(xx[2])/(Ra+Rb(xx[2])))**2)*xx[0]**2-K*(xx[2]-T0)-S*(xx[2]**
    return [ff0,ff1,ff2]

# Initial conditions
v_0 = 0.01
w_0 = 0.0
T_0 = T0

timestepfactor = 0.01
timerangefactor = 15000
timestep = timestepfactor*period
timerange = timerangefactor*period
Nstep = int(timerangefactor/timestepfactor)
print(Nstep,'time steps  ','{0:6.3f} sec total time'.format(timerange))

# time array
a_tt = np.arange(0,timerange,timestep)

# See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.
# Also https://nathantypanski.com/blog/2014-08-23-ode-solver-py.html

a_eps = []
a_vmax = []
a_TTavg = []
epsilon = 0.0
print('epsilon,vmax,<T-T0>,Rb(T),Ra/Rb(T)')

Ntenper = int(10*period/timestep)
for i in range(0,100):
    vmaxsum = 0.
    epsilon = epsilon + 0.01
    a_eps = np.append(a_eps,epsilon)
    Ra = Rb0*(2.0+epsilon)
    a_sol = integrate.odeint(dynsys,[v_0,w_0,T_0],a_tt)
    vv = a_sol[0:Nstep,0]
    TT = a_sol[0:Nstep,2]-T0
    jmax = 20
    for j in range(0,jmax): #average the maximum found over ten interval
        Nlimitupper = Nstep-j*Ntenper
        Nlimitlower = Nlimitupper - Ntenper
        vmax = np.amax(vv[Nlimitlower:Nlimitupper])
        vmaxsum = vmaxsum + vmax
    vmaxavg = vmaxsum/jmax
    a_vmax = np.append(a_vmax,vmaxavg)
    TTavg = np.sum(TT[Nstep-jmax*Ntenper:Nstep])/(jmax*Ntenper)
    a_TTavg = np.append(a_TTavg,TTavg)
    T_0 = T0 + TTavg # use longterm average temperature from this run as
```

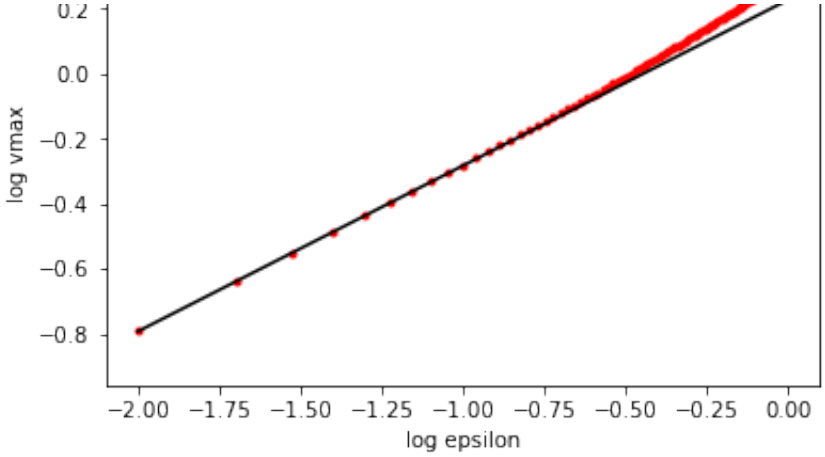
```
    print('{0:0.4f},{1:7.4f},{2:7.3f},{3:7.3f},{4:6.3f}'.format(epsilon,v

# Compute logaritms of epsilon and vmax
logeps = np.log10(a_eps)
logvmax = np.log10(a_vmax)

# Do a linear regression of the resulting logarithms for first ten values
altslope, altintercept, altr_value, altp_value, altstd_err = stats.linreg
print('First ten logarithmic values slope={0:0.4f}, intercept={1:0.4f}'.f

# Do a linear regression of the resulting logarithms of all data
slope, intercept, r_value, p_value, std_err = stats.linregress(logeps,log
print('Full range of ogarithmic values slope={0:0.4f}, intercept={1:0.4f}

#Computer theoretical values
logvmaxtheorval = altintercept + altslope*logeps
a_vmaxtheorval = np.power(10,logvmaxtheorval)

plt.figure(0)
plt.plot(a_eps,a_vmax,'r.',a_eps,a_vmaxtheorval,'k')
plt.xlabel('epsilon')
plt.ylabel('vmax')

plt.figure(1)
plt.plot(logeps,logvmax,'r.',logeps,logvmaxtheorval,'k')
plt.xlabel('log epsilon')
plt.ylabel('log vmax')
plt.axis('equal')

plt.figure(2)
plt.plot(a_eps,a_TTavg,'r.')
plt.xlabel('epsilon')
plt.ylabel('<T-T0>')

plt.show()
```

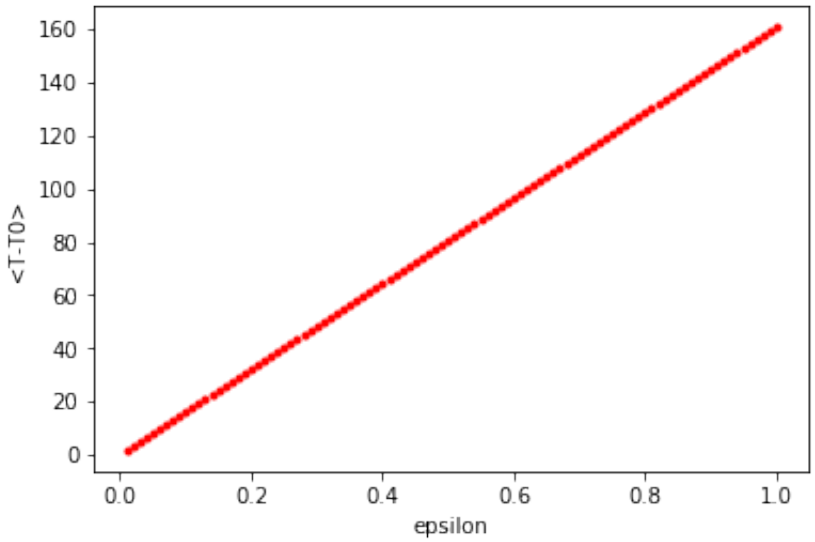
```
Oscillator period =  0.000421 sec
1500000 time steps    6.315 sec total time
epsilon,vmax,<T-T0>,Rb(T),Ra/Rb(T)
0.0100, 0.1615,  1.600, 54.572, 2.000
0.0200, 0.2290,  3.199, 54.843, 2.000
0.0300, 0.2812,  4.799, 55.115, 2.000
0.0400, 0.3255,  6.399, 55.386, 2.000
0.0500, 0.3648,  7.999, 55.658, 2.000
0.0600, 0.4008,  9.600, 55.929, 2.000
0.0700, 0.4339, 11.200, 56.201, 2.000
0.0800, 0.4651, 12.801, 56.472, 2.000
0.0900, 0.4946, 14.402, 56.744, 2.000
0.1000, 0.5226, 16.002, 57.015, 2.000
0.1100, 0.5496, 17.604, 57.287, 2.000
0.1200, 0.5753, 19.205, 57.558, 2.000
0.1300, 0.6005, 20.806, 57.830, 2.000
0.1400, 0.6247, 22.408, 58.101, 2.000
0.1500, 0.6482, 24.009, 58.373, 2.000
```

0.1600, 0.6712, 25.611, 58.644, 2.000
0.1700, 0.6932, 27.213, 58.916, 2.000
0.1800, 0.7155, 28.815, 59.187, 2.000
0.1900, 0.7367, 30.418, 59.459, 2.000
0.2000, 0.7578, 32.020, 59.730, 2.000
0.2100, 0.7784, 33.623, 60.002, 2.000
0.2200, 0.7985, 35.226, 60.273, 2.000
0.2300, 0.8186, 36.829, 60.545, 2.000
0.2400, 0.8379, 38.432, 60.816, 2.000
0.2500, 0.8576, 40.035, 61.088, 2.000
0.2600, 0.8763, 41.638, 61.359, 2.000
0.2700, 0.8955, 43.242, 61.631, 2.000
0.2800, 0.9138, 44.845, 61.902, 2.000
0.2900, 0.9324, 46.449, 62.174, 2.000
0.3000, 0.9504, 48.053, 62.445, 2.000
0.3100, 0.9685, 49.657, 62.717, 2.000
0.3200, 0.9862, 51.262, 62.988, 2.000
0.3300, 1.0039, 52.866, 63.260, 2.000
0.3400, 1.0213, 54.471, 63.531, 2.000
0.3500, 1.0386, 56.076, 63.803, 2.000
0.3600, 1.0558, 57.680, 64.074, 2.000
0.3700, 1.0728, 59.286, 64.346, 2.000
0.3800, 1.0897, 60.891, 64.617, 2.000
0.3900, 1.1064, 62.496, 64.889, 2.000
0.4000, 1.1231, 64.102, 65.160, 2.000
0.4100, 1.1396, 65.707, 65.432, 2.000
0.4200, 1.1561, 67.313, 65.703, 2.000
0.4300, 1.1723, 68.919, 65.975, 2.000
0.4400, 1.1886, 70.525, 66.246, 2.000
0.4500, 1.2047, 72.132, 66.518, 2.000
0.4600, 1.2207, 73.738, 66.789, 2.000
0.4700, 1.2367, 75.345, 67.061, 2.000
0.4800, 1.2524, 76.951, 67.332, 2.000
0.4900, 1.2683, 78.558, 67.604, 2.000
0.5000, 1.2837, 80.165, 67.875, 2.000
0.5100, 1.2997, 81.773, 68.147, 2.000
0.5200, 1.3147, 83.380, 68.418, 2.000
0.5300, 1.3307, 84.988, 68.690, 2.000
0.5400, 1.3456, 86.595, 68.961, 2.000
0.5500, 1.3614, 88.203, 69.233, 2.000
0.5600, 1.3764, 89.811, 69.504, 2.000
0.5700, 1.3919, 91.419, 69.776, 2.000
0.5800, 1.4070, 93.028, 70.047, 2.000
0.5900, 1.4220, 94.636, 70.319, 2.000
0.6000, 1.4373, 96.245, 70.590, 2.000
0.6100, 1.4518, 97.853, 70.862, 2.000
0.6200, 1.4673, 99.462, 71.133, 2.000
0.6300, 1.4816,101.071, 71.405, 2.000
0.6400, 1.4970,102.681, 71.676, 2.000
0.6500, 1.5116,104.290, 71.948, 2.000
0.6600, 1.5264,105.900, 72.219, 2.000
0.6700, 1.5413,107.509, 72.491, 2.000
0.6800, 1.5555,109.119, 72.762, 2.000

```
0.6900, 1.5707,110.729, 73.034, 2.000
0.7000, 1.5848,112.339, 73.305, 2.000
0.7100, 1.5997,113.950, 73.577, 2.000
0.7200, 1.6142,115.560, 73.848, 2.000
0.7300, 1.6284,117.171, 74.119, 2.000
0.7400, 1.6433,118.781, 74.391, 2.000
0.7500, 1.6572,120.392, 74.662, 2.000
0.7600, 1.6721,122.004, 74.934, 2.000
0.7700, 1.6864,123.615, 75.206, 2.000
0.7800, 1.7005,125.226, 75.477, 2.000
0.7900, 1.7152,126.838, 75.749, 2.000
0.8000, 1.7289,128.449, 76.020, 2.000
0.8100, 1.7436,130.061, 76.291, 2.000
0.8200, 1.7579,131.673, 76.563, 2.000
0.8300, 1.7716,133.286, 76.835, 2.000
0.8400, 1.7864,134.898, 77.106, 2.000
0.8500, 1.8002,136.510, 77.377, 2.000
0.8600, 1.8144,138.123, 77.649, 2.000
0.8700, 1.8288,139.736, 77.920, 2.000
0.8800, 1.8422,141.349, 78.192, 2.000
0.8900, 1.8570,142.962, 78.463, 2.000
0.9000, 1.8709,144.575, 78.735, 2.000
0.9100, 1.8845,146.189, 79.006, 2.000
0.9200, 1.8992,147.802, 79.278, 2.000
0.9300, 1.9128,149.416, 79.549, 2.000
0.9400, 1.9268,151.030, 79.821, 2.000
0.9500, 1.9412,152.644, 80.092, 2.000
0.9600, 1.9546,154.258, 80.364, 2.000
0.9700, 1.9688,155.872, 80.635, 2.000
0.9800, 1.9829,157.487, 80.907, 2.000
0.9900, 1.9961,159.102, 81.179, 2.000
1.0000, 2.0106,160.717, 81.450, 2.000
First ten logarithmic values slope=0.5101, intercept=0.2268
Full range of ogarithmic values slope=0.5628, intercept=0.2846
```







12 Frequency content of the signal

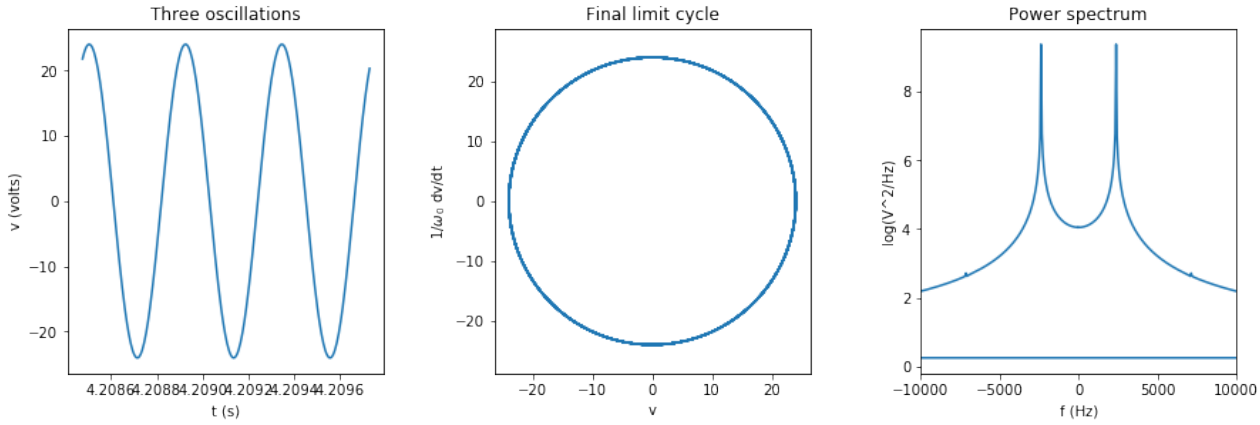
In [2]:
```
% matplotlib inline
# Code to compute the power spectrum of the Wien bridge oscillator outpu
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from scipy import stats
import math
#
T0 = 300           # ambient temperature K
Rb0 = 54.3         # ohms - Radio Shack 12V 25mA mini lamp part number 27
alpha = 3.126E-3   # linear temperature coefficient for resistance in K^-
beta = -9.291E-8   # quadratic temperature coefficient for resistance in
K = 1.610E-5       # W/K estimated conductive/convective transport coeffi
S = 4.607E-15      # W/K^4 estimated radiative transport coefficient
tau_fil = 0.25     # seconds estimated thermal time constant of filament

C_fil = tau_fil*(K+4*T0**3*S)
#print(C_fil,' J/K')

def Rb(T):
```

```
    Rb = Rb0*(1+alpha*(T-T0)+beta*(T-T0)*(T-T0))
    return Rb

R = 6.7E3
C = 10.0E-9
omega0 = 1/(R*C)
f0 = omega0/(2*math.pi)
period = 1./f0
print('Oscillator period = {0:9.6f} sec'.format(period))

epsilon = 0.01
Ra = Rb0*(2.0+epsilon)

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt):
    ff0 = omega0*xx[1]
    ff1 = (Ra/Rb(xx[2])-2)*omega0*xx[1]-omega0*xx[0]
    ff2 = ((Rb(xx[2])/(Ra+Rb(xx[2])))**2)*xx[0]**2-K*(xx[2]-T0)-S*(xx[2]*
    return [ff0,ff1,ff2]

# Initial conditions
v_0 = 0.01
w_0 = 0.0
T_0 = T0

timestepfactor = 0.01
timerangefactor = 10000
timestep = timestepfactor*period
timerange = timerangefactor*period
Nstep = int(timerangefactor/timestepfactor)
print(Nstep,'time steps  ','{0:6.3f} sec total time'.format(timerange))

# time array
a_tt = np.arange(0,timerange,timestep)

# See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.
# Also https://nathantypanski.com/blog/2014-08-23-ode-solver-py.html

a_eps = []
a_vmax = []
a_TTavg = []
epsilon = 9.99
print('epsilon,vmax,<T-T0>,Rb(T),Ra/Rb(T)')

T_0=T0+1600.7
Ntenper = int(10*period/timestep)
for i in range(0,1): # Eventually might embed the plotting in this loop
    vmaxsum = 0.
    epsilon = epsilon + 0.01
    a_eps = np.append(a_eps,epsilon)
    Ra = Rb0*(2.0+epsilon)
    a_sol = integrate.odeint(dynsys,[v_0,w_0,T_0],a_tt)
    vv = a_sol[0:Nstep,0]
    ww = a_sol[0:Nstep,1]
```

```
    TT = a_sol[0:Nstep,2]-T0
    jmax = 20
    for j in range(0,jmax): #average the maximum found over ten interval
        Nlimitupper = Nstep-j*Ntenper
        Nlimitlower = Nlimitupper - Ntenper
        vmax = np.amax(vv[Nlimitlower:Nlimitupper])
        vmaxsum = vmaxsum + vmax
    vmaxavg = vmaxsum/jmax
    a_vmax = np.append(a_vmax,vmaxavg)
    TTavg = np.sum(TT[Nstep-jmax*Ntenper:Nstep])/(jmax*Ntenper)
    a_TTavg = np.append(a_TTavg,TTavg)
    T_0 = T0 + TTavg # use longterm average temperature from this run as
    print('{0:0.4f},{1:7.4f},{2:7.3f},{3:7.3f},{4:6.3f}'.format(epsilon,v

#Now compute the spectrum
sp = np.fft.fft(vv[Nstep-4096:Nstep])
freq = np.fft.fftfreq(4096, d=timestep)

plt.figure(1)
plt.subplot(1,3,1)
Nthreeper = int(.3*Ntenper)
plt.plot(a_tt[Nstep-Nthreeper:Nstep],vv[Nstep-Nthreeper:Nstep])
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('Three oscillations')
#plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3} msec\ntau_fil = {
#            .format(epsilon,1000*period,1000*tau_fil))

plt.subplot(1,3,2)
plt.plot(vv[Nstep-Ntenper:Nstep],ww[Nstep-Ntenper:Nstep])
plt.xlabel('v')
plt.ylabel('1/$\omega_0$ dv/dt')
plt.title('Final limit cycle')
#plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3} msec\ntau_fil = {
#            .format(epsilon,1000*period,1000*tau_fil))
plt.axis('equal')

plt.subplot(1,3,3)
plt.plot(freq, np.log10(sp.real**2+sp.imag**2))
plt.xlim(-10000,10000)
plt.xlabel('f (Hz)')
plt.ylabel('log(V^2/Hz)')
plt.title('Power spectrum')
#plt.figtext(0.6,0.7,'eps = {0:4.2f}\nperiod = {1:5.3} msec\ntau_fil = {
#            .format(epsilon,1000*period,1000*tau_fil))

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=2.0, hspace=(

plt.show()
```

```
Oscillator period =  0.000421 sec
1000000 time steps    4.210 sec total time
epsilon,vmax,<T-T0>,Rb(T),Ra/Rb(T)
```

```
10.0000,24.0223,1683.750,325.800, 2.000
```



13 Approximate Model Near Onset and at Long Time

13.1 Developing the approximate model

Near onset we'll assume the temperature change is small enough that we only need a first-order correction in the filament resistance:

$$R_b(T) = R_{b0}[1 + \alpha(T - T_0)].$$

The dynamical equation including temperature dependence of the resistance R_b is then:

$$\frac{d^2v}{dt^2} - \left(\frac{R_a}{R_{b0}[1 + \alpha(T - T_0)]} - 2 \right) \omega_0 \frac{dv}{dt} + \omega_0^2 v = 0.$$

The self-heating is modest and thus the correction is assumed small compared to 1:

$$\frac{d^2v}{dt^2} - \left(\frac{R_a}{R_{b0}}[1 - \alpha(T - T_0)] - 2 \right) \omega_0 \frac{dv}{dt} + \omega_0^2 v \approx 0.$$

Rearranging

$$\frac{d^2v}{dt^2} - \left(\frac{R_a}{R_{b0}} - 2 - \frac{R_a}{R_{b0}} \alpha(T - T_0) \right) \omega_0 \frac{dv}{dt} + \omega_0^2 v \approx 0.$$

In the temperature term, we'll replave the resistor ratio with its onset value of 2:

$$\frac{d^2v}{dt^2} - \left(\frac{R_a}{R_{b0}} - 2 - 2\alpha(T - T_0) \right) \omega_0 \frac{dv}{dt} + \omega_0^2 v \approx 0.$$

We need to model how the temperature of the lamp filament behaves once the system has reached steady-state oscillation, after the squegging pulses have settled out. Recall (with only first order corrections to resistance with temperature):

$$\frac{d(T-T_0)}{dt} = \frac{1}{\mathbf{C}_f}\left[\frac{R_{b0}[1+\alpha(T-T_0)]}{\{R_a + R_{b0}[1+\alpha(T-T_0)]\}^2}v^2 - \mathrm{K}(T-T_0) - \mathrm{S}(T^4 - T_0^4)\right].$$

and

$$T^4 - T_0^4 = (T-T_0)^4 + 4T_0(T-T_0)^3 + 6T_0^2(T-T_0)^2 + 4T_0^3(T-T_0).$$

Again, if the temperature difference is small near onset we can just use

$$T^4 - T_0^4 \approx 4T_0^3(T-T_0).$$

Then

$$\frac{d(T-T_0)}{dt} = \frac{1}{\mathbf{C}_f}\left[\frac{R_{b0}[1+\alpha(T-T_0)]}{\{R_a + R_{b0}[1+\alpha(T-T_0)]\}^2}v^2 - \mathrm{K}(T-T_0) - 4\mathrm{S}T_0^3(T-T_0)\right].$$

$$\frac{d(T-T_0)}{dt} = \frac{1}{\mathbf{C}_f}\left[\frac{R_{b0}[1+\alpha(T-T_0)]}{\{R_a + R_{b0}[1+\alpha(T-T_0)]\}^2}v^2 - \mathrm{K}_{eff}(T-T_0)\right],$$

where

$$\mathrm{K}_{eff} \equiv \mathrm{K} + 4\mathrm{S}T_0^3.$$

Pull out the factor of R_{b0}:

$$\frac{d(T-T_0)}{dt} = \frac{1}{\mathbf{C}_f}\left[\frac{[1+\alpha(T-T_0)]}{R_{b0}\left\{\frac{R_a}{R_{b0}} + [1+\alpha(T-T_0)]\right\}^2}v^2 - \mathrm{K}_{eff}(T-T_0)\right].$$

Defining $\epsilon \equiv \frac{R_a}{R_{b0}} - 2$, and anticipating that v^2 and $T-T_0$ are going to be $O(\epsilon)$, we rearrange and keep leading-order terms to obtain:

$$\mathbf{C}_f\frac{d(T-T_0)}{dt} + \mathrm{K}_{eff}(T-T_0) = \frac{v^2}{R_{b0}(2+1)^2}.$$

Now assume close to onset $v \approx V\cos\omega t$ and
$T - T_0 \approx < T - T_0 > + T_c\cos 2\omega t + T_s\sin 2\omega t$ where $< T - T_0 >$ is the time-independent average temperature deviation from ambient after dynamical equilibrium is reached to give a steady amplitude of oscillation.

Then

$$\mathrm{K}_{eff} < T - T_0 > + (\mathbf{C}_f\omega_0 T_s + \mathrm{K}_{eff}T_c)\cos 2\omega t + (-\mathbf{C}_f\omega_0 T_c + \mathrm{K}_{eff}T_s)\sin 2\omega t = \frac{1}{9R_{b0}}.$$

where we have used

$$\cos^2\omega t = \frac{1}{2} + \frac{1}{2}\cos 2\omega t.$$

Requring this to be true at all times gives a "harmonic balance":

$$\mathrm{K}_{eff} < T - T_0 > = \frac{1}{9R_{b0}} \frac{1}{2} V^2.$$
$$\mathbf{C}_f \omega_0 T_s + \mathrm{K}_{eff} T_c) = \frac{1}{9R_{b0}} \frac{1}{2} V^2.$$
$$(-\mathbf{C}_f \omega_0 T_c + \mathrm{K}_{eff} T_s) = 0.$$

From this we get

$$< T - T_0 > = \frac{1}{9R_{b0}\mathrm{K}_{eff}} \frac{1}{2} V^2.$$
$$T_c = \frac{1}{9R_{b0}\mathrm{K}_{eff}} \frac{1}{2} V^2 \frac{1}{1 + \left(\frac{\omega_0 \mathbf{C}_f}{\mathrm{K}_{eff}}\right)^2}$$
$$. T_s = \frac{1}{9R_{b0}\mathrm{K}_{eff}} \frac{1}{2} V^2 \frac{\frac{\omega_0 \mathbf{C}_f}{\mathrm{K}_{eff}}}{1 + \left(\frac{\omega_0 \mathbf{C}_f}{\mathrm{K}_{eff}}\right)^2}$$

.

So

$$T - T_0 \approx \frac{1}{9R_{b0}\mathrm{K}_{eff}} \frac{1}{2} V^2 \left[1 + \frac{1}{1 + \left(\frac{\omega_0 \mathbf{C}_f}{\mathrm{K}_{eff}}\right)^2} \cos 2\omega t + \frac{\frac{\omega_0 \mathbf{C}_f}{\mathrm{K}_{eff}}}{1 + \left(\frac{\omega_0 \mathbf{C}_f}{\mathrm{K}_{eff}}\right)^2} \sin 2\omega t \right]$$

Now we define the filament time constant

$$\tau_{\mathrm{fil}} \equiv \frac{\mathbf{C}_f}{\mathrm{K}_{eff}},$$

and recognize that the period τ_0 of the oscillator is given by

$$\tau_0 = \frac{2\pi}{\omega_0},$$

so that we can write:

$$T - T_0 \approx \frac{1}{9R_{b0}\mathrm{K}_{eff}} \frac{1}{2} V^2 \left[1 + \frac{1}{1 + \left(2\pi \frac{\tau_{\mathrm{fil}}}{\tau_0}\right)^2} \cos 2\omega t + \frac{2\pi \frac{\tau_{\mathrm{fil}}}{\tau_0}}{1 + \left(2\pi \frac{\tau_{\mathrm{fil}}}{\tau_0}\right)^2} \sin 2\omega t \right]$$

Now for the system we have simulated as well as examined experimentally, $\tau_0 \approx 0.4$ msec and $\tau_{\mathrm{fil}} \approx 250$ msec so that

$$2\pi \frac{\tau_{\mathrm{fil}}}{\tau_0} \approx 4000.$$

Thus

$$T - T_0 \approx \frac{1}{9R_{b0}\mathrm{K}_{eff}}\frac{1}{2}V^2\left[1 + 6.3\times10^{-8}\cos 2\omega t + 2.5\times10^{-4}\sin 2\omega t\right]$$

So once steady amplitude oscillation is reached, the temperature makes tiny oscillations at twice the frequency around an average value. This means that to a very good approximation, we can subtitute the average value into the dynamical equation

$$\frac{d^2v}{dt^2} - \left(\frac{R_a}{R_{b0}} - 2 - 2\alpha\frac{1}{9\mathrm{K}R_{b0}}\frac{1}{2}V^2\right)\omega_0\frac{dv}{dt} + \omega_0^2 v = 0.$$

Now here is an important step: we can relate the oscillation amplitude V to the dynamical variables as follows:

$$V^2 = V^2\cos^2\omega_0 t + V^2\sin^2\omega_0 t.$$

and then since we have assumed that $v \approx V\cos\omega_0 t$ and we have defined $w \equiv \frac{1}{\omega_0}\frac{dv}{dt}$ so that $w \approx -V\sin\omega_0 t$, we have

$$V^2 = v^2 + w^2.$$

Thus at long time the dynamical equation becomes:

$$\frac{d^2v}{dt^2} - \left(\frac{R_a}{R_{b0}} - 2 - \frac{\alpha}{9\mathrm{K}_{\mathrm{eff}}R_{b0}}(v^2 + w^2)\right)\omega_0\frac{dv}{dt} + \omega_0^2 v = 0.$$

Again, recognize that we defined

$$\epsilon \equiv \frac{R_a}{R_{b0}} - 2.$$

and now define

$$v_* \equiv \sqrt{\frac{9\mathrm{K}_{\mathrm{eff}}R_{b0}}{\alpha}}.$$

Notice how these parameters indeed set a voltage scale, as might have been achieved through dimensional analysis. $\mathrm{K}_{\mathrm{eff}} \equiv \mathrm{K} + 4\mathrm{S}T_0^3$ has units of watts/kelvin = volts^2/(ohm kelvin). R_{b0} has units of ohms. So $\mathrm{K}_{\mathrm{eff}}R_{b0}$ has units of volts^2/kelvin. Dividing by α with units 1/kelvin gives a quantity that simply has units of volts^2, and then taking the square root gives volts. In the code below, using experimental parameter values for $R_{b0}, \mathrm{K}, \mathrm{S}, T_0$, and α , we obtain a value $v_* = 1.611$ volts.

We can now write:

$$\frac{d^2v}{dt^2} - \left(\epsilon - \frac{1}{v_*^2}(v^2 + w^2)\right)\omega_0\frac{dv}{dt} + \omega_0^2 v = 0.$$

or in terms of v alone:

$$\frac{d^2v}{dt^2} - \left(\epsilon - \frac{1}{v_*^2}\left[v^2 + \left(\frac{1}{\omega_0}\frac{dv}{dt}\right)^2\right]\right)\omega_0\frac{dv}{dt} + \omega_0^2 v = 0.$$

This translates to a two-dimensional dynamical system:

$$\frac{dv}{dt} = \omega_0 w.$$
$$\frac{dw}{dt} = -\omega_0 v + \left(\epsilon - \frac{1}{v_*^2}(v^2 + w^2)\right)\omega_0 w.$$

We will call this the Rayleigh-van der Pol model because it combines both the Rayleigh and the van der Pol nonlinear terms in equal amounts. This reflects the fact that the nonlinearity comes from the average power dissipated in the part of the circuit controlling the gain. We might also call it the modified Hopf model since it has a structure similar to the Hopf dynamical system. Indeed, we can try the same change of variables:

$$v = r\cos\theta.$$
$$w = r\sin\theta.$$

The equivalent system is

$$\frac{dr}{dt} = \left(\epsilon - \frac{r^2}{v_*^2}\right)\omega_0 r\sin^2\theta.$$
$$\frac{d\theta}{dt} = -\omega_0 + \left(\epsilon - \frac{r^2}{v_*^2}\right)\omega_0\cos\theta\sin\theta.$$

A constant-amplitude solution is

$$r = v_*\sqrt{\epsilon}.$$
$$\theta = -\omega_0 t.$$

This is a remarkably straightforward result similar to that obtained for the basic Hopf model. It predicts a pure sinusoidal oscillation at all values of ϵ, that is, one that contains no harmonics - unlike the van der Pol model. Moreover the limit cycle is a circle of fixed radius proportional to $\sqrt{\epsilon}$ in (v, w) phase space. A key point to remember, however, is that the initial transient does *not* represent the actual dynamics since it ignores the initial large temperature changes and associated "squegging" of the voltage oscillations. For this, the full three-dimensional model must be used that was explored numerically above.

Note also:

$$T - T_0 = T_*\epsilon\,,$$

where

$$T_* \equiv \frac{1}{2\alpha}.$$

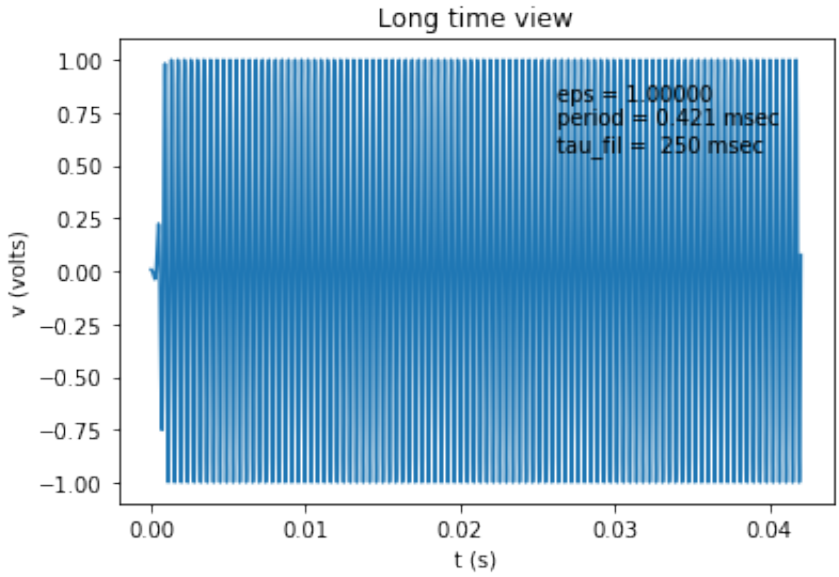
13.2 Numerical integration of the Rayleigh - van der Pol model

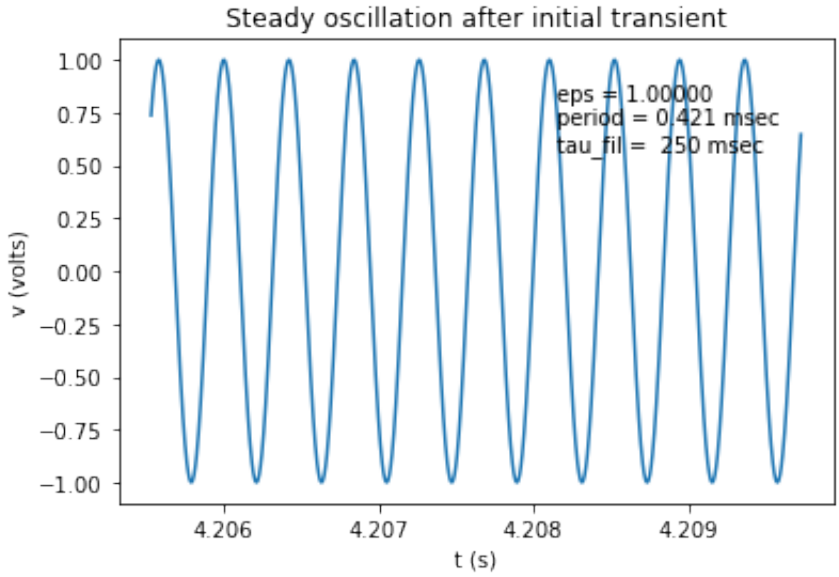
In [3]:
```
%matplotlib inline
# Code to numerically integrate the modified Hopf oscillator model of th
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math
#
T0 = 300           # ambient temperature K
Rb0 = 54.3         # cold resistance ohms - Radio Shack 12V 25mA mini lam
alpha = 3.126E-3   # linear temperature coefficient for resistance in K^-
beta = -9.291E-8   # quadratic temperature coefficient for resistance in
K = 1.610E-5       # W/K estimated conductive/convective transport coeffi
S = 4.607E-15      # W/K^4 estimated radiative transport coefficient
tau_fil = 0.25      # seconds estimated thermal time constant of filament

Keff = K + 4*S*T0**3
vstar = math.sqrt(9*Keff*Rb0/alpha)

R = 6.7E3
C = 10.0E-9
omega0 = 1/(R*C)
f0 = omega0/(2*math.pi)
period = 1./f0
print('Oscillator period = {0:9.6f} sec'.format(period))

Keff = K + 4*T0**3*S # effective linear heat transport coefficient

#epsilon = 0.02
epsilon = 1.0
Ra = Rb0*(2.0+epsilon)

vstar = math.sqrt(9*Keff*Rb0/alpha)
print('vstar = {0:5.3f} volts'.format(vstar))
if epsilon > 0:
    vpredicted = vstar*math.sqrt(epsilon)
else:
    vpredicted = 0.
print('Vpredicted = {0:5.3f} volts'.format(vpredicted))
T_T0predicted = 0.5*epsilon/alpha
print('T_T0predicted = {0:5.3f} K '.format(T_T0predicted))

timestepfactor = 0.01
timerangefactor = 10000 #15000
timestep = timestepfactor*period
timerange = timerangefactor*period
Nstep = int(timerangefactor/timestepfactor)
print(Nstep,'time steps  ','{0:6.3f} sec total time'.format(timerange) )
```

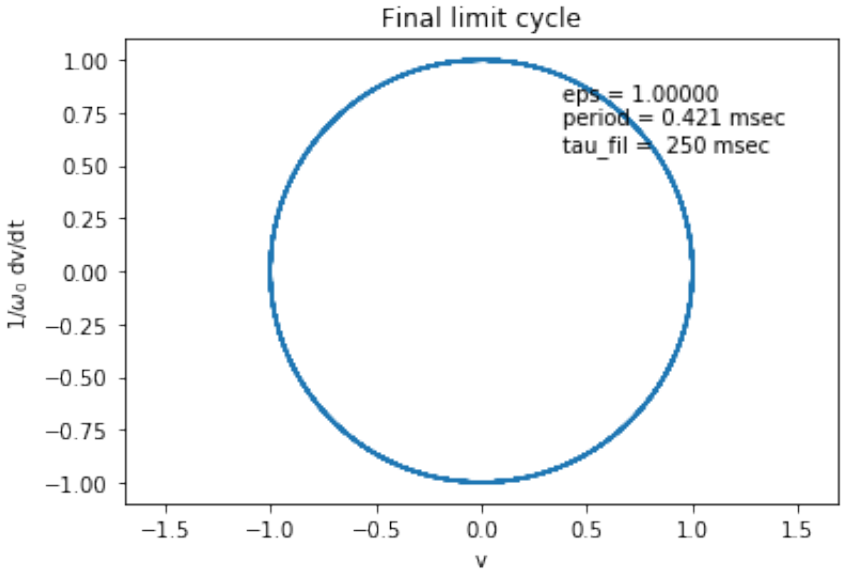
```
# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt):
    ff0 = omega0*xx[1]
    ff1 = (epsilon-(xx[0]*xx[0]+xx[1]*xx[1])/vstar**2)*omega0*xx[1]-omega
#    ff1 = (Ra/Rb0-2-alpha*(xx[0]*xx[0]+xx[1]*xx[1])/(9*Keff*Rb0))*omega
#    ff1 = (Ra/Rb0-2-4*alpha*(xx[0]*xx[0])/(9*K*Rb0))*omega0*xx[1]-omega
    return [ff0,ff1]

a_tt = np.arange(0,timerange,timestep) # time array

# Initial conditions
v_0 = 0.01
w_0 = 0.0
T_0 = T0

# Integrate the dynamical system numerically.
# See https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.
# Also https://nathantypanski.com/blog/2014-08-23-ode-solver-py.html
a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt)
vv = a_sol[0:Nstep,0]/vstar #normalize the voltage
ww = a_sol[0:Nstep,1]/vstar

Ntenper = int(10*period/timestep)

vmax = np.amax(vv[Nstep-Ntenper:Nstep])
print('Vamplitude = {0:5.3f} volts'.format(vmax))

#Now compute the spectrum
#sp = np.fft.fft(vv[Nstep-4096:Nstep])
#freq = np.fft.fftfreq(4096, d=timestep)
sp = np.fft.fft(vv)
freq = np.fft.fftfreq(Nstep, d=timestep)/omega0

plt.figure(1)
plt.plot(a_tt[0:10*Ntenper],vv[0:10*Ntenper])
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('Long time view')
plt.figtext(0.6,0.7,'eps = {0:5.5f}\nperiod = {1:5.3} msec\ntau_fil = {2
            .format(epsilon,1000*period,1000*tau_fil))

plt.figure(2)
plt.plot(a_tt[Nstep-Ntenper:Nstep],vv[Nstep-Ntenper:Nstep])
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('Steady oscillation after initial transient')
plt.figtext(0.6,0.7,'eps = {0:5.5f}\nperiod = {1:5.3} msec\ntau_fil = {2
            .format(epsilon,1000*period,1000*tau_fil))

plt.figure(3)
plt.plot(vv[Nstep-Ntenper:Nstep],ww[Nstep-Ntenper:Nstep])
plt.xlabel('v')
plt.ylabel('1/$\omega_0$ dv/dt')
```

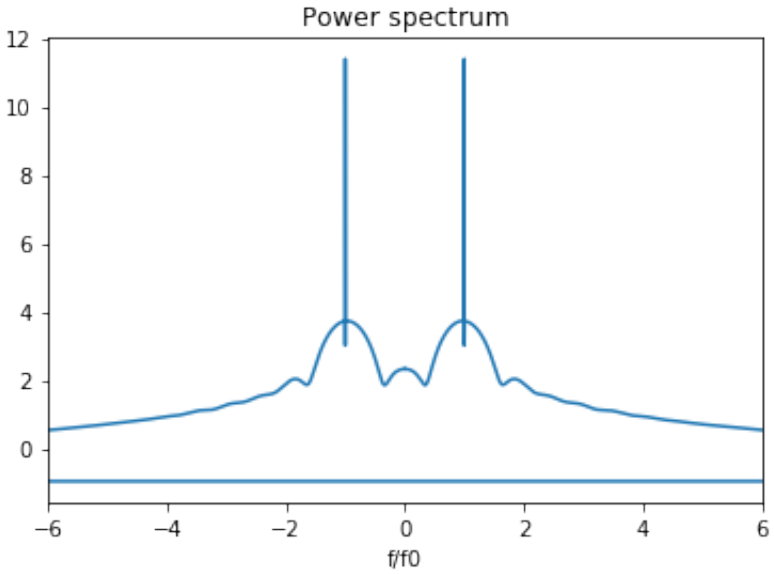
```
plt.title('Final limit cycle')
plt.figtext(0.6,0.7,'eps = {0:5.5f}\nperiod = {1:5.3} msec\ntau_fil = {2
            .format(epsilon,1000*period,1000*tau_fil))
plt.axis('equal')

plt.figure(4)
plt.plot(2*math.pi*freq, np.log10(sp.real**2+sp.imag**2))
plt.xlim(-6,6)
plt.xlabel('f/f0')
#plt.ylabel('V^2/Hz')
plt.title('Power spectrum')

plt.show()
```

```
Oscillator period =  0.000421 sec
vstar = 1.611 volts
Vpredicted = 1.611 volts
T_T0predicted = 159.949 K
1000000 time steps    4.210 sec total time
Vamplitude = 1.000 volts
```









14 Modeling noise

When noise is intected into the system, the output response is expected to fluctuate with a variance that is large near the bifurcation point. Ongoing work is demonstrating this behavior, reminiscent of the divergence of fluctuations in the thermodynamic behavior of fluids near their critical point.

[This section is still under development. 24 April 2018.]

14.1 Rayleigh van der Pol model with noise

Here we use an Euler-Maruyama method https://en.wikipedia.org/wiki/Euler–Maruyama_method (https://en.wikipedia.org/wiki/Euler%E2%80%93Maruyama_method)

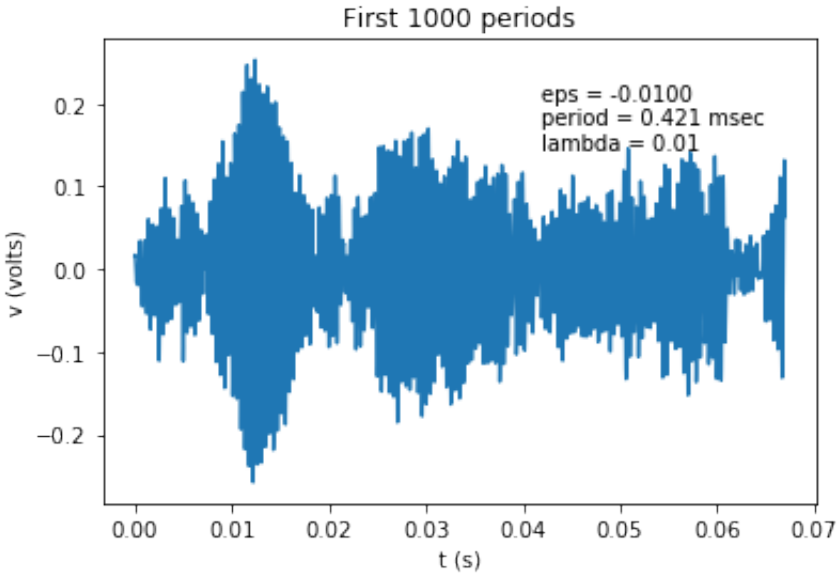
In [1]:
```
% matplotlib inline
```

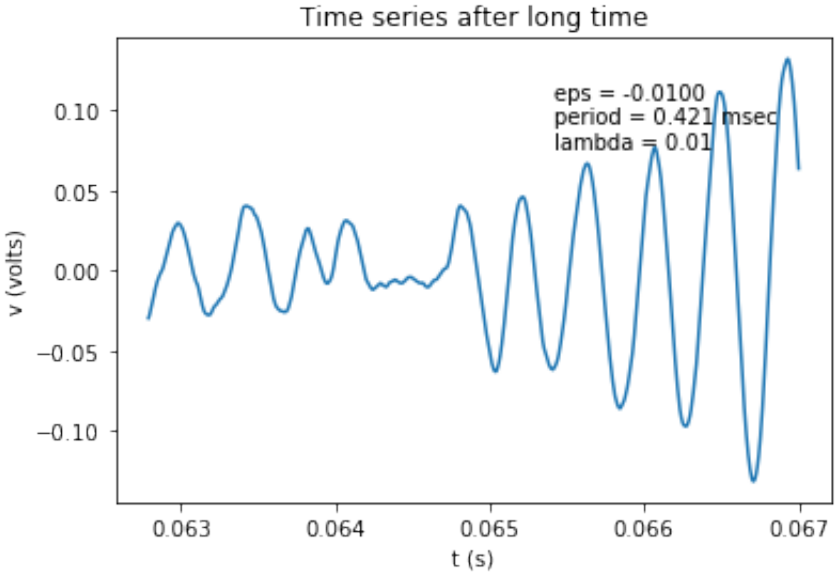
```
%matplotlib inline
# Code to numerically integrate the Rayleigh - van der Pol
# (modified Hopf) oscillator model of the Wien bridge oscillator
# including additive noise

import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
import math,sys,random
#
T0 = 300           # ambient temperature K
Rb0 = 54.3         # cold resistance ohms - Radio Shack 12V 25mA mini lamp
alpha = 3.126E-3   # linear temperature coefficient for resistance in K^-1
beta = -9.291E-8   # quadratic temperature coefficient for resistance in K^-2
K = 1.610E-5       # W/K estimated conductive/convective transport coefficient
S = 4.607E-15      # W/K^4 estimated radiative transport coefficient
tau_fil = 0.25      # seconds estimated thermal time constant of filament

R = 6.7E3
C = 10.0E-9
omega0 = 1/(R*C)
f0 = omega0/(2*math.pi)
period = 1./f0
print('Oscillator period = {0:9.6f} sec'.format(period))

Keff = K + 4*T0**3*S # effective linear heat transport coefficient

epsilon = -0.01
Ra = Rb0*(2.0+epsilon)
print('epsilon = ',epsilon)

lamb = 0.01
#lamb = 0.01 # noise level
print('Noise amplitude lamb = ',lamb)

vstar = math.sqrt(9*Keff*Rb0/alpha)
print('vstar = {0:5.3f} volts'.format(vstar))
if epsilon > 0:
    vpredicted = vstar*math.sqrt(epsilon)
else:
    vpredicted = 0.
print('Vpredicted = {0:5.3f} volts'.format(vpredicted))
T_T0predicted = 0.5*epsilon/alpha
print('T_T0predicted = {0:5.3f} K '.format(T_T0predicted))

timestepfactor = 0.001
timerangefactor = 1000.
#timestep = timestepfactor*period
#timerange = timerangefactor*period
timestep = timestepfactor #*2*math.pi #period
timerange = timerangefactor #*2*math.pi #period
Nstep = int(timerangefactor/timestepfactor)
print(Nstep,'time steps  ','{0:6.3f} sec total time'.format(timerange/ome

# define the dynamical system to be used by the numerical integrator
```

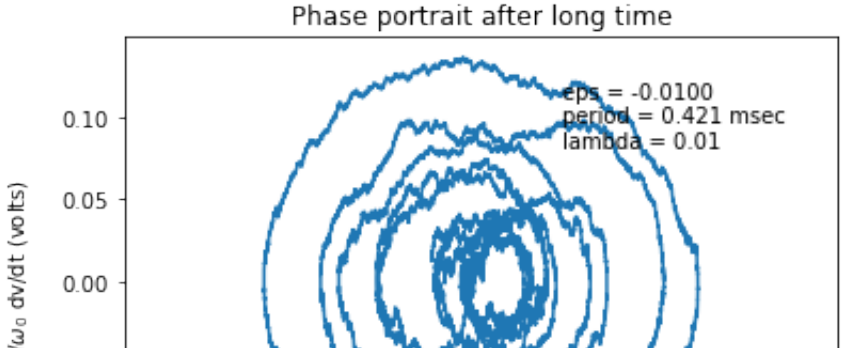
```
# define the dynamical system to be used by the numerical integrator
#def dynsys(xx,tt):
#    ff0 = omega0*xx[1]
#    ff1 = (epsilon-(xx[0]*xx[0]+xx[1]*xx[1])/vstar**2)*omega0*xx[1]-omeg
#    ff1 = (epsilon-(xx[0]*xx[0])/vstar**2)*omega0*xx[1]-omega0*xx[0]
#    return [ff0,ff1]

# define the dynamical system to be used by the numerical integrator
def dynsys(xx,tt):
    ff0 = xx[1]
    ff1 = (epsilon-(xx[0]*xx[0]+xx[1]*xx[1]))*xx[1]-xx[0]
#    ff1 = (epsilon-(xx[0]*xx[0]))*xx[1]-xx[0] # van der Pol version
    return [ff0,ff1]


a_tt = np.arange(0,timerange,timestep) # time array
a_sol = np.zeros((Nstep+1,2))

# Initial conditions
v_0 = 0.01
w_0 = 0.0
T_0 = T0
a_sol[0] = [v_0,w_0]
print('Initial conditions ',a_sol[0],'vstar')


# implement the Euler-Maruyama method
sigma = math.sqrt(timestep) # this root variance is a key part of the Ito
delta_sol=[0,0]
for i in range(0,Nstep):
    delta_sol=np.multiply(dynsys(a_sol[i],a_tt[i]),timestep)
    if lamb != 0.0: #Add noise
        #delta_sol[0] += lamb * random.gauss(mu = 0.0, sigma = sigma)
        delta_sol[1] += lamb * random.gauss(mu = 0.0, sigma = sigma)
    a_sol[i+1]=np.add(a_sol[i],delta_sol)
    if i % 1000 == 0: print(i,end='\r')

#print('delta_sol=',delta_sol)
#sys.exit("Stopped exucation after integration loop")

#Alternative using built-in integrator for noise free case
#a_sol = integrate.odeint(dynsys,[v_0,w_0],a_tt)

vv = a_sol[0:Nstep,0]
ww = a_sol[0:Nstep,1]

a_save = a_sol[Nstep-1]

vv = a_sol[0:Nstep,0]
ww = a_sol[0:Nstep,1]

#Ntenper = int(10*period/timestep)
Ntenper = int(10*2*math.pi/timestep)

vmax = np.amax(vv[Nstep-Ntenper:Nstep])
```

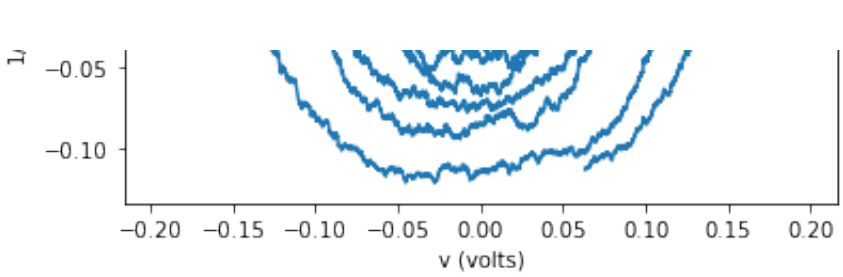
```
vmax = np.amax(vv[Nstep-Ntenper:Nstep])
print('Vamplitude = {0:5.3f} volts'.format(vmax))

#Now compute the spectrum
#sp = np.fft.fft(vv[Nstep-4096:Nstep])
#freq = np.fft.fftfreq(4096, d=timestep)
sp = np.fft.fft(vv)
freq = np.fft.fftfreq(Nstep, d=timestep)
#freq = np.fft.fftfreq(Nstep, d=timestep)/omega0

plt.figure(1)
#plt.plot(a_tt[0:100*Ntenper],vv[0:100*Ntenper])
#plt.xlabel('$\omega_0$t')
#plt.ylabel('v/vstar')
plt.plot(a_tt[0:100*Ntenper]/omega0,vv[0:100*Ntenper]*vstar)
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('First 1000 periods')
plt.figtext(0.6,0.7,'eps = {0:7.4f}\nperiod = {1:5.3} msec\nlambda = {2:4
            .format(epsilon,1000*period,lamb))

plt.figure(2)
#plt.plot(a_tt[Nstep-Ntenper:Nstep],vv[Nstep-Ntenper:Nstep])
#plt.xlabel('$\omega_0$t')
#plt.ylabel('v/vstar')
plt.plot(a_tt[Nstep-Ntenper:Nstep]/omega0,vv[Nstep-Ntenper:Nstep]*vstar)
plt.xlabel('t (s)')
plt.ylabel('v (volts)')
plt.title('Time series after long time')
plt.figtext(0.6,0.7,'eps = {0:7.4f}\nperiod = {1:5.3} msec\nlambda = {2:4
            .format(epsilon,1000*period,lamb))

plt.figure(3)
#plt.plot(vv[Nstep-Ntenper:Nstep],ww[Nstep-Ntenper:Nstep])
#plt.xlabel('v/vstar')
#plt.ylabel('1/($\omega_0 vstar)$ dv/dt')
plt.plot(vv[Nstep-Ntenper:Nstep]*vstar,ww[Nstep-Ntenper:Nstep]*vstar)
plt.xlabel('v (volts)')
plt.ylabel('1/$\omega_0$ dv/dt (volts)')
plt.title('Phase portrait after long time')
plt.figtext(0.6,0.7,'eps = {0:7.4f}\nperiod = {1:5.3} msec\nlambda = {2:4
            .format(epsilon,1000*period,lamb))
plt.axis('equal')

plt.figure(4)
plt.plot(2*math.pi*freq, np.log10(sp.real**2+sp.imag**2))
plt.xlim(-6,6)
plt.xlabel('f/f0')
#plt.ylabel('V^2/Hz')
plt.title('Power spectrum')
plt.figtext(0.6,0.7,'eps = {0:7.4f}\nfreq f0 = {1:6.2f} kHz\nlambda = {2
            .format(epsilon,0.001/period,lamb))

plt.show()
```

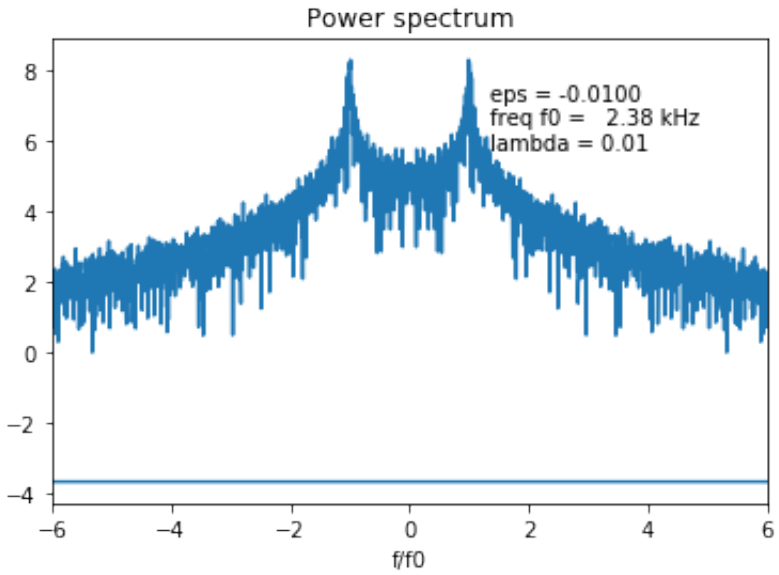
```
Oscillator period =  0.000421 sec
epsilon =  -0.01
Noise amplitude lamb =  0.01
vstar = 1.611 volts
Vpredicted = 0.000 volts
T_T0predicted = -1.599 K
1000000 time steps    0.067 sec total time
Initial conditions  [ 0.01  0.  ] vstar
Vamplitude = 0.082 volts
```











REF References

REF.1 Wien Bridge Oscillators

Casaleiro, J., L. B. Oliveira, & A. C. Pinto, "Van der Pol approxximation applied to Wien oscillators," Procedia Technology 17 (2014) 335-342.

Enns, R. H. and McGuire, G. C., "van der Pol limit cycle", in *Nonlinear Physics with Mathematica for Scientists and Engineers*, (Birkhauer, 2001) pp. 587-590.

Hayashi, Y & T. Nakagawa, "Transient behaviors in the Wien Bridge oscillator: a laboratory experiment for the undergraduate student," Am. J. Phys. v.52 (1984) pp. 1021-2014.

Horn, P. M., T. Carruthers, and M. T. Long, "Threshold instabilities in nonlinear self-excited oscillators", Phys. Rev. A, v. 14, no. 2 (1976) 833-839.

Lerner, L., "The dynamics of a stabilised Wien bridge oscillator," Eur. J. Phys v. 37 (2016) 065807 (22p).

Kriegsman, G. A., "The rapid bifurcation of the Wien Bridge Oscillator", IEEE Trans. Cicuits & Systems, v. CAS-34, no. 9 (1987) pp. 1093-1096.

Oliver, B. M., "The effecct of μ-circuit non-linearity on the amplitiude stability of RC oscillators," Hewlett-Packard Journal, v. 11, no. 8-10 (1960) pp 1-8.

REF.2 Tungsten filaments

Carla, M., Stefan-Boltzmann law for the tungsten filament of a light-bulb: revisiting the experiment," Am. J. Phys. v. 81 (2013) pp. 512-517.

Desai, P. D., T. K. CHi, H. M. James, & C. Y. Ho, "Electrical resistivity of selected elements," J. Phys. Chem. Ref. Data, v. 13, No. 4 (1984) pp. 1069-1096.

Edmonds, I. R., "Stephan-Boltzmann Law in the labtoratory,", Am. J. Phys. v. 36 (1968) pp. 845-846.

Heddle, D. W. O. "A simple test of Stefan's law," Phys. Ed., v. 10 (1975) pp. 392-393.

MacIsaac, D., G. Kanner & G. Anderson, "Basic physics of the incandescent lamp (lightbulb)," Phys. Teach. v. 37 (1999) pp. 520-525.

Prasad, B. S. N. & R. Mascarenhas, "A laboratory experiment of the application of Stefan's law to tungsten filament electric lamps,", Am. J. Phys. v. 46 (1978) pp. 420-423.

Ribeiro, C. I., "Blackbody radiation from an incandescent lamp,", Am. J. Phys. v. 52 (2014) pp. 371-372.

Wray, E. M. "A simple test of Stefan's law (student experiment)," Phys. Educ. v. 10 (1975) pp. 25-27.

REF.3 Books

Abarbanel, H., *Predicting the Future: Completing Models of Observed Complex Systems,* (Springer, 2013). Section 2.2 "The Colpitts Oscillator."

Odyniec, M. (ed.), *RF and Microwave Oscillator Design* (Artech House, 2002).

Rybin, Yu. K., *Electronic Devices for Analog Signal Processing* (Springer, 2012), Ch. 2 "Sine Wave Oscillators".

Senani, R., D. R. Bhaskar, V. K. Singh, and R. K. Sharma, *Sinusoidal Oscillators and Waveform Generators using Modern Electronic Circuit Building Blocks,* (Springer, 2016).