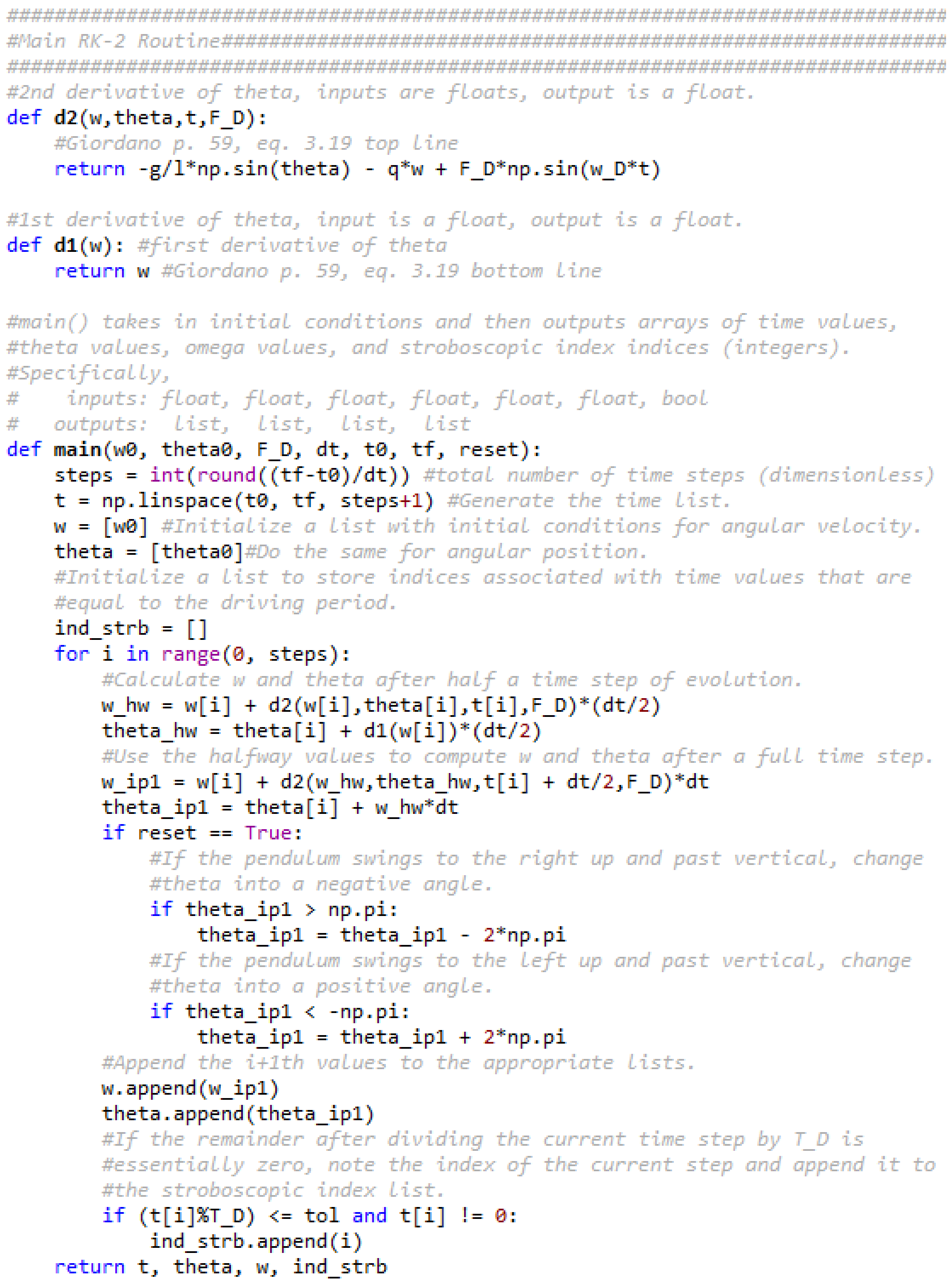
PHYS486-HW3: Damped Driven Pendulum

Nicholas Cemenenkoff

May 4, 2018

A



```
##############################################################################
#Main RK-2 Routine############################################################
##############################################################################
#2nd derivative of theta, inputs are floats, output is a float.
def d2(w,theta,t,F_D):
    #Giordano p. 59, eq. 3.19 top line
    return -g/l*np.sin(theta) - q*w + F_D*np.sin(w_D*t)

#1st derivative of theta, input is a float, output is a float.
def d1(w): #first derivative of theta
    return w #Giordano p. 59, eq. 3.19 bottom line

#main() takes in initial conditions and then outputs arrays of time values,
#theta values, omega values, and stroboscopic index indices (integers).
#Specifically,
#    inputs: float, float, float, float, float, float, bool
#   outputs:  list,  list,  list,  list
def main(w0, theta0, F_D, dt, t0, tf, reset):
    steps = int(round((tf-t0)/dt)) #total number of time steps (dimensionless)
    t = np.linspace(t0, tf, steps+1) #Generate the time list.
    w = [w0] #Initialize a list with initial conditions for angular velocity.
    theta = [theta0]#Do the same for angular position.
    #Initialize a list to store indices associated with time values that are
    #equal to the driving period.
    ind_strb = []
    for i in range(0, steps):
        #Calculate w and theta after half a time step of evolution.
        w_hw = w[i] + d2(w[i],theta[i],t[i],F_D)*(dt/2)
        theta_hw = theta[i] + d1(w[i])*(dt/2)
        #Use the halfway values to compute w and theta after a full time step.
        w_ip1 = w[i] + d2(w_hw,theta_hw,t[i] + dt/2,F_D)*dt
        theta_ip1 = theta[i] + w_hw*dt
        if reset == True:
            #If the pendulum swings to the right up and past vertical, change
            #theta into a negative angle.
            if theta_ip1 > np.pi:
                theta_ip1 = theta_ip1 - 2*np.pi
            #If the pendulum swings to the left up and past vertical, change
            #theta into a positive angle.
            if theta_ip1 < -np.pi:
                theta_ip1 = theta_ip1 + 2*np.pi
        #Append the i+1th values to the appropriate lists.
        w.append(w_ip1)
        theta.append(theta_ip1)
        #If the remainder after dividing the current time step by T_D is
        #essentially zero, note the index of the current step and append it to
        #the stroboscopic index list.
        if (t[i]%T_D) <= tol and t[i] != 0:
            ind_strb.append(i)
    return t, theta, w, ind_strb
```

Figure 1: Above shows an RK-2 implementation for a damped, sinusoidally driven pendulum.

B

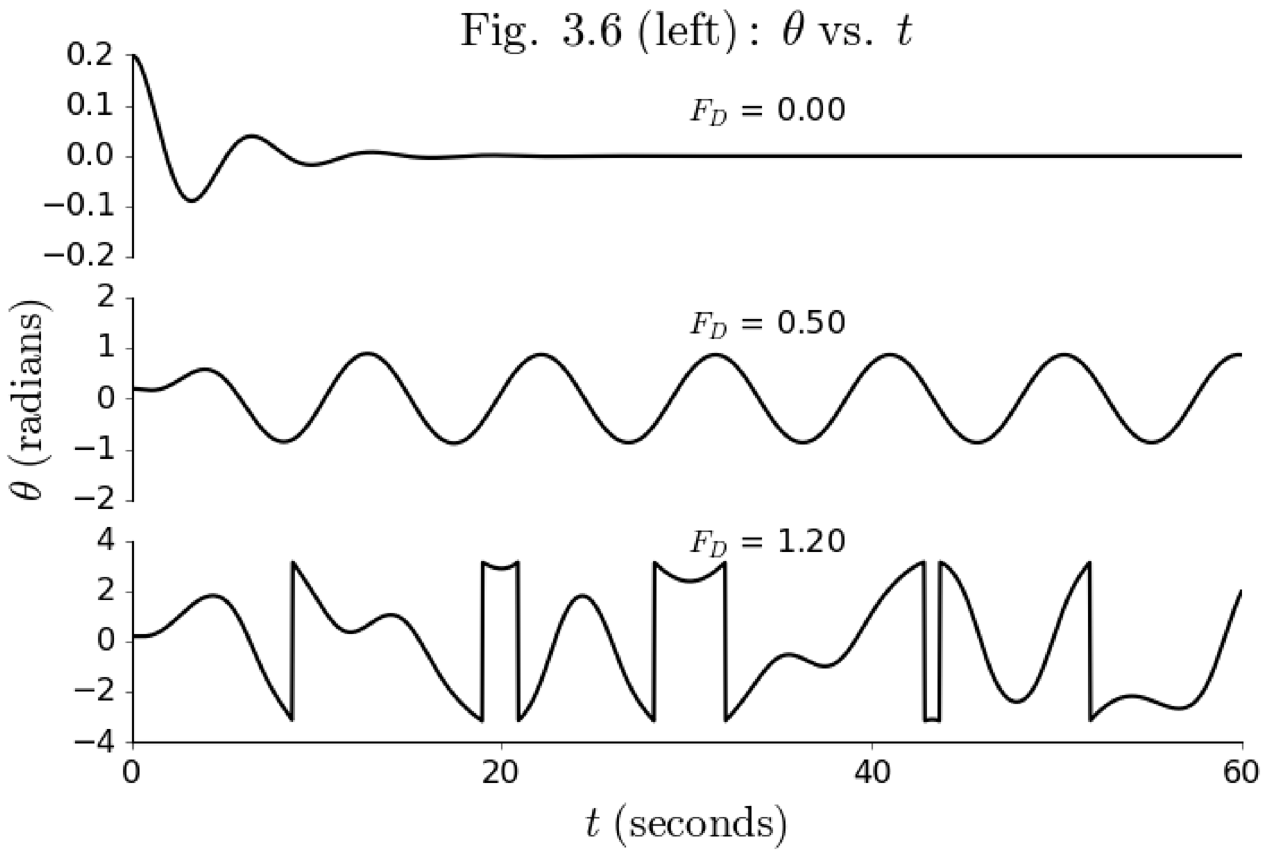


Figure 2: Left panel of Giordano's figure found on page 60. Notice for $t \in [40, 60]$ our model's graph deviates from Giordano's. The beginning of the plot is fine, so we believe this represents the regime where the pendulum begins to exhibit chaotic behavior. Since Giordano's plot was generate with Euler-Cromer, I think our result is more accurate.

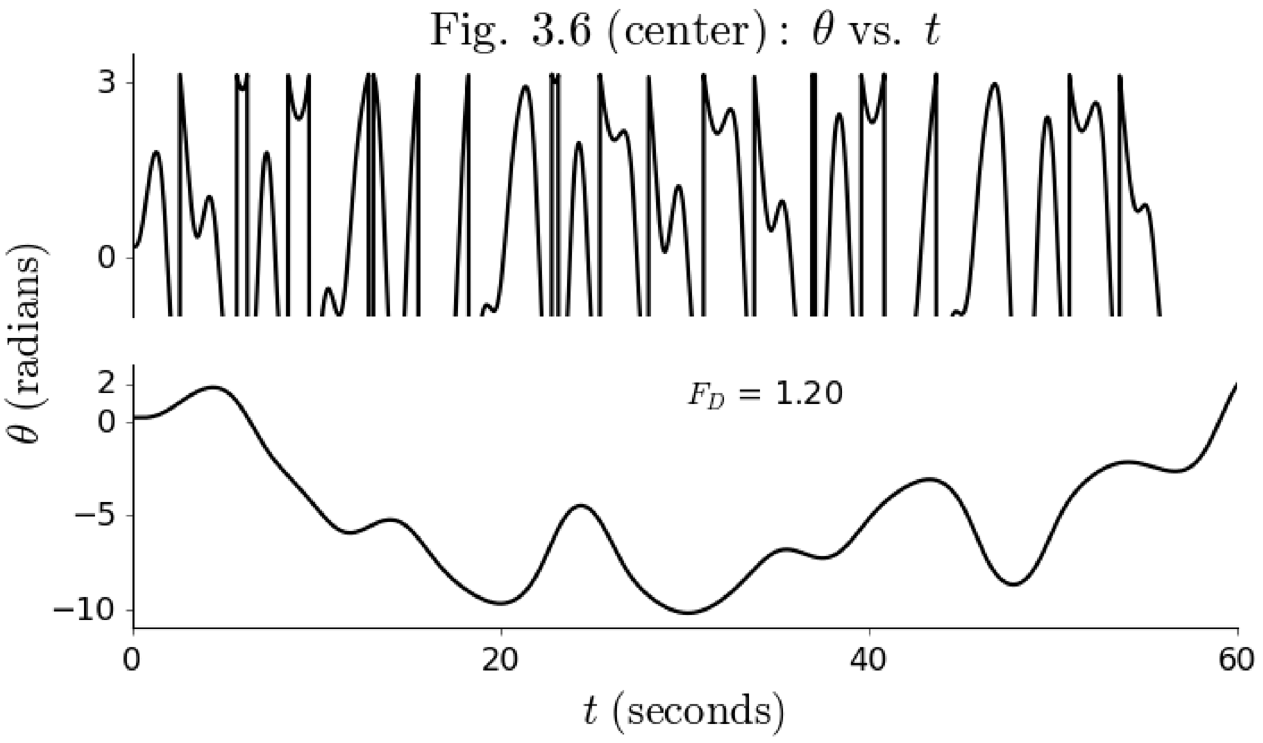


Figure 3: Center panel of Giordano's figure found on page 60. Again, for $t \in [40, 60]$ our model tends to deviate from Giordano's due to the beginnings of the chaotic regime. Note the top half of the plot differs by Giordano's significantly only at later times.

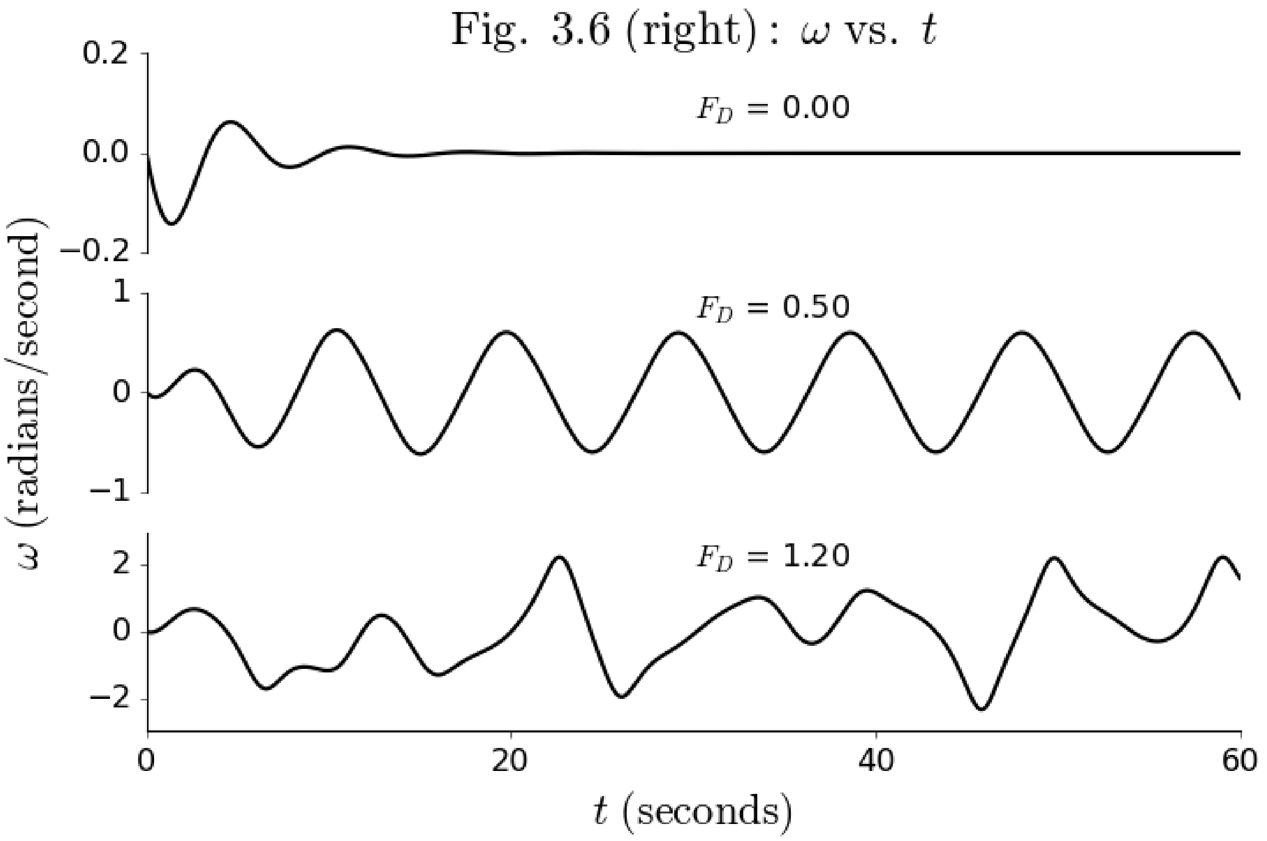


Figure 4: Right panel of Giordano's figure found on page 60. For small driving forces, our plot matches Giordano's exactly, but at least at $F_D = 1.20$, later times differ significantly due to the onset of the chaotic regime.

C

The tests that follow show that when our model runs with the same given initial conditions, our model differs from Covey's on the order of 10^{-10} on average. This average deviation is well below machine epsilon for single precision floats $\left(2^{-24} \approx 5.96 \times 10^{-8}\right)$ which makes sense since the comparison was done with all relevant variables declared as double precision floats. Additionally, the Absolute Deviation vs Time plots show that our numerical solution converges as a smaller and smaller time step is chosen. Taken as a combination, these tests indicate our RK-2 solution is extremely stable, and if Covey's data is taken to be the "true" answer to this physical situation, is accurate to $\approx \pm 10^{-10}$ with a precision of $2^{-53} \approx 1.11 \times 10^{-16}$ if conducted in high resolution (i.e. double floating point resolution).

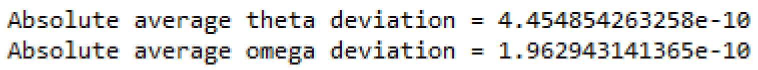
```python
#Since we want our results to be as accurate as possible for this comparison,
#ensure calculations are done in high resolution.
if gen_text_file == True and hires == False:
    print('Please ensure hires = True and then try again.')
if gen_text_file == True and hires == True:
    #Generate data for Covey's initial conditions. Note the False to obtain
    #"non-remapped theta values". Change dt_std to something nonstandard to
    #perform a convergence test.
    data = main(w0_std, theta0_std, F_D_std[2], 0.04, t0_std,
                np.float_(60.0), False)
    t_tofile     = data[0]
    theta_tofile = data[1]
    w_tofile     = data[2]
    f = open('Cemenenkoff-PHYS486-HW3.txt','w')
    for i in range(len(t_tofile)):
        f.write('%3.15f'%t_tofile[i]+'\t'+'%3.15f'%theta_tofile[i]+'\t'
                +'%3.15f\n'%w_tofile[i])
    f.close() #Close the file after writing to it.

    #We can quantify how much our calculations differ from Covey's by finding
    #the absolute difference between each set of values.
    t_Cov = [] #Initialize empty lists to import Covey's .txt file data.
    theta_Cov = []
    w_Cov = []
    with open('verify_Pendulum_HW3.txt', 'r') as file:
        next(file) #Skip the header line.
        for row in file:
            row = row.strip()
            #Each time a space is hit in the current row, store the element in
            #a list called column.
            column = row.split()
            #Append elements from column to the appropriate lists. Ensure the
            #values are stored at double precision and that hires == True.
            t_Cov.append(np.float_(column[0]))
            theta_Cov.append(np.float_(column[1]))
            w_Cov.append(np.float_(column[2]))

    w_diffs = []
    theta_diffs = []
    for i in range(len(w_Cov)):
        abserr_w = abs(w_tofile[i]-w_Cov[i])
        abserr_theta = abs(theta_tofile[i]-theta_Cov[i])
        w_diffs.append(abserr_w)
        theta_diffs.append(abserr_theta)

    w_avgdev = np.average(abserr_w)
    theta_avgdev = np.average(abserr_theta)
    print('Absolute average theta deviation = %.12e'%theta_avgdev)
    print('Absolute average omega deviation = %.12e'%w_avgdev)
```

Figure 5: Code for comparing to Covey's .txt file.



```
Absolute average theta deviation = 4.454854263258e-10
Absolute average omega deviation = 1.962943141365e-10
```

Figure 6: Since the absolute average deviation of both relevant quantities is within 10^{-10}, I am confident my model is producing accurate and precise results.

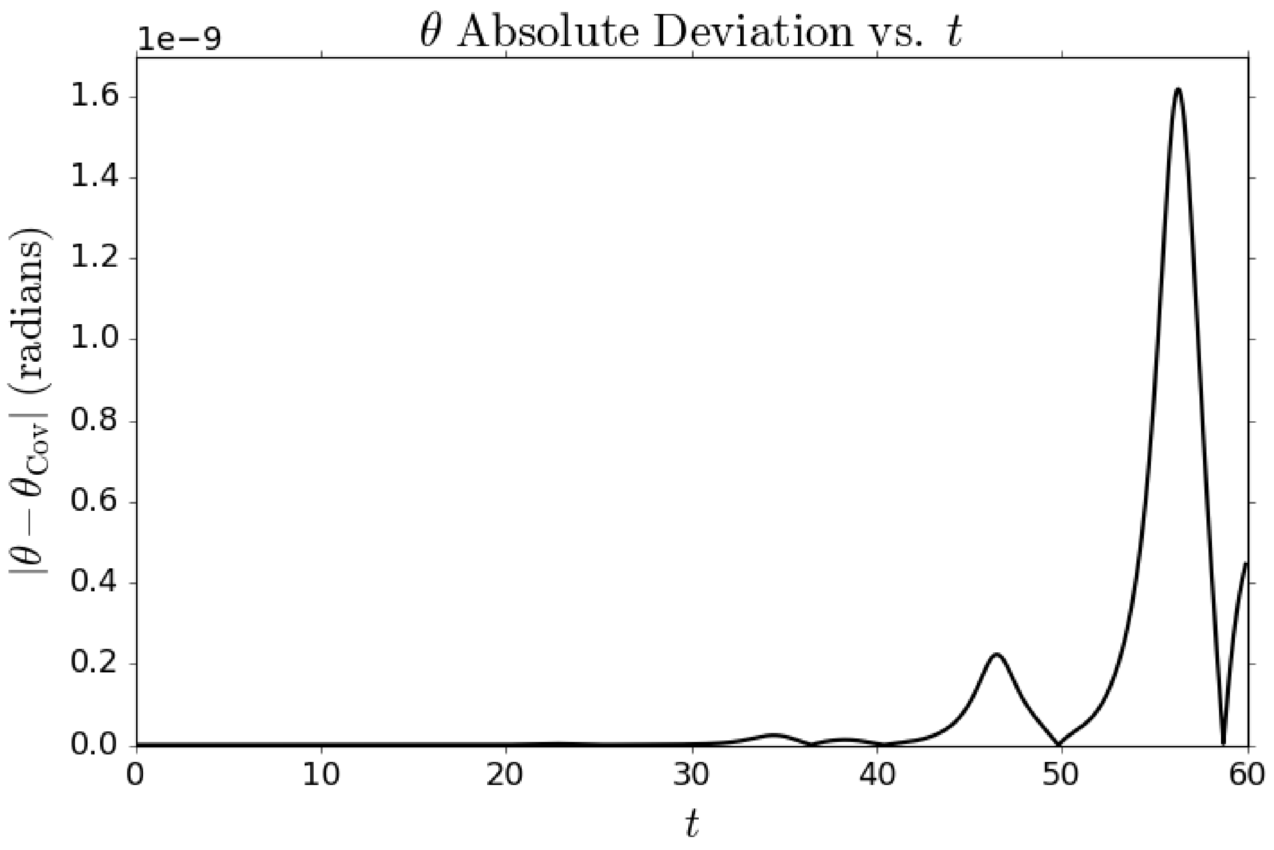


Figure 7: Here, we log the difference between Covey's data at $t = 60.0\,\mathrm{s}$ and model θ data for a time step of $dt = 0.04$. Note how the deviation starts to increase dramatically once the pendulum enters the chaotic regime.

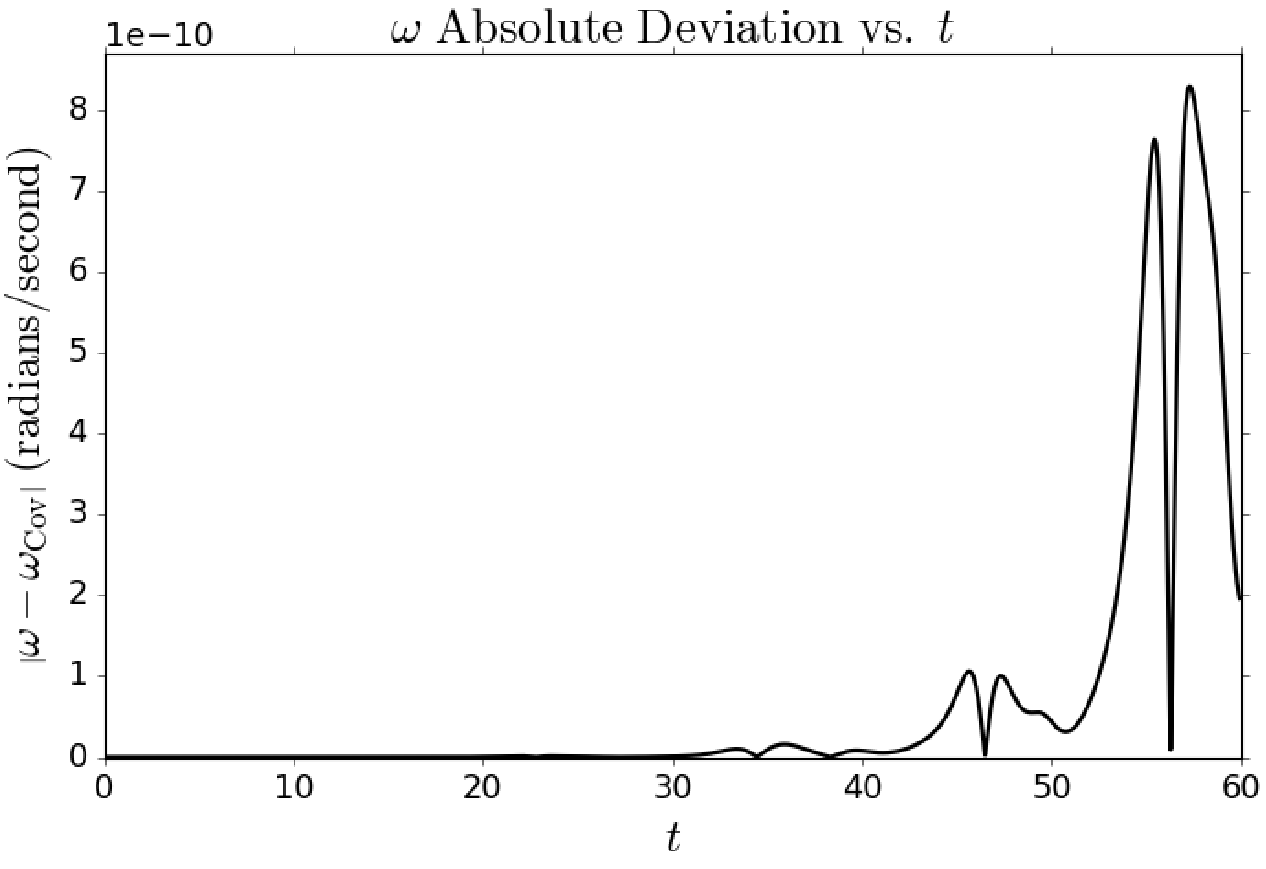


Figure 8: This plot logs the difference between Covey's data at $t = 60.0\,\mathrm{s}$ and model ω data for a time step of $dt = 0.04$. Note how the deviation starts to increase dramatically once the pendulum enters the chaotic regime.

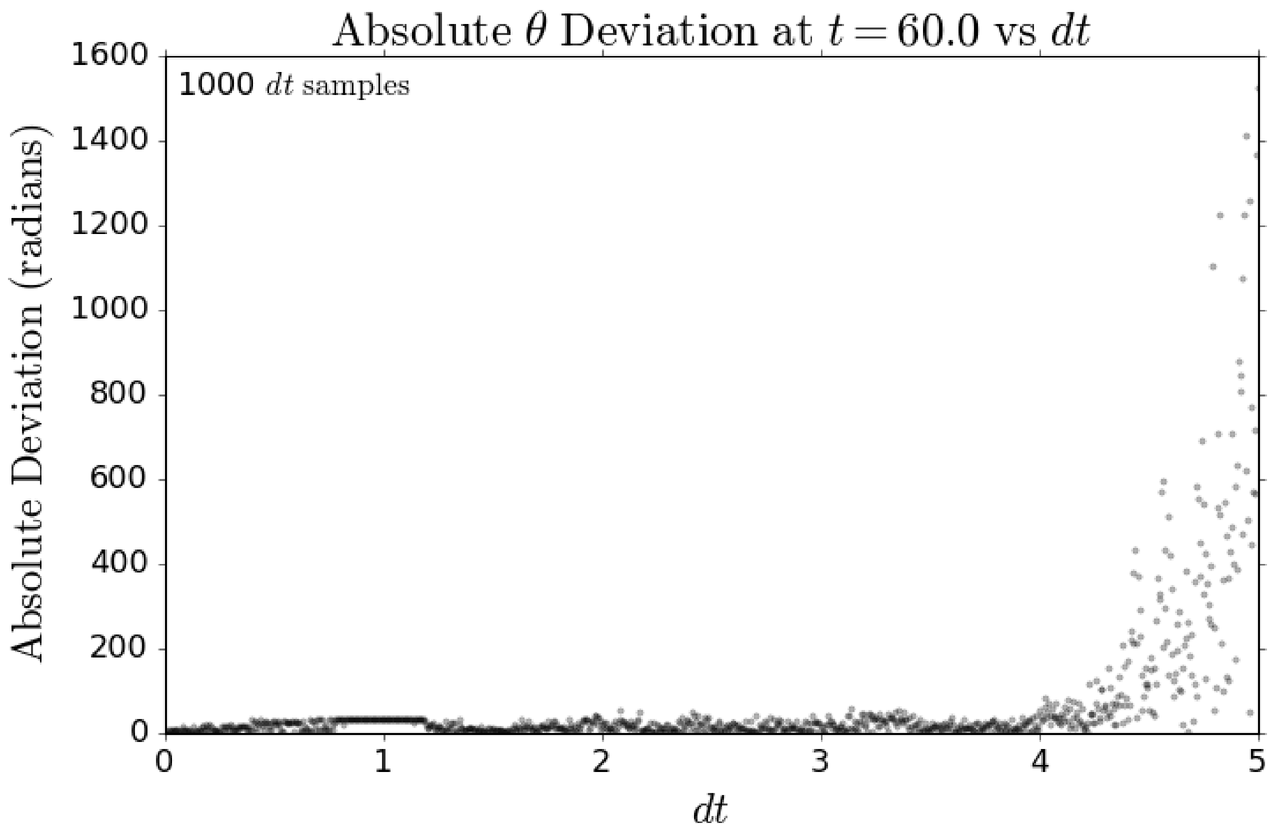


Figure 9: Here, we log the difference between Covey's data at $t = 60.0\,\mathrm{s}$ and model θ data for several different time steps. Note that as $dt \to 0$, so does the absolute deviation.

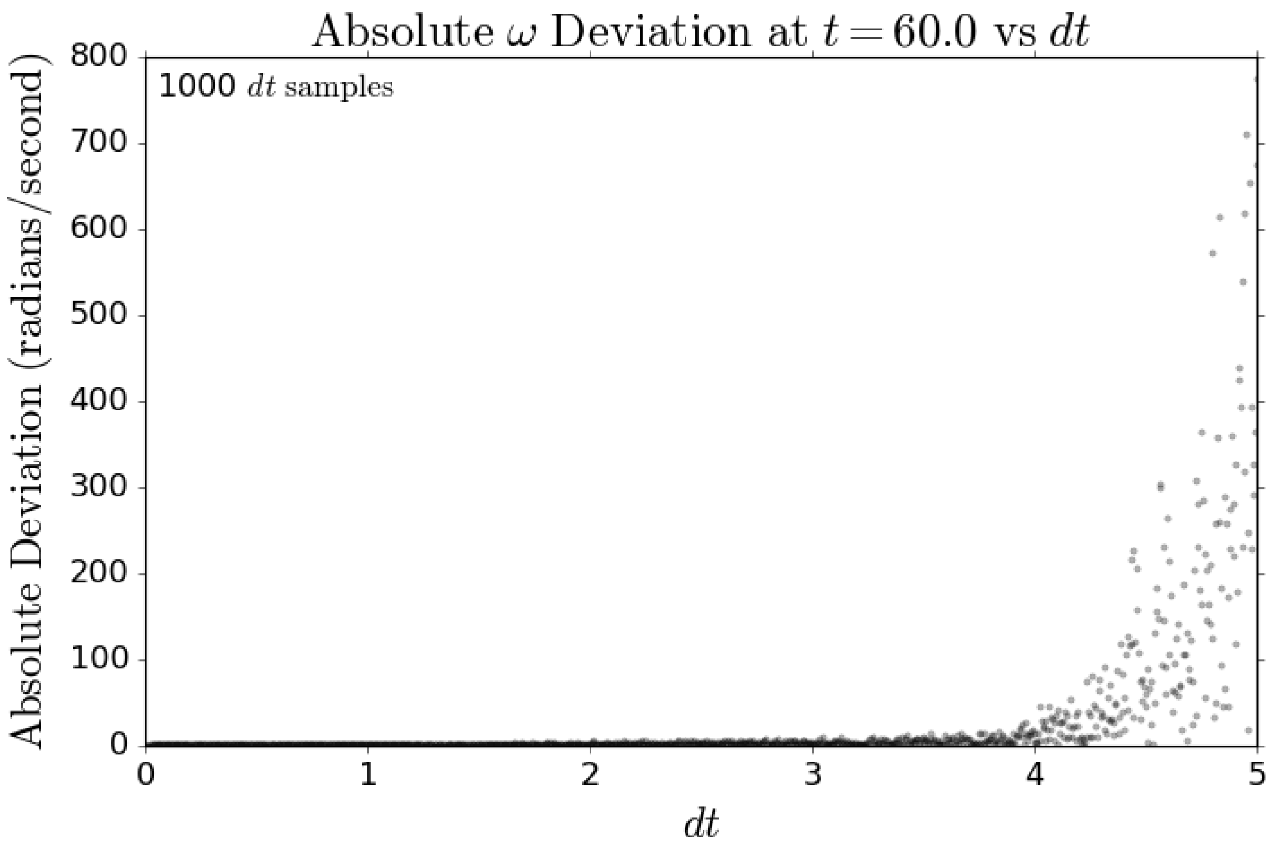


Figure 10: Similarly, we log the difference between Covey's data at $t = 60.0\,\mathrm{s}$ and model ω data for several different time steps. Note that as $dt \to 0$, so does the absolute deviation.

D

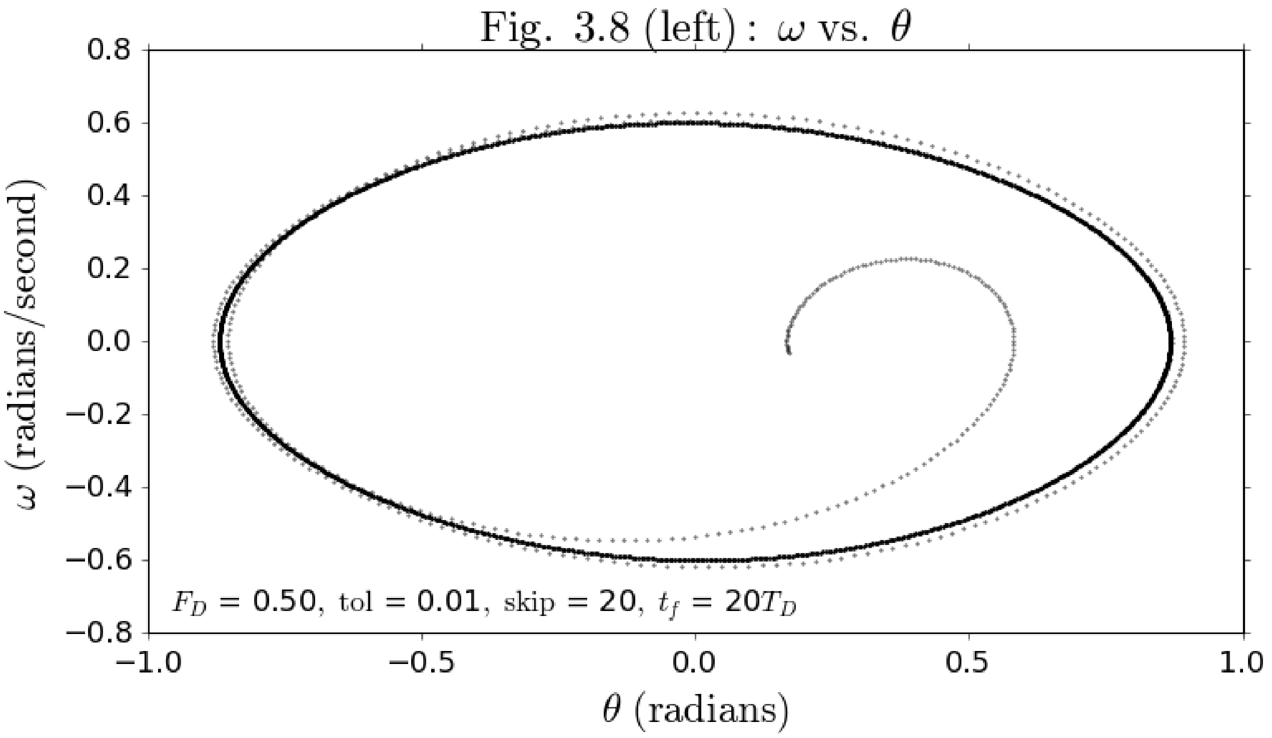


Figure 11: Recreation of the right panel of Figure 3.8 from Giordano page 63. Our plot matches Giordano's almost exactly.

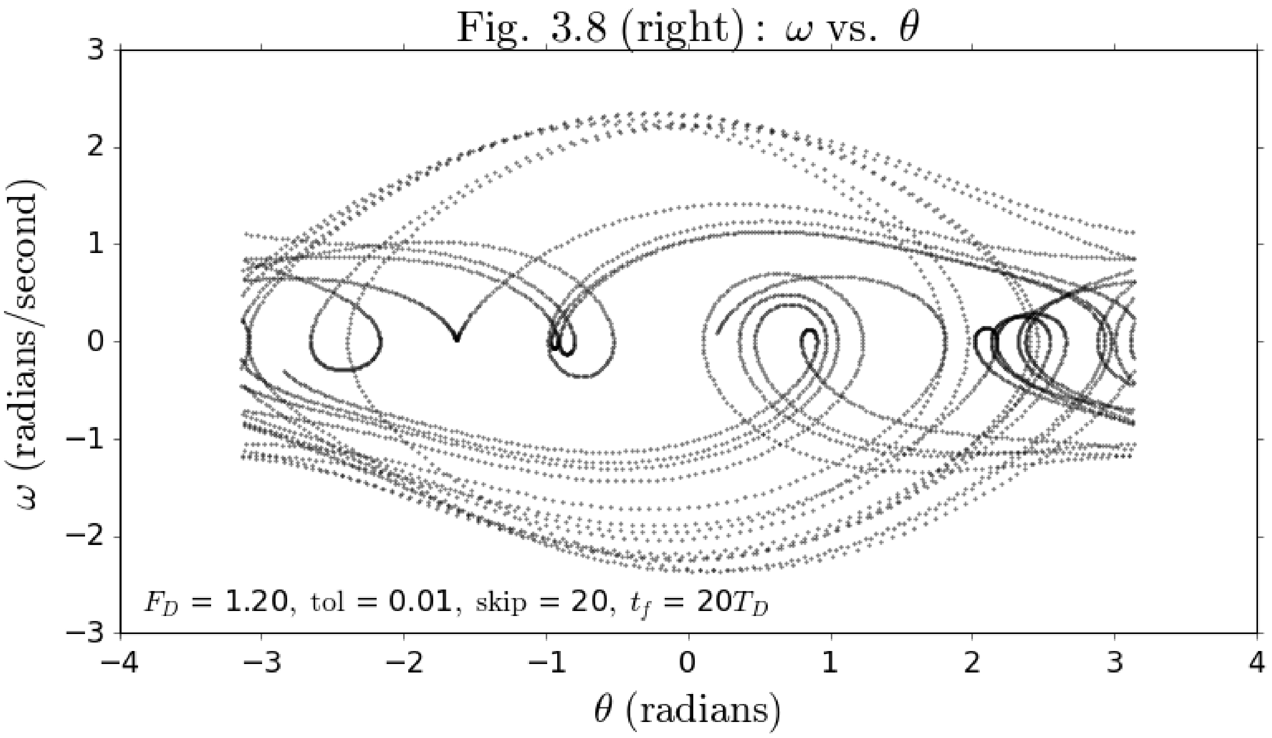
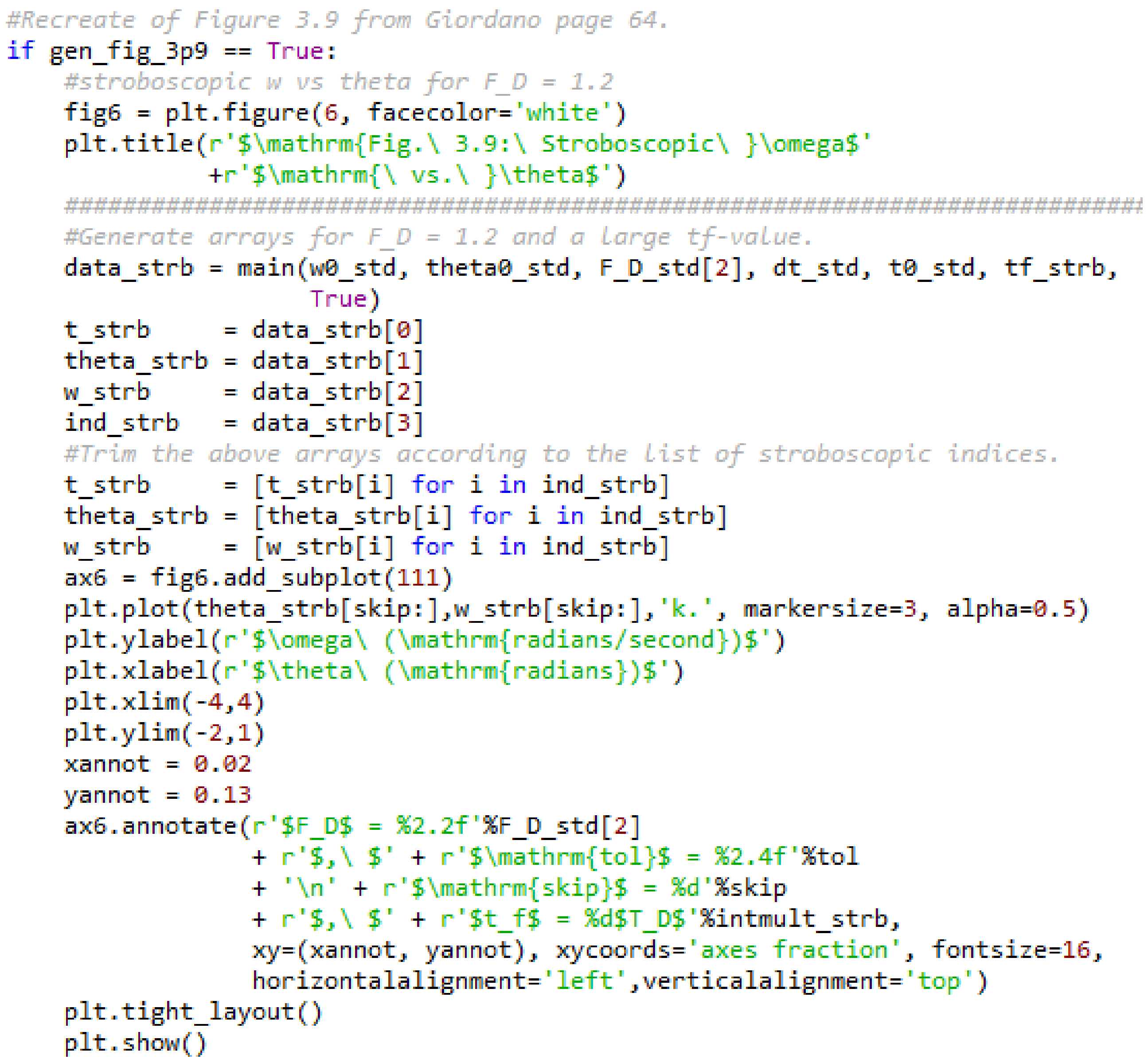


Figure 12: Recreation of the left panel of Figure 3.8 from Giordano page 63. Our plot matches Giordano's almost exactly.

E



```
#Recreate of Figure 3.9 from Giordano page 64.
if gen_fig_3p9 == True:
    #stroboscopic w vs theta for F_D = 1.2
    fig6 = plt.figure(6, facecolor='white')
    plt.title(r'$\mathrm{Fig.\ 3.9:\ Stroboscopic\ }\omega$'
              +r'$\mathrm{\ vs.\ }\theta$')
    ################################################################################
    #Generate arrays for F_D = 1.2 and a large tf-value.
    data_strb = main(w0_std, theta0_std, F_D_std[2], dt_std, t0_std, tf_strb,
                     True)
    t_strb     = data_strb[0]
    theta_strb = data_strb[1]
    w_strb     = data_strb[2]
    ind_strb   = data_strb[3]
    #Trim the above arrays according to the list of stroboscopic indices.
    t_strb     = [t_strb[i] for i in ind_strb]
    theta_strb = [theta_strb[i] for i in ind_strb]
    w_strb     = [w_strb[i] for i in ind_strb]
    ax6 = fig6.add_subplot(111)
    plt.plot(theta_strb[skip:],w_strb[skip:],'k.', markersize=3, alpha=0.5)
    plt.ylabel(r'$\omega\ (\mathrm{radians/second})$')
    plt.xlabel(r'$\theta\ (\mathrm{radians})$')
    plt.xlim(-4,4)
    plt.ylim(-2,1)
    xannot = 0.02
    yannot = 0.13
    ax6.annotate(r'$F_D$ = %2.2f'%F_D_std[2]
                 + r'$,\ $' + r'$\mathrm{tol}$ = %2.4f'%tol
                 + '\n' + r'$\mathrm{skip}$ = %d'%skip
                 + r'$,\ $' + r'$t_f$ = %d$T_D$'%intmult_strb,
                 xy=(xannot, yannot), xycoords='axes fraction', fontsize=16,
                 horizontalalignment='left',verticalalignment='top')
    plt.tight_layout()
    plt.show()
```

Figure 13: Code used to generate Figure 3.9 from Giordano page 64.

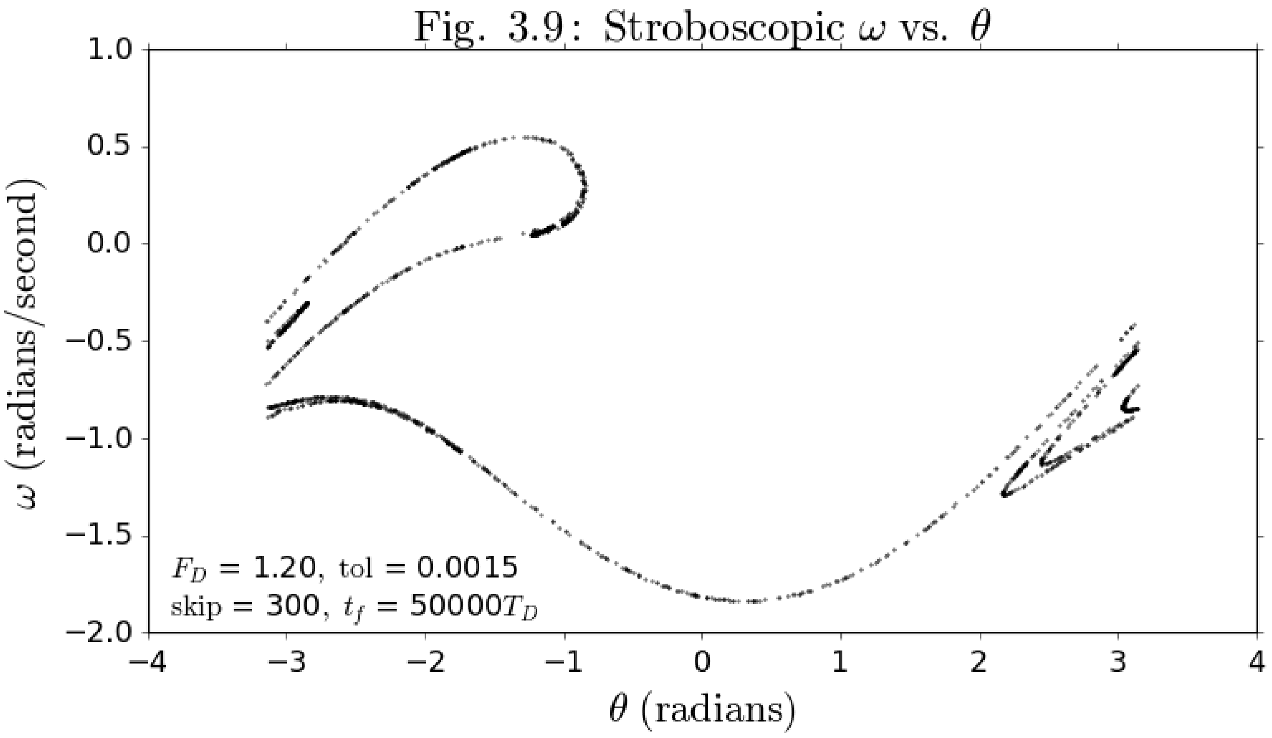


Figure 14: Recreation of Figure 3.9 from Giordano page 64. Notice that our plot matches Giordano's almost exactly.

F

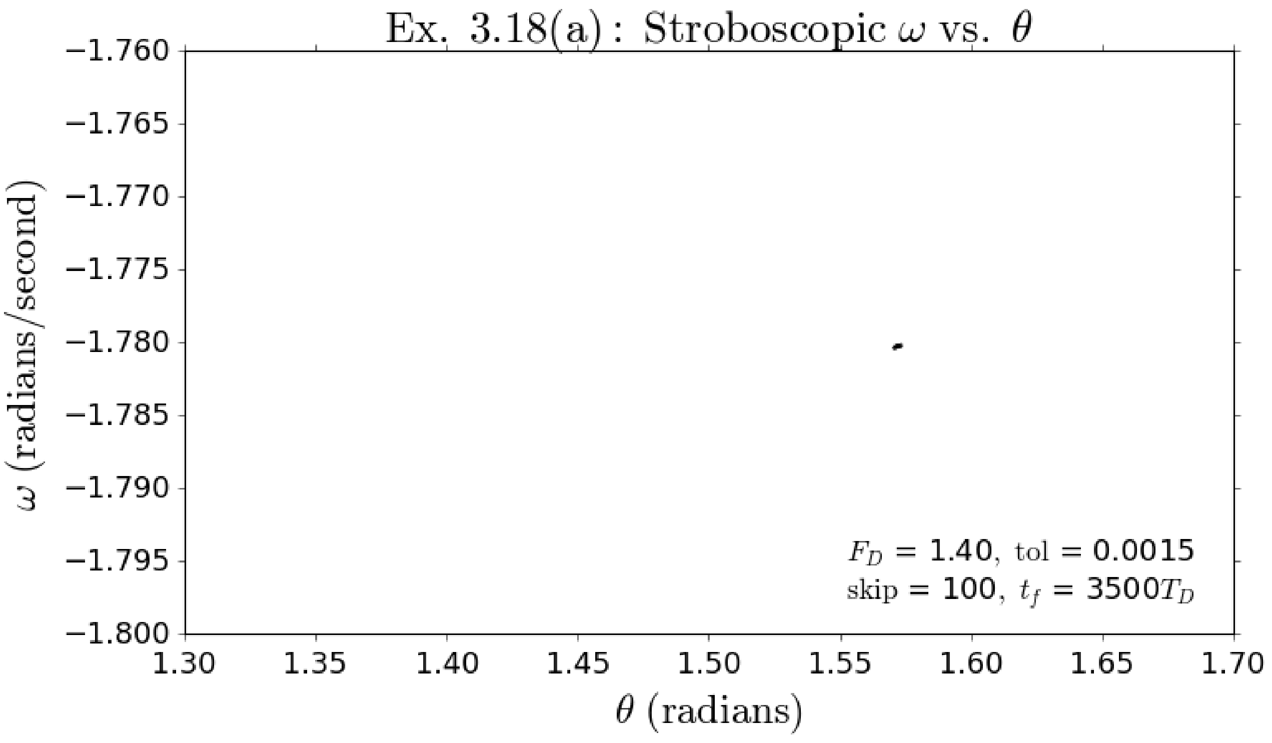


Figure 15: Since the period of motion is exactly equal to the period of the drive, we get one point in phase space.

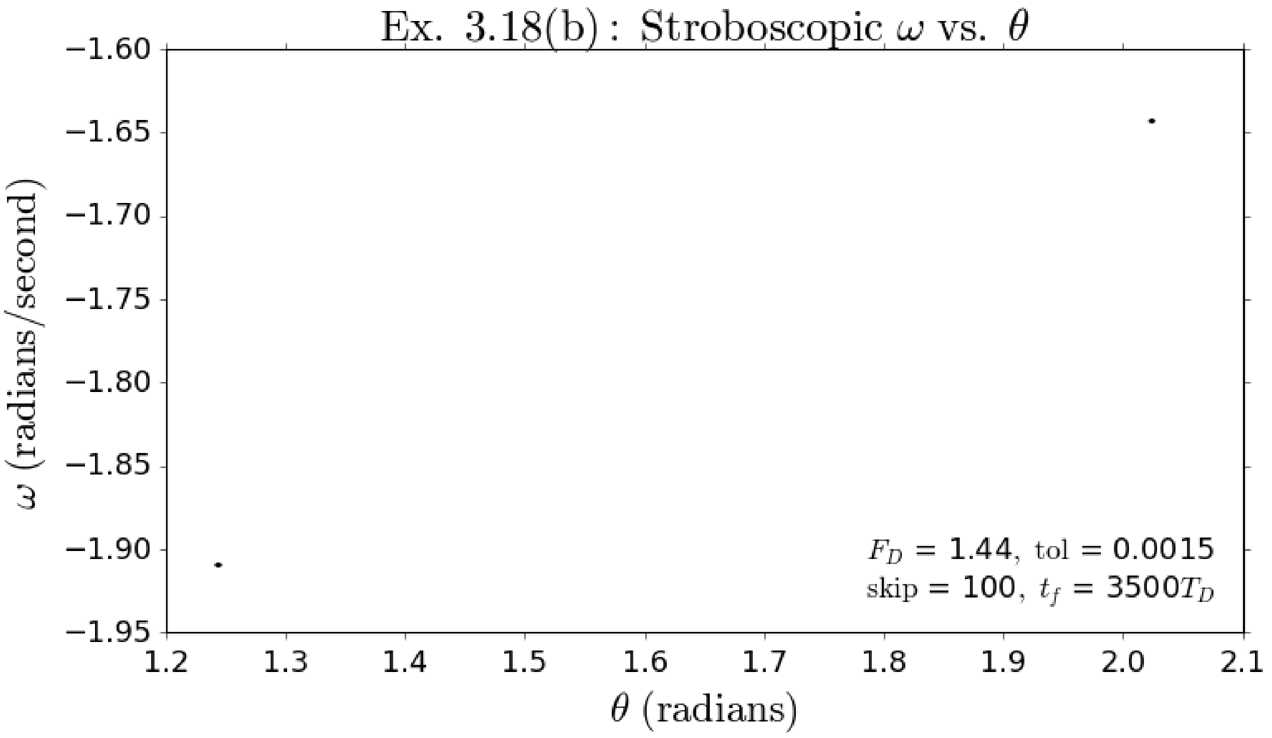


Figure 16: Here, since the period of motion is exactly twice the period of the drive, we get two points in phase space.

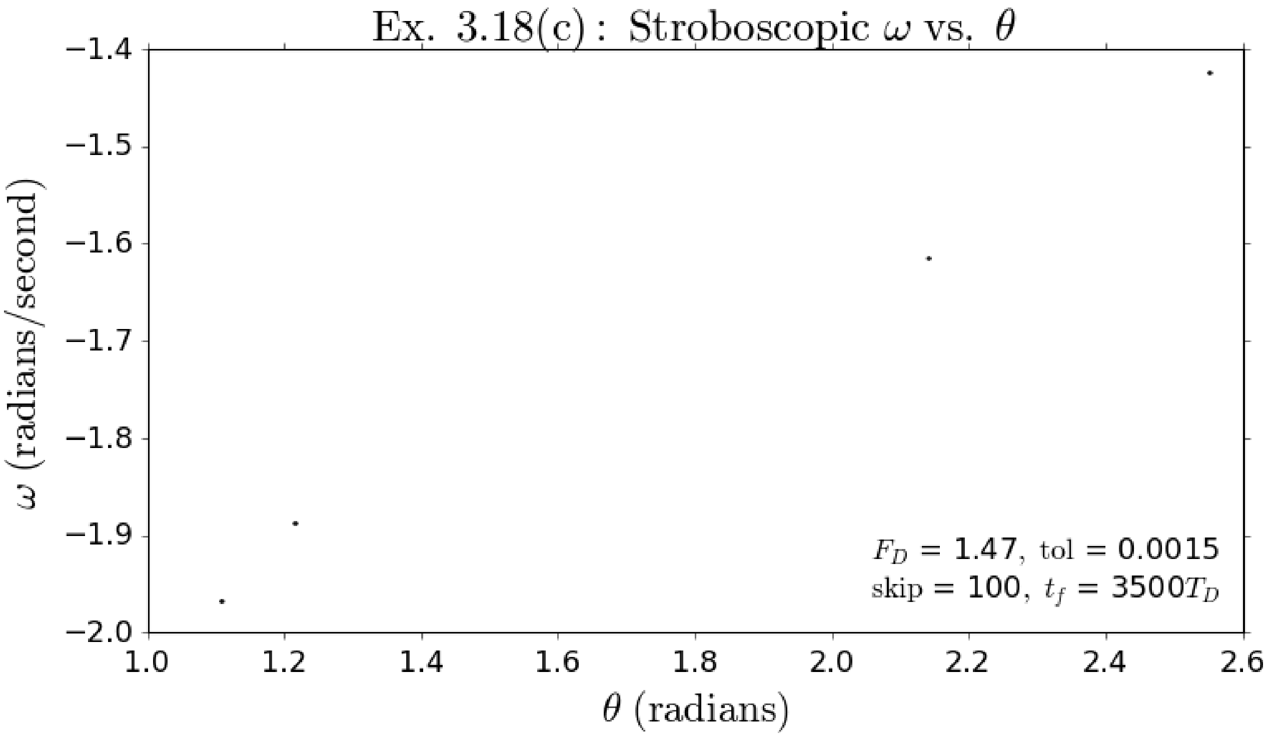


Figure 17: Lastly, since the period of motion here is exactly four times the period of the drive, we get four points in phase space.

G

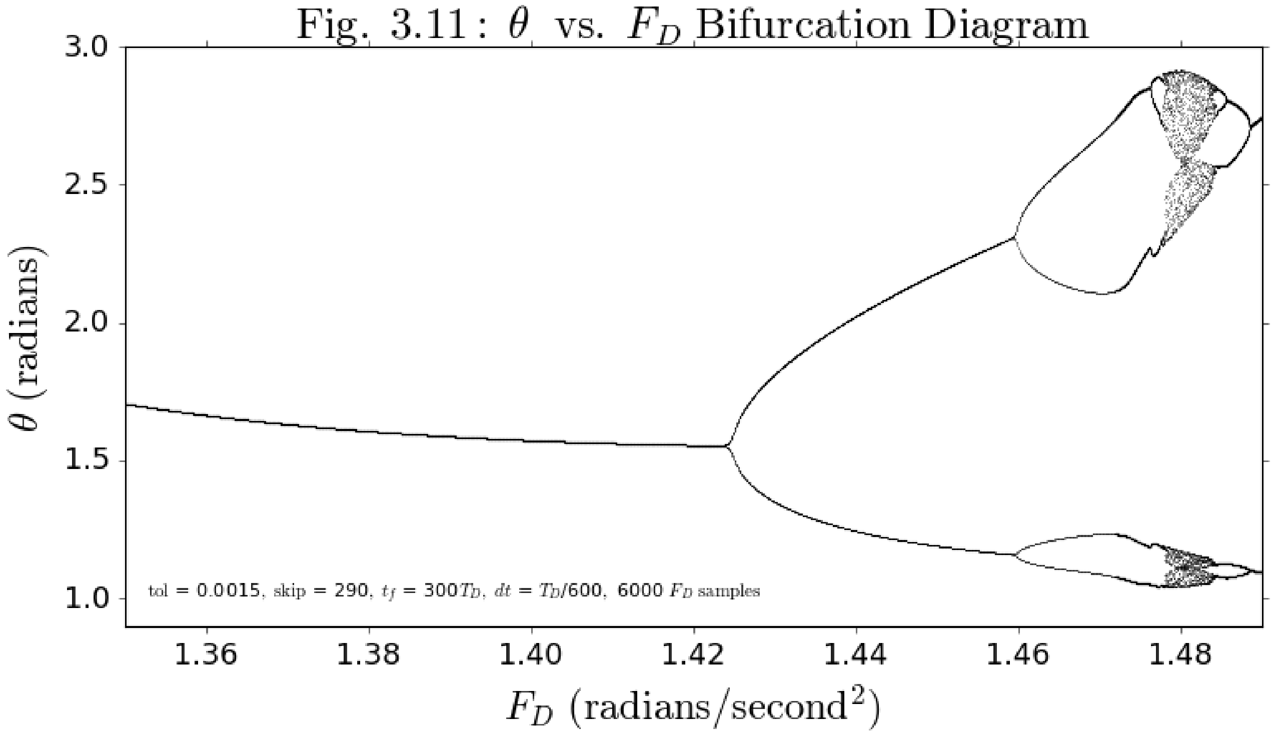


Figure 18: Recreation of Figure 3.11 from Giordano page 68. Again, our plot matches Giordano's almost exactly, except this time it is in high resolution.