NON-LINEAR DYNAMICS AND VIBRATIONS PROJECT

HARMONIC BALANCE METHOD

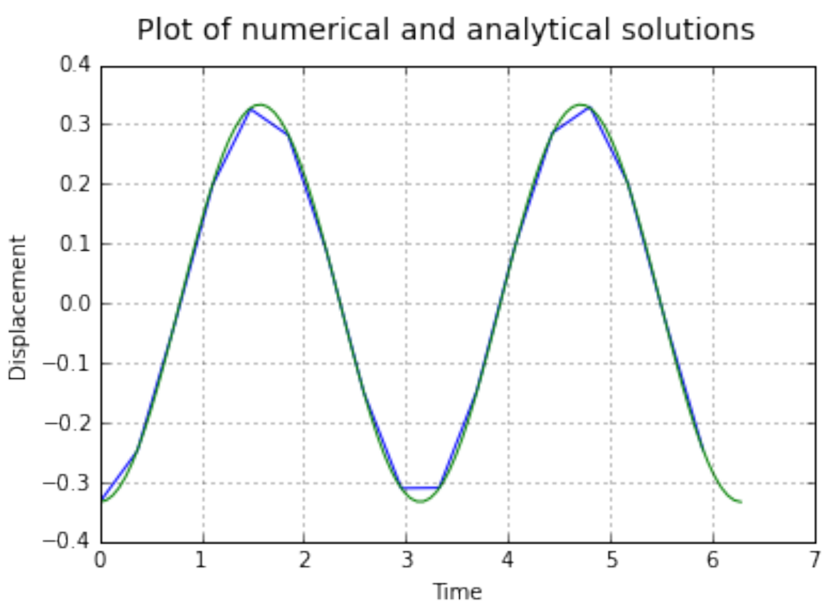
Initially start with linear equation solving using FFT and comparing with the general solution

$\ddot{x} + x = cos(2t)$

In [13]:
```
%matplotlib inline
import pylab as pl
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as sci

# I am solving the equation dotdotX + X = cos(2*t)


# this is method 1
N = 17
t = np.linspace(0, 2*np.pi, N+1)    # time samples of forcing function
t = t[0:-1]                         # Removing the extra sample
f = np.cos(2*t)                     # My forcing function
F = np.fft.fft(f)
omega = np.fft.fftfreq(N, 1/N) + 0.0000001 # list of frequencies
X = np.divide(F, 1 - omega**2)
x = np.fft.ifft(X)

t_eval = np.linspace(0,2*np.pi,100)
X_analytical = -(np.cos(2*t_eval)/3)

#pl.scatter(t,x)
#pl.show()

# this is the Dr. Slater method, this will work with nonlinear functio
ns
xbar = f*0 + np.cos(2*t)

def FUNCTION(xbar):
```

```
    N = len(xbar)
    Xbar = np.fft.fft(xbar)
    omega = np.fft.fftfreq(N, 1/N) + 0.0000001 # list of frequencies
    dotdotxbar = np.fft.ifft(np.multiply((1j*omega)**2,Xbar))
    R = np.sum(np.real(np.abs(dotdotxbar + xbar - f)))
    return R

optimizedResults = sci.minimize(FUNCTION, xbar, method='SLSQP')
xbar = optimizedResults.x

#optimizedResults = sci.fmin(FUNCTION, xbar, args=(), xtol=0.0000001,
ftol=0.0000001,maxiter=10000, maxfun= 100000)
#print(optimizedResults)
#xbar = optimizedResults

# func, x0, args=(), xtol=0.0001, ftol=0.0001, maxiter=None, maxfun=No
ne, full_output=0, disp=1, retall=0, callback=None


print(optimizedResults)
print(xbar)

# pl.plot(t_eval, X_analytical)
fig = plt.figure()
pl.plot(t,xbar)
pl.plot(t_eval, X_analytical)
fig.suptitle('Plot of numerical and analytical solutions', fontsize =
14)
plt.xlabel('Time')
plt.ylabel('Displacement')
plt.grid()
pl.show()
```

```
 message: 'Optimization terminated successfully.'
       x: array([-0.33399111, -0.24765072, -0.03254632,  0.19884773,
0.32566032,
        0.28171368,  0.090065  , -0.14903459, -0.31051256, -0.309779
85,
       -0.14694131,  0.09323082,  0.28551165,  0.32957866,  0.202354
9 ,
       -0.02992393, -0.24624732])
     nit: 57
     fun: 0.00061673628981899964
    njev: 57
     jac: array([ 50.20082419, -49.72596857,  24.96704244,  14.99586
373,
       -19.59926716,   4.82634378,  -0.98889774,   3.21597896,
        -2.61095873,   8.83656553, -27.47077913,  51.4611969 ,
       -47.60202626,  26.80753248,  14.87814988, -11.54410474,
       -26.40722304,   0.        ])
 success: True
    nfev: 1257
  status: 0
[-0.33399111 -0.24765072 -0.03254632  0.19884773  0.32566032  0.2817
1368
  0.090065   -0.14903459 -0.31051256 -0.30977985 -0.14694131  0.0932
3082
  0.28551165  0.32957866  0.2023549  -0.02992393 -0.24624732]
```



Now non-Linear governing equation

$\ddot{x} + \dot{x} + x - (x^3) = cos(2t)$

In [30]:
```
%matplotlib inline
import pylab as pl
import numpy as np
import scipy.optimize as sci
import scipy.integrate as sp

# this code is the nonlinear case of \dotdot{x} + \dot{x} + x - x**3 =
cos(2*t)

# Plotting some solutions
def solve_hw2(max_time=5,x0 = 1, v0 = 1):
    def hw2_deriv(x1_x2,t):
        x1, x2 = x1_x2
        return [x2, -x2-x1+0*x1**3+np.cos(2*t)]
    t = np.linspace(0, max_time, int(2000*max_time))
    x_t = sp.odeint(hw2_deriv, [x0,v0], t)
    return t, x_t

t, x_t = solve_hw2(max_time=10*np.pi, x0 = 0, v0 = 0)

pl.plot(t,x_t[:,0])
pl.xlabel('$t$', fontsize=20)
pl.ylabel('$x(t)$', fontsize=20)
pl.grid()

# this is method 1
N = 99
t = np.linspace(0, 10*np.pi, N+1)    # time samples of forcing functio
n
t = t[0:-1]                         # Removing the extra sample
f = np.cos(2*t)                     # My forcing function
T = t[-1]

# this is the Dr. Slater method, this will work with nonlinear functio
ns
xbar = 10*f

def FUNCTION(xbar):
    N = len(xbar)
    Xbar = np.fft.fft(xbar)
    omega = np.fft.fftfreq(N, T/(2*np.pi*N) )# + 0.0000001 # list of f
requencies
    dotxbar = np.fft.ifft(np.multiply((1j*omega),Xbar))
    dotdotxbar = np.fft.ifft(np.multiply((1j*omega)**2,Xbar))
    R = dotdotxbar + dotxbar + xbar - 0*xbar**3 - f
    R = R**2
```

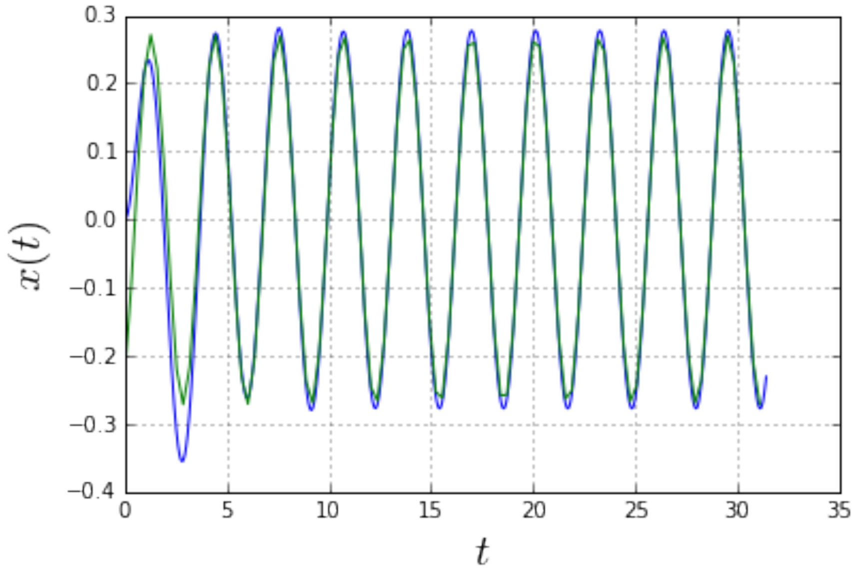
```
    R = np.sum(R)
    return R

optimizedResults = sci.minimize(FUNCTION, xbar, method='SLSQP')
xbar = optimizedResults.x

print(optimizedResults)
print(xbar)

pl.plot(t,xbar)
fig.suptitle('Plot of numerical and analytical solutions', fontsize =
14)
pl.show()
```

```
 message: 'Optimization terminated successfully.'
       x: array([-0.2269576 , -0.09457638,  0.07464096,  0.21477433,
0.2712435 ,
        0.22206631,  0.08640336, -0.08294603, -0.21998014, -0.271367
19,
       -0.21705891, -0.07824011,  0.0910573 ,  0.224883  ,  0.271101
27,
        0.21175185,  0.06991033, -0.09916168, -0.22962745, -0.270672
43,
       -0.20630755, -0.06161503,  0.10706445,  0.23403439,  0.269836
51,
        0.20055443,  0.05313599, -0.11496897, -0.23833719, -0.268887
41,
       -0.19474358, -0.04474781,  0.12266197,  0.24230869,  0.267540
44,
        0.18856447,  0.03614386, -0.13034627, -0.24606704, -0.265977
5 ,
       -0.18229411, -0.02763065,  0.13780366,  0.24956817,  0.264124
95,
        0.17580149,  0.01900045, -0.14520008, -0.25284252, -0.262036
93,
       -0.16919084, -0.01047116,  0.15231495,  0.25577571,  0.259637
89,
        0.16237244,  0.00186643, -0.15938775, -0.25855739, -0.257036
24,
       -0.1554077 ,  0.00674356,  0.16624768,  0.26099878,  0.254087
75,
        0.14823427, -0.0153723 , -0.17300518, -0.26328521, -0.251032
88,
       -0.14102553,  0.02391705,  0.17952267,  0.26519556,  0.247574
49,
        0.13350905, -0.03254442, -0.18595457, -0.26693181, -0.244000
56,
       -0.12602346,  0.0410053 ,  0.19206557,  0.26829345,  0.240018
84,
        0.11826323, -0.04956223, -0.19808594, -0.26947622, -0.235912
```

```
89,
       -0.11047376,  0.05799932,  0.20386087,  0.27032064,  0.231491
68,
        0.10252667, -0.06638535, -0.20944567, -0.27095904])
     nit: 7
     fun: (9.3350815920486145e-06+2.1763039852596424e-17j)
    njev: 7
     jac: array([ 0.02963091, -0.03594344,  0.02704825, -0.0194165 ,
0.02821188,
       -0.04945367,  0.06693083, -0.07285542,  0.07138489, -0.065417
59,
        0.05643656, -0.04334512,  0.01634988,  0.02058951, -0.040981
44,
        0.03206228, -0.01267933,  0.00369413, -0.00264571, -0.001145
39,
        0.00296166,  0.00222337, -0.0111953 ,  0.02540854, -0.045823
36,
        0.06578334, -0.08210321,  0.08292786, -0.0691236 ,  0.066022
66,
       -0.0659803 ,  0.06098209, -0.07374723,  0.07287435, -0.039344
7 ,
        0.02159294, -0.01278491, -0.01700141,  0.03629565, -0.035189
46,
        0.035307  , -0.02926687,  0.00654301,  0.011297  , -0.008654
35,
        0.00373906,  0.00734438, -0.02707554,  0.02346103, -0.009847
45,
        0.00748757, -0.00410849,  0.01823706, -0.04646991,  0.051245
05,
       -0.04769921,  0.04923475, -0.04015107,  0.0304789 , -0.023422
41,
        0.00835121,  0.00655341, -0.01541291,  0.03506062, -0.057920
21,
        0.05067566, -0.03565204,  0.03732305, -0.03476962,  0.038933
07,
       -0.05004896,  0.03143551,  0.00536126, -0.03615547,  0.061615
28,
       -0.07660452,  0.08379688, -0.09643309,  0.10870267, -0.103097
64,
        0.08517776, -0.07080981,  0.04506939, -0.00571451, -0.017942
37,
        0.02575222, -0.03941623,  0.05170951, -0.05091811,  0.051812
94,
       -0.06807327,  0.07341853, -0.05505013,  0.05553206, -0.067772
83,
        0.05134556, -0.03168856,  0.02564437, -0.02240041,  0.
])
 success: True
    nfev: 734
  status: 0
```

```
[-0.2269576  -0.09457638  0.07464096  0.21477433  0.2712435   0.2220
6631
  0.08640336 -0.08294603 -0.21998014 -0.27136719 -0.21705891 -0.0782
4011
  0.0910573   0.224883    0.27110127  0.21175185  0.06991033 -0.0991
6168
 -0.22962745 -0.27067243 -0.20630755 -0.06161503  0.10706445  0.2340
3439
  0.26983651  0.20055443  0.05313599 -0.11496897 -0.23833719 -0.2688
8741
 -0.19474358 -0.04474781  0.12266197  0.24230869  0.26754044  0.1885
6447
  0.03614386 -0.13034627 -0.24606704 -0.2659775  -0.18229411 -0.0276
3065
  0.13780366  0.24956817  0.26412495  0.17580149  0.01900045 -0.1452
0008
 -0.25284252 -0.26203693 -0.16919084 -0.01047116  0.15231495  0.2557
7571
  0.25963789  0.16237244  0.00186643 -0.15938775 -0.25855739 -0.2570
3624
 -0.1554077   0.00674356  0.16624768  0.26099878  0.25408775  0.1482
3427
 -0.0153723  -0.17300518 -0.26328521 -0.25103288 -0.14102553  0.0239
1705
  0.17952267  0.26519556  0.24757449  0.13350905 -0.03254442 -0.1859
5457
 -0.26693181 -0.24400056 -0.12602346  0.0410053   0.19206557  0.2682
9345
  0.24001884  0.11826323 -0.04956223 -0.19808594 -0.26947622 -0.2359
1289
 -0.11047376  0.05799932  0.20386087  0.27032064  0.23149168  0.1025
2667
 -0.06638535 -0.20944567 -0.27095904]
```

```
/Users/soumithvodnala/anaconda/lib/python3.5/site-packages/scipy/opt
imize/slsqp.py:62: ComplexWarning: Casting complex values to real di
scards the imaginary part
  jac[i] = (func(*((x0+dx,)+args)) - f0)/epsilon
/Users/soumithvodnala/anaconda/lib/python3.5/site-packages/scipy/opt
imize/slsqp.py:406: ComplexWarning: Casting complex values to real d
iscards the imaginary part
  slsqp(m, meq, x, xl, xu, fx, c, g, a, acc, majiter, mode, w, jw)
```



linear equation with damping

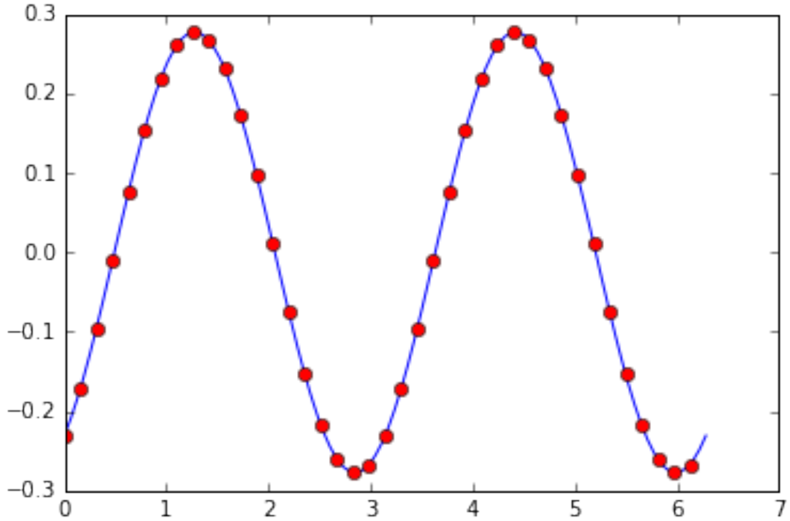
$\ddot{x} + \dot{x} + x = cos(2t)$

In [50]:
```
%matplotlib inline
import pylab as pl
import numpy as np
import scipy.optimize as sci

# This equation has a damped term
# I am solving the equation dotdotX + dotX + X = cos(2*t)

# this is method 1
N = 40
t = np.linspace(0, 2*np.pi, N+1)    # time samples of forcing function
t = t[0:-1]                          # Removing the extra sample
f = np.cos(2*t)                      # My forcing function

t_eval = np.linspace(0,2*np.pi,100)
X_analytical = (2/13)*np.sin(2*t_eval) - (3/13)*np.cos(2*t_eval)

# this is the Dr. Slater method, this will work with nonlinear functio
ns
xbar = f

def FUNCTION(xbar):
    N = len(xbar)
    Xbar = np.fft.fft(xbar)
    omega = np.fft.fftfreq(N, 1/N) + 0.0000001 # list of frequencies
    dotxbar = np.fft.ifft(np.multiply((1j*omega),Xbar))
    dotdotxbar = np.fft.ifft(np.multiply((1j*omega)**2,Xbar))
    R = dotdotxbar + dotxbar + xbar - f
    R = R**2
    R = np.sum(R)
    return R

optimizedResults = sci.minimize(FUNCTION, xbar, method='SLSQP')
xbar = optimizedResults.x

print(optimizedResults)
print(xbar)
pl.plot(t_eval, X_analytical)
pl.plot(t,xbar,'ro')
pl.show()
```

```
 message: 'Optimization terminated successfully.'
       x: array([-0.23078748, -0.17195167, -0.09628588, -0.01119672,
0.07498669,
        0.15382806,  0.2176098 ,  0.26008859,  0.27710652,  0.266997
57,
        0.2307509 ,  0.17191524,  0.09624947,  0.01116024, -0.075022
99,
       -0.15386426, -0.21764609, -0.26012496, -0.27714288, -0.267033
```

```
89,
       -0.23078764, -0.17195183, -0.09628583, -0.01119651,  0.074986
62,
        0.15382803,  0.21760979,  0.26008849,  0.2771066 ,  0.266997
61,
        0.23075103,  0.17191536,  0.09624953,  0.0111603 , -0.075023
03,
       -0.15386438, -0.21764624, -0.26012494, -0.27714293, -0.267033
93])
     nit: 2
     fun: (1.870940816986629e-08+8.579837955882194e-11j)
    njev: 2
     jac: array([-0.00098101, -0.00347981,  0.01079302, -0.01592444,
0.02098032,
       -0.02296801,  0.02703271, -0.0265175 ,  0.02122016, -0.013295
44,
        0.01481399, -0.02200066,  0.03179598, -0.03448112,  0.031857
78,
       -0.02466498,  0.01964074, -0.01324119,  0.00906573, -0.003089
37,
        0.00139186, -0.00101529, -0.00056345,  0.00445414,  0.001826
87,
       -0.01452569,  0.03107271, -0.03938777,  0.03622966, -0.024167
58,
        0.01751106, -0.01471969,  0.01539185, -0.01140917,  0.004536
4 ,
        0.00655207, -0.01494941,  0.01919772, -0.01357294,  0.007196
62,  0.        ])
 success: True
    nfev: 93
  status: 0
[-0.23078748 -0.17195167 -0.09628588 -0.01119672  0.07498669  0.1538
2806
  0.2176098   0.26008859  0.27710652  0.26699757  0.2307509   0.1719
1524
  0.09624947  0.01116024 -0.07502299 -0.15386426 -0.21764609 -0.2601
2496
 -0.27714288 -0.26703389 -0.23078764 -0.17195183 -0.09628583 -0.0111
9651
  0.07498662  0.15382803  0.21760979  0.26008849  0.2771066   0.2669
9761
  0.23075103  0.17191536  0.09624953  0.0111603  -0.07502303 -0.1538
6438
 -0.21764624 -0.26012494 -0.27714293 -0.26703393]
```

```
/Users/soumithvodnala/anaconda/lib/python3.5/site-packages/scipy/opt
imize/slsqp.py:62: ComplexWarning: Casting complex values to real di
scards the imaginary part
  jac[i] = (func(*((x0+dx,)+args)) - f0)/epsilon
/Users/soumithvodnala/anaconda/lib/python3.5/site-packages/scipy/opt
imize/slsqp.py:406: ComplexWarning: Casting complex values to real d
iscards the imaginary part
  slsqp(m, meq, x, xl, xu, fx, c, g, a, acc, majiter, mode, w, jw)
```



The Duffing oscillator

$$\ddot{x} + \dot{x} + x + sin(x) = Acos(2t)$$

In [2]:
```
%matplotlib inline
#From nonlinear.py posted by Daniel Clark
import pylab as pl
import numpy as np
import scipy.optimize as sci
import scipy.integrate as sp

# this code is the nonlinear case of \dotdot{x} + \dot{x} + sin(x) = A
*cos(w*t)

def DuffingOscillatorTimeSeriesResults(N = 3,w = 2,A = 1.2):
    t = np.linspace(0, 10*np.pi, N+1)    # time samples of forcing fun
ction
    t = t[0:-1]                          # Removing the extra sample
    f = A*np.cos(w*t)                      # My forcing function
    T = t[-1]
    xbar = 10*f

    def FUNCTION(xbar):
        N = len(xbar)
        Xbar = np.fft.fft(xbar)
        omega = np.fft.fftfreq(N, T/(2*np.pi*N) )# + 0.0000001 # list
of frequencies
        dotxbar = np.fft.ifft(np.multiply((1j*omega),Xbar))
        dotdotxbar = np.fft.ifft(np.multiply((1j*omega)**2,Xbar))
        R = dotdotxbar + dotxbar + xbar + xbar**3 - f
        R = R**2
        R = np.sum(R)
        return R

    optimizedResults = sci.minimize(FUNCTION, xbar, method='SLSQP')
    xbar = optimizedResults.x

    print(optimizedResults)
    print(xbar)

    pl.plot(t,xbar,t,f)
    pl.legend(['x','Forcing Function'])
    pl.xlabel('Time (s)')
    pl.show()
```

In [55]:
```
DuffingOscillatorTimeSeriesResults(N = 100)
```

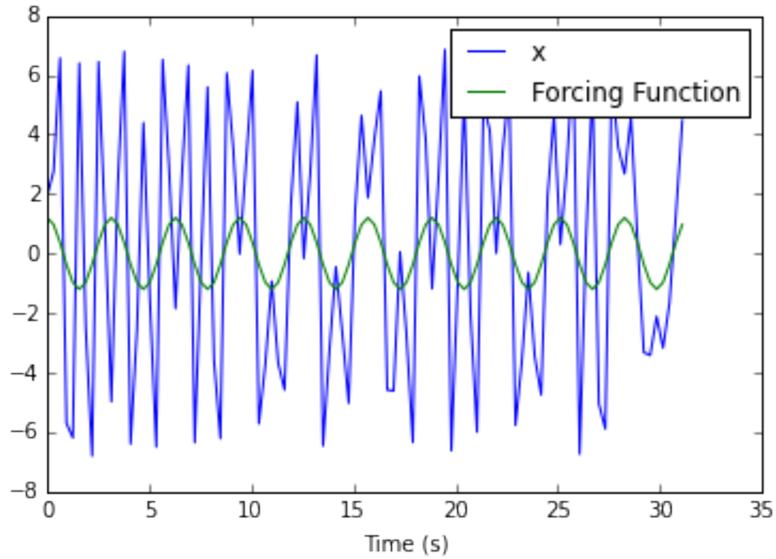
```
 message: 'Iteration limit exceeded'
       x: array([ 1.93231467,  2.761191  ,  6.56007893, -5.69931079,
-6.18589595,
        6.39014429, -2.43811206, -6.79780479,  6.4360562 ,  1.049421
78,
       -4.97578723,  2.27259475,  6.78116385, -6.3970828 , -2.537597
```

```
07,
        4.38919529, -1.49647857, -6.49488355,  6.51123622,  2.858001
51,
       -1.84717195,  2.6175274 ,  6.31776043, -6.35084087, -0.209775
08,
        5.58697419, -3.56354495, -6.21113358,  6.06015409,  3.526388
29,
       -0.01560659,  3.06725075,  6.14298348, -5.71493984, -3.724656
19,
       -0.92800328, -3.66997754, -4.57900394,  1.67260324,  5.082752
59,
       -0.16495957,  2.67374523,  6.6608363 , -6.45740315, -3.245548
95,
       -0.449441  , -2.66062281, -5.02471201,  1.43495415,  4.629689
53,
        1.8734431 ,  3.83741209,  5.45473806, -4.57953635, -4.610333
93,
        0.05355531, -2.70677313, -6.34262689,  5.95604728,  3.827707
11,
       -1.18260163,  2.55618692,  6.84805821, -6.61914357, -0.924394
09,
        5.33069177, -2.06601128, -5.99600655,  5.02065392,  4.124493
56,
        0.00704862,  3.42803705,  5.63993052, -5.7671419 , -3.588640
64,
       -0.63731389, -3.45666529, -4.74614198,  1.92318711,  4.674378
2 ,
        0.30659006,  2.90502672,  6.73157741, -6.73444189,  0.488344
69,
        5.64905455, -5.04895119, -5.88796434,  5.87672073,  3.578220
35,
        2.67748897,  4.5759838 ,  0.51027309, -3.32313041, -3.416845
6 ,
       -2.11545865, -3.16977364, -1.66698476,  1.29716472,  4.468891
7 ])
     nit: 101
     fun: (232169.97653255676+15501.871111197212j)
    njev: 101
     jac: array([ 3117.09765625,   266.61914062,  7954.96875   , -91
66.40039062,
       -6707.15625   , -4153.03125   , -1150.3046875 , -3471.5195312
5,
       -1831.93359375,   475.40234375,  3032.29101562,  -535.0703125
,
        2763.015625  , -2286.49609375, -2063.46289062, -4027.5820312
5,
        2683.3984375 ,  5879.00390625,  7920.29101562,  1733.5253906
2,
        1630.03515625,  -813.91015625, -3935.0234375 ,   766.21875
,
```

```
        1630.3515625 , -2485.20117188,   350.6875    ,  1393.9609375
,
        4059.0390625 ,  1414.06445312,  3169.7265625 ,  -645.4707031
2,
          50.6015625 , -2595.18945312, -1087.4453125 , -1733.6132812
5,
       -2088.9296875 , -2067.3515625 ,  1885.1484375 ,  2578.3085937
5,
        -436.48828125,   656.36132812,  2026.54296875, -7975.5546875
,
       -1080.51953125, -3176.828125  ,  1231.890625  , -3351.4609375
,
        4037.29296875,   944.765625  ,  3791.9296875 ,  2249.4179687
5,
        1681.93359375, -2661.46875   , -1496.765625  , -3447.5
,
        1761.71484375,  -353.59765625,  4055.61328125,  1379.453125
,
        2316.55859375,  -444.6484375 ,  4836.80273438, -3022.4042968
8,
         -48.4375    , -1719.125     ,  2216.6171875 ,  1461.7539062
5,
        -344.62695312,   866.97460938,  1260.484375  ,   662.5742187
5,
       -3728.13085938, -5224.109375  ,  -794.77929688, -1325.9824218
8,
       -1275.70507812, -2418.90625   ,  3090.7265625 ,   223.8925781
2,
        1324.21875   ,   293.81640625,  3042.80664062, -2973.4414062
5,
         612.5546875 , -6928.64257812, -4656.4921875 , -1388.9179687
5,
        4390.60546875,  2994.40820312,  3006.91015625,  2484.2773437
5,
         702.02539062,  -773.60351562, -1768.9140625 ,  1025.1992187
5,
       -1724.30273438,   713.17382812,  -374.59960938,  2581.6855468
8,
           0.        ])
 success: False
    nfev: 10691
  status: 9
[ 1.93231467  2.761191    6.56007893 -5.69931079 -6.18589595  6.3901
4429
 -2.43811206 -6.79780479  6.4360562   1.04942178 -4.97578723  2.2725
9475
  6.78116385 -6.3970828  -2.53759707  4.38919529 -1.49647857 -6.4948
8355
  6.51123622  2.85800151 -1.84717195  2.6175274   6.31776043 -6.3508
4087
```

```
 -0.20977508  5.58697419 -3.56354495 -6.21113358  6.06015409  3.5263
8829
 -0.01560659  3.06725075  6.14298348 -5.71493984 -3.72465619 -0.9280
0328
 -3.66997754 -4.57900394  1.67260324  5.08275259 -0.16495957  2.6737
4523
  6.6608363  -6.45740315 -3.24554895 -0.449441   -2.66062281 -5.0247
1201
  1.43495415  4.62968953  1.8734431   3.83741209  5.45473806 -4.5795
3635
 -4.61033393  0.05355531 -2.70677313 -6.34262689  5.95604728  3.8277
0711
 -1.18260163  2.55618692  6.84805821 -6.61914357 -0.92439409  5.3306
9177
 -2.06601128 -5.99600655  5.02065392  4.12449356  0.00704862  3.4280
3705
  5.63993052 -5.7671419  -3.58864064 -0.63731389 -3.45666529 -4.7461
4198
  1.92318711  4.6743782   0.30659006  2.90502672  6.73157741 -6.7344
4189
  0.48834469  5.64905455 -5.04895119 -5.88796434  5.87672073  3.5782
2035
  2.67748897  4.5759838   0.51027309 -3.32313041 -3.4168456  -2.1154
5865
 -3.16977364 -1.66698476  1.29716472  4.4688917 ]
```

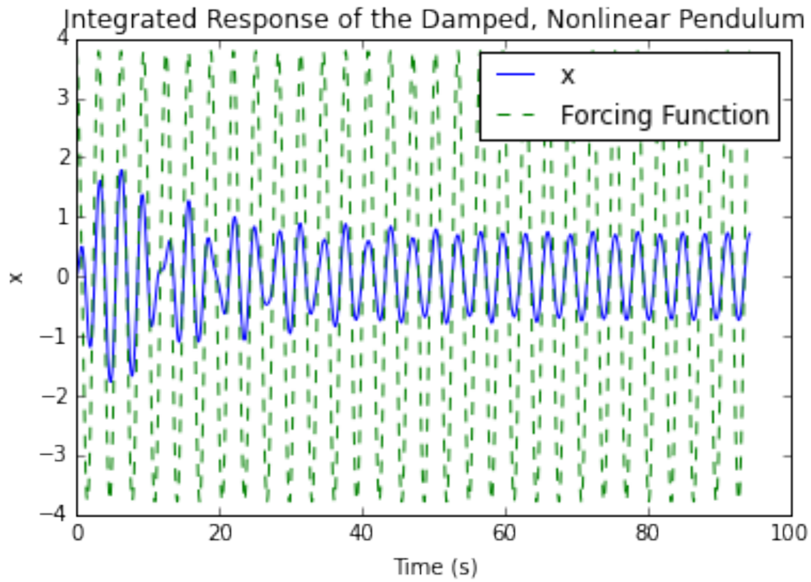
```
/Users/soumithvodnala/anaconda/lib/python3.5/site-packages/scipy/opt
imize/slsqp.py:62: ComplexWarning: Casting complex values to real di
scards the imaginary part
  jac[i] = (func(*((x0+dx,)+args)) - f0)/epsilon
/Users/soumithvodnala/anaconda/lib/python3.5/site-packages/scipy/opt
imize/slsqp.py:406: ComplexWarning: Casting complex values to real d
iscards the imaginary part
  slsqp(m, meq, x, xl, xu, fx, c, g, a, acc, majiter, mode, w, jw)
```



In [56]:
```
#import numpy as np
import scipy as sp
import scipy.integrate
import matplotlib.pyplot as plt

# More plotting stuff
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation

# Needed for sliders that I use.
import IPython.core.display as ipcd
from ipywidgets.widgets.interaction import interact, interactive

# These make vector graphics... higher quaility. If it doesn't work, c
omment these and try the preceeding.
```

In [57]:
```
import numpy as np
import pylab as pl
def solve_sdof(max_time=10.0, g = 9.81,l = 1,m = 1,zeta = 0.1, A = 3.7
8, w = 2, x0 = 0, v0 = 0, plotnow = 1):


    def sdof_deriv(x1_x2, t, g = 9.81,l = 1,m = 1,zeta = 0.1,A = 3.78,
w = 2):
        """Compute the time-derivative of a SDOF system."""
        x1, x2 = x1_x2
        return [x2, -zeta/m/l*x2 - g/l*np.sin(x1) + A*np.cos(w*t)]

    x0i=((x0, v0))
    # Solve for the trajectories
    t = sp.linspace(0, max_time, int(250*max_time))
    x_t = sp.integrate.odeint(sdof_deriv, x0i, t)

    x, v = x_t.T
    f = A*np.cos(w*t)

    if plotnow == 1:
        #fig = plt.figure()
        #ax = fig.add_axes([0, 0, 1, 1], projection='3d')
        plt.plot(t,x,t,f,'--')
        pl.legend(['x','Forcing Function'])
        plt.xlabel('Time (s)')
        plt.ylabel('x')
        plt.title('Integrated Response of the Damped, Nonlinear Pendul
um')
        plt.show()

    return t, x, v
```

```
In [58]: solve_sdof(max_time=3*10*np.pi, x0 = 0, v0 = 0, plotnow = 1)
```




```
Out[58]: (array([  0.00000000e+00,   4.00033020e-03,   8.00066041e-03, ...,
                   9.42397789e+01,   9.42437793e+01,   9.42477796e+01]),
          array([  0.00000000e+00,   3.02402364e-05,   1.20938577e-04, ...,
                   7.21703534e-01,   7.21888299e-01,   7.22029750e-01]),
          array([ 0.        ,  0.01511765,  0.03022593, ...,  0.05160135,
                  0.04077355,  0.02994625]))
```

still working...

```
In [ ]:
```