**Application of the Method of Harmonic Balance**

**Date Due**: December 15, 2015

**Project Contributors:**

Admir Makas

Alec Blankenship

Shashankashekhar Dutta

ME 7160 Non Linear Dynamics and Vibrations

**Course Instructor**:

Dr. Joseph C. Slater

**Wright State University**

Table of Contents

1. Project Overview 1
2. Applied Models and Results 1
   1. Linear System 1
   2. Duffing Oscillator 3
   3. Van der Pol Equation 4
   4. Determining a Closed form Solution 5
3. Conclusions 7
4. Coding 8
5. Appendix 14

# Project Overview

The objective of this study is to utilize the method of harmonic balance (MHB) to find a solution of a harmonically excited system. The procedure involves finding a solution using FFT and comparing it with an analytical numerical solution. Python is used as the primary software platform for coding the solver routines.

The method is programmed using the following sequence

1. Define number of sample points (N).
2. Identify initial guess (1xN-1 matrix), in most cases just 1.
3. Define Fourier transform frequencies given N.
4. Take FFT of initial guess matrix (X).
5. Take IFFT of resultant and to determine derivatives ***ẋ*** and ***ẍ***.
6. Substitute back into governing equation and define a residual function.
7. Use optimization scheme to minimize the residual and determine a one period, steady state solution to the nonlinear system.

It should be noted that using this perturbation method results in a solution that is only obtained over one period and at steady state of the governing system. For this study, four different linear and nonlinear oscillatory systems were reviewed to exhibit the application of the method of Harmonic Balance. The Duffing equation and Van der pol oscillator were among the nonlinear systems studied.

# Applied models and results

The governing models used in this study are described in the following sections. To start, a linear system that has a known solution was used to write the python script that utilized the method of harmonic balance. The program was later adapted for more complex nonlinear systems. All codes used in each of the presented systems are available in the “Coding” section of the report. A sample MATLAB code is also provided in the appendix.

## Linear system

The initial linear system used to develop the initial code is given by the following equation:

The governing equation was found to have a solution of the form:

Using the method of harmonic balance, an approximate solution is found and plotted with respect to the analytical solution determined for this system in order to show the accuracy of the harmonic balance solution. In Fig.1, the method of harmonic balance is shown as dotted points while the analytical solution is the dashed lines.

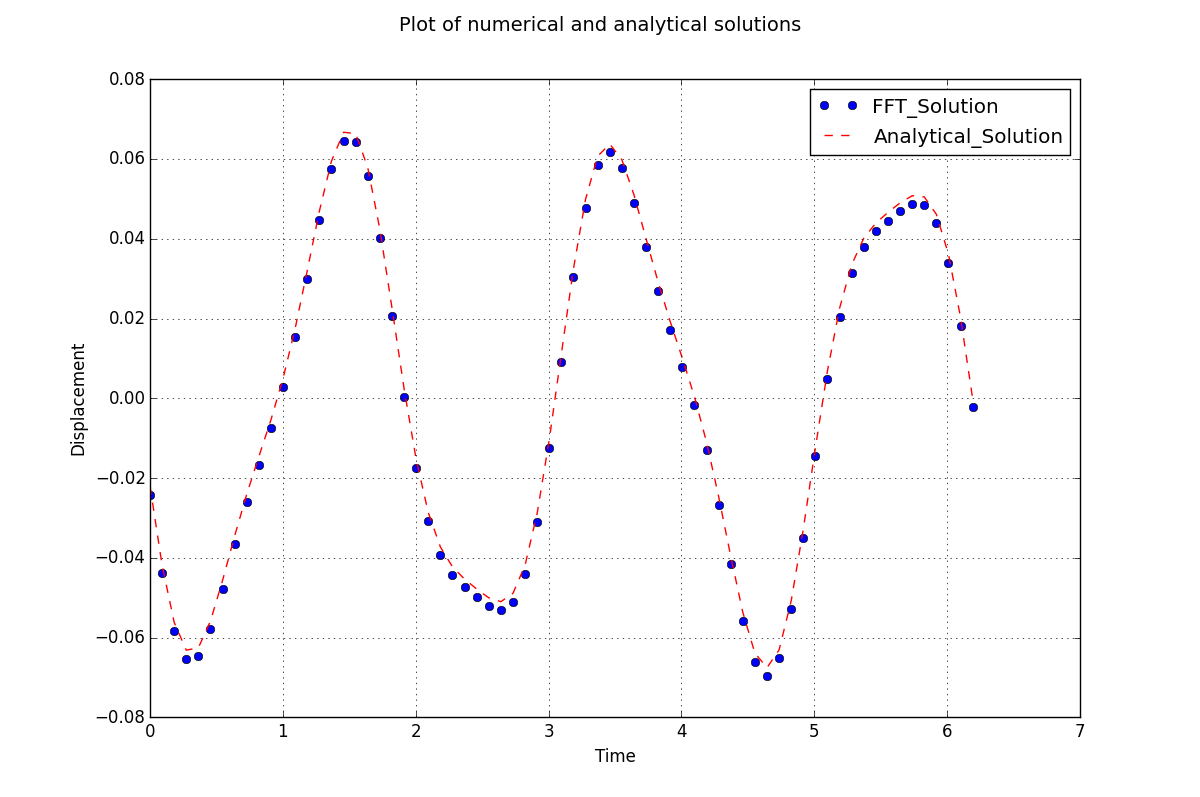


Figure - Method of harmonic balance approximation of linear system

This is a linear model, which has a unique forcing function that causes the complex oscillatory pattern. The MHB approach shows good agreement with the analytical solution with small negligible variability at the peaks throughout the period. Depending on the tolerance for error, this perturbation method can be said to sufficiently approximate the system response. Since this approach works well with the linear system, the code can be applied to more complex systems.

## Duffing Oscillator

The following is an equation for the Duffing oscillator with a unique forcing function. The MHB is used to approximate the system steady state response. Time integrated solution will be plotted with respect to the MHB solution in order to illustrate solver accuracy. The equation is given as:

The solution is plotted in Fig 2. Recall that the solution to the method of harmonic balance is only valid at the steady state condition, hence the shifted x axis values in Fig. 2 for the Duffing oscillator. The plot shows that the MHB approach yields an appropriate solution since only small deviations from the numerical solution are present at the solution peaks.

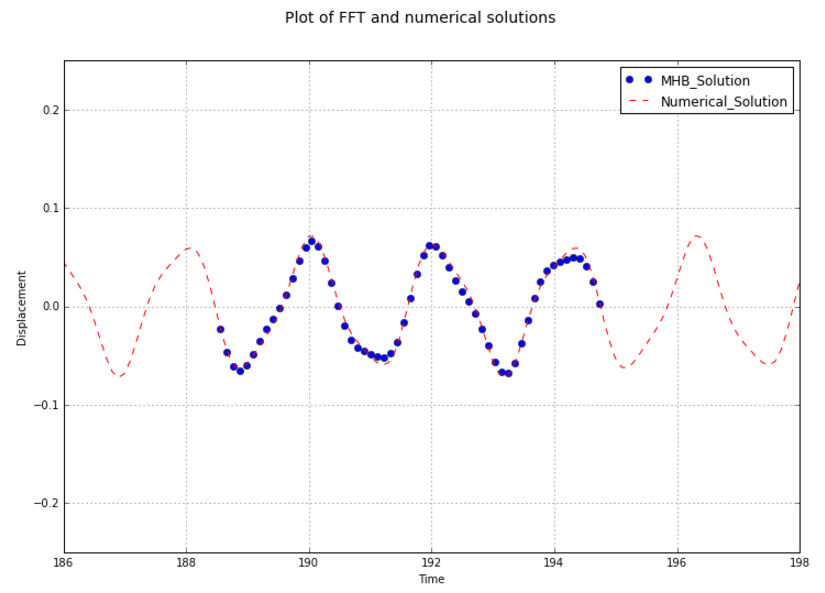


Figure - Method of harmonic balance approximation of the Duffing oscillator

## Van der Pol Equation

Finally, the MHB code was adapted to different variations of the Van der Pol equation. Those variations have different forcing function and are given below.

Variation 1 🡪

Variation 2 🡪 , where F = 2

As was done before, the MHB approach was plotted with the numerical solution in order to determine accuracy. Since this solution has a transient regime, the x axis was shifted such that the MHB was plotted with the steady state regime of the numerical solution. The results for variation 1 and 2 are given in Figs. 3 and 4 respectively.

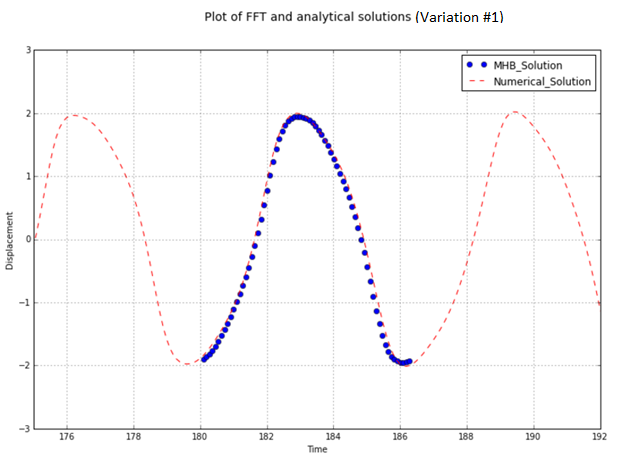
****

Figure -Method of harmonic balance approximation of Van der Pol equation, variation 1

For the first variation of the Van der Pol equation there is a small deviation of the MHB solution from the numerical solution throughout the period. However this may be compensated for at the shift in the peak of the solution at the beginning of the period. Better initial guess, which is fed into the optimization function could help get a better result.

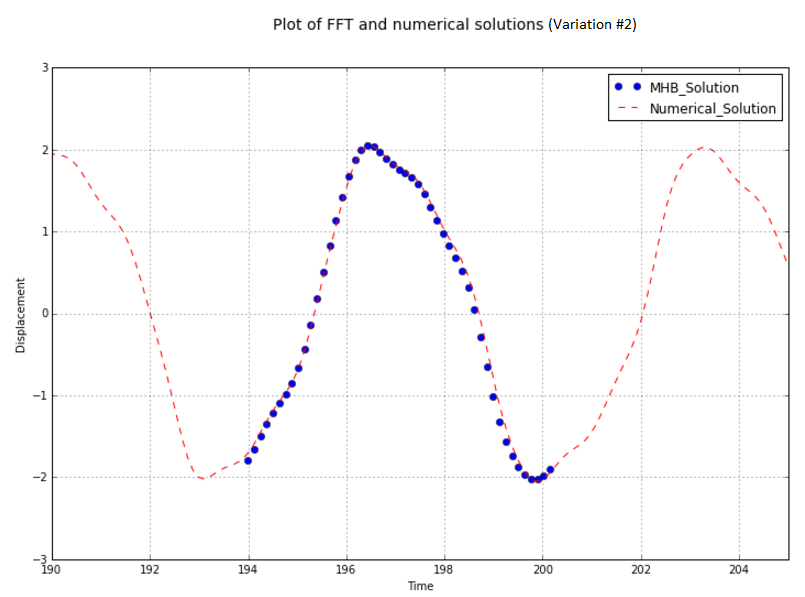
****

Figure -Method of harmonic balance approximation of Van der Pol equation, variation 2

FFT solution for the second variation yields a good results and matches well with the time integration technique. Small deviations could be improved by using a better initial guess. Additionally, there may still be a small transient component in the time integration solution that has is not completely dissipated. Harmonic balance method as implemented in this project is somewhat sensitive to the initial guess for. The user may need to provide multiple initial guesses before the optimizer yields a successful results. As a general procedure it is always recommended to start by using a vector of 1’s or 0’s and move to sinusoidal initial guess when these fail.

## Determining a Closed Form Solution

Additional goal of the project is to use optimized results and attempt to generate an approximate function for the solution that can further be of use for additional analysis. There are a few ways to accomplish this feat. While the optimized Fourier coefficients can be used to generate a function, it was desirable to use other python methods in order to facilitate additional learning. Method used below is a fitting scheme that uses time domain **x** in order to generate the complementary function using a Fourier cosine series. Method below is crude but it can be easily optimized for better usability.

Assumed solution has 8 terms:

Here, AN, ω, and β are the unknowns and will be solved for using least square fitting. For illustration purposes, an approximate closed solution for the second variation of the Van der Pol equation will be given. However, any of the governing models presented could also be used. The following procedure was used to obtain the solution:

1. Define residual function to be minimized using the least squares function. The function must find the difference between the results from the MHB and the assumed solution form.
2. Make initial guess for the closed form solution coefficients, in this case 1.
3. Run the least squares optimization scheme to minimize the error of the residual function
4. Simplify closed form and plot solution to check accuracy

This is a way to develop an approximate solution to the governing parameters. It can be a useful tool given that the fitted solution provides an acceptable approximation.

Fig. 5 gives the numerical result plotted against the approximated closed form solution.

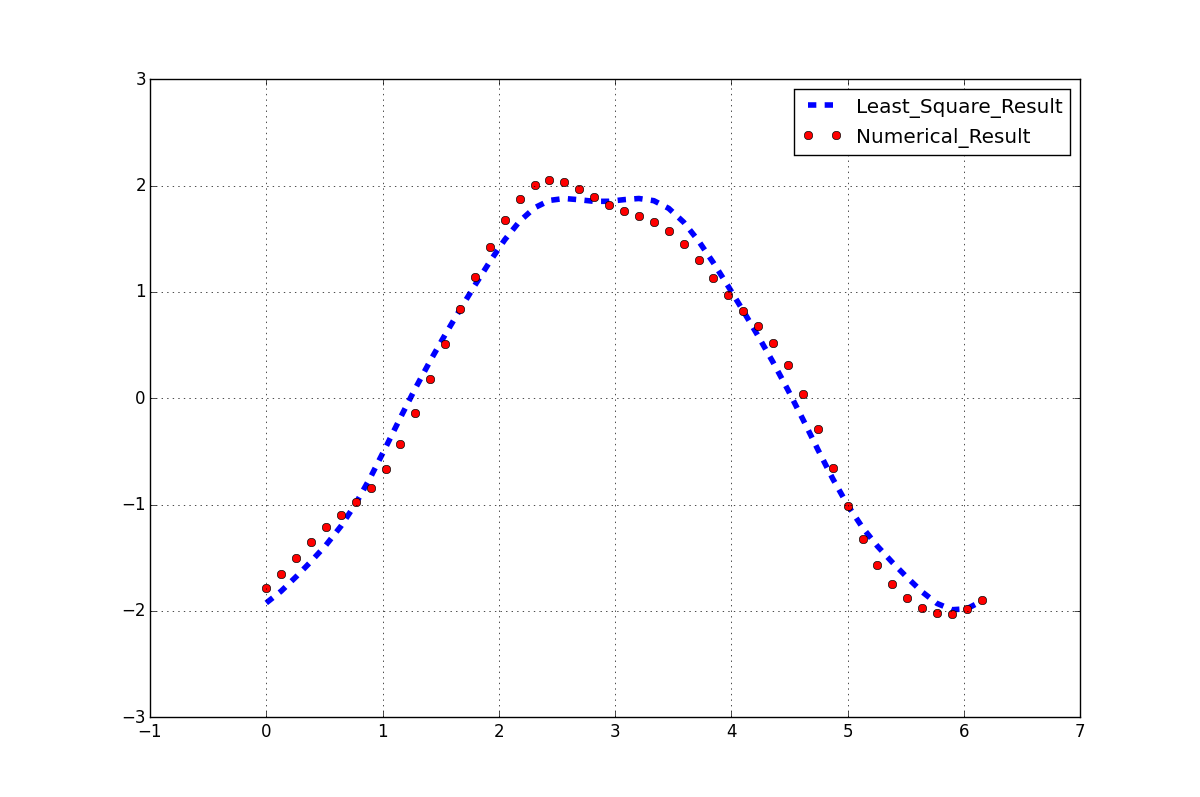
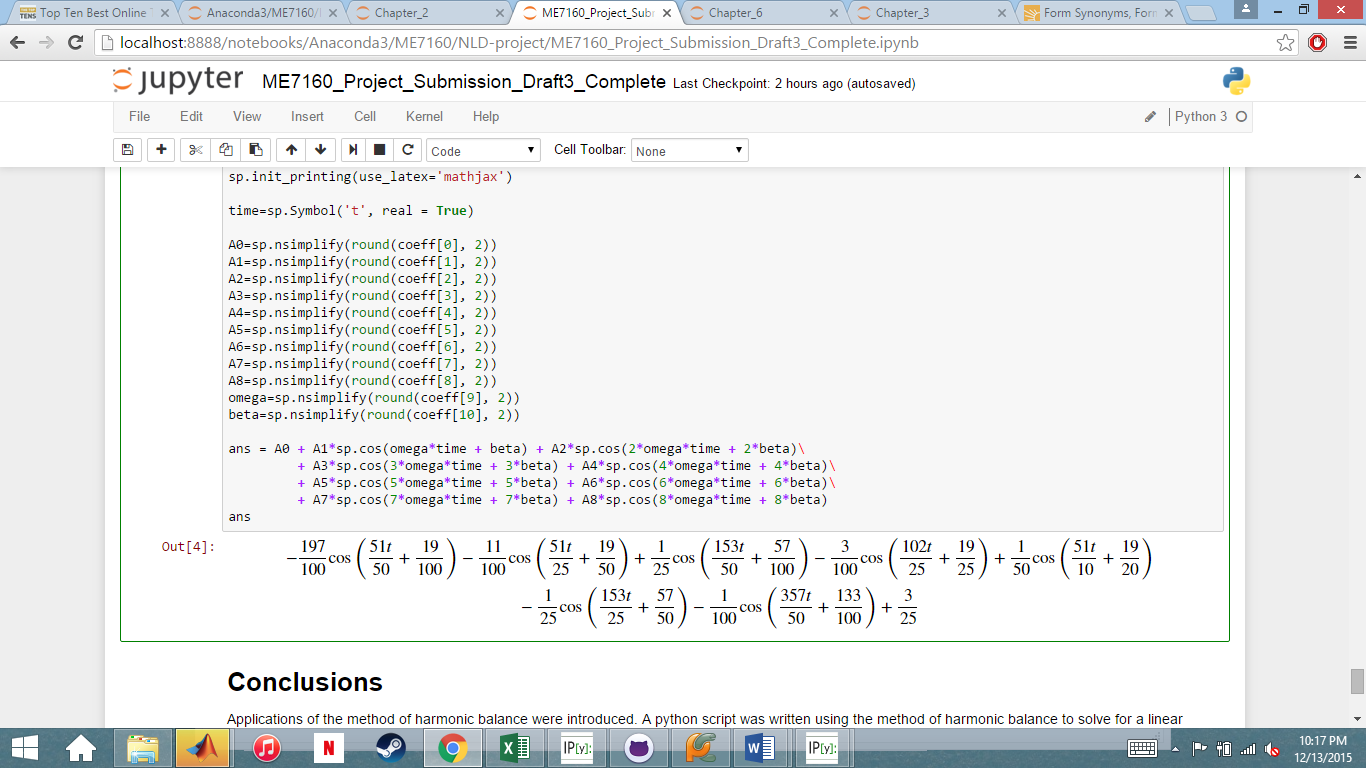


Figure -Least squares solution for an approximate closed solution for the Van der Pol equation, variation 2

The assumed closed form solution gives us a fairly close approximation. There are several deviations from the numerical solution but the fit may be sufficient depending on the desired error tolerance. Other forms of the solution including exponential and polynomial can be assumed and may provide a better fit depending on the governing system. The approximated closed form equation for the MHB solution is presented below.



**3.0 Conclusions**

Applications of the method of harmonic balance were introduced. A python script was written using the method of harmonic balance to solve for a linear system. Once accomplished, complex nonlinearities were introduced into other governing systems and the method was used to approximate those solutions. The approximations given by the method of harmonic balance are valid at the steady state for the system and over one period. The only limitation encountered was that for more complex forcing functions, the accuracy depended on the quality of the guess.

The Duffing oscillator and Van der Pol equation were the nonlinear systems used in this study. The approximations for both systems showed comparable levels of fidelity with respect to time integration results, which further validates this method. The accuracy of each approximation was confirmed by plotting against the time integration solution. Based on the accuracy of each approximation it can be stated that the method of harmonic balance was successfully applied to each of the presented linear and nonlinear systems.

Finally, a cosine Fourier series fit for the second variation of the Van der Pol equation was approximated by using the least squares method. This was done to show that the method of harmonic balance can be used to develop an approximate closed form model to describe a wide range of nonlinear systems.

# PYTHON Coding

## Linear System

import numpy as np

import matplotlib.pyplot as plt

import sympy as sp

N = 70 #define number of sample points

t=np.linspace(0, 2\*np.pi, N)

t=t[0:-1]

x0=np.ones(N-1)

#x0=np.cos(2\*t)

f = np.cos(2\*t)\*np.sin(5\*t)

freq = np.fft.fftfreq(N-1, 1/(N-1))+.000000001 #define Fourier Transform sample frequencies

# Function resFun defines a residual term that will be used in the optimization operation

def resFun(x):

X = np.fft.fft(x)

xddot = np.fft.ifft(-freq\*\*2\*X)

xdot = np.fft.ifft(1j\*freq\*X)

res = xddot + xdot + x - f

RES = np.sum(np.abs(res\*\*2))

return RES

# Goal of the function is to minimize error between solution guess x0 and response in order to

# converge to a solution

from scipy.optimize import minimize

sol = minimize(resFun, x0, method = 'BFGS', options={'maxiter':50000, 'disp':True})

#sol = minimize(resFun, x0)

print('Values of x after optimization:')

print(sol.x)

#Optional to plot Jacobian of the minimize function

#print(sol.jac)

A = (-3785/171769)

B = (-3274/171769)

C = (-11164/171769)

D = (7660/171769)

Analytical = (A)\*np.cos(2\*t)\*np.cos(5\*t) + (B)\*np.sin(2\*t)\*np.sin(5\*t) + (C)\*np.cos(2\*t)\*np.sin(5\*t)\

+ (D)\*np.sin(2\*t)\*np.cos(5\*t)

#Code for Plotting numerical and analytical solutions not shown

## Duffing Oscillator

import numpy as np

import matplotlib.pyplot as plt

import sympy as sp

N = 70 #define number of sample points

t=np.linspace(0, 2\*np.pi, N)

t = t[0:-1]

x0=np.ones(N-1)

#x0=np.cos(2\*t)

f = np.cos(2\*t)\*np.sin(5\*t)

freq = np.fft.fftfreq(N-1, 1/(N-1))+.000000001 #define Fourier Transform sample frequencies

# Function resFun defines a residual term that will be used in the optimization operation

def resFun(x):

X = np.fft.fft(x)

xddot = np.fft.ifft(-freq\*\*2\*X)

xdot = np.fft.ifft(1j\*freq\*X)

res = xddot + xdot + x + x\*\*3 - f

RES = np.sum(np.abs(res\*\*2))

return RES

# Goal of the function is to minimize error between solution guess x0 and response in order to

# converge to a solution

from scipy.optimize import minimize

sol = minimize(resFun, x0, method = 'BFGS', options={'maxiter':50000, 'disp':True})

#sol = minimize(resFun, x0)

print('Values of x after optimization:')

print(sol.x)

#Optional to plot Jacobian of the minimize function

#print(sol.jac)

#Numerical solution for comparison

from scipy.integrate import odeint

def deriv(x, t):

return np.array([x[1], -0.25\*x[1] - x[0] - x[0]\*\*3 + np.cos(2\*t)\*np.sin(5\*t)])

time = np.linspace(0.0, 100, 2000)

xinit=np.array([0,0])

x = odeint(deriv, xinit, time)

#Plotting of numerical and MHB solutions are not shown

## Van der Pol Variation 1

import numpy as np

import matplotlib.pyplot as plt

import sympy as sp

N = 70 #define number of sample points

t=np.linspace(0, 2\*np.pi, N)

t=t[0:-1]

F = 1

#x0=1.5\*np.ones(N-1)

#x0=1.48\*np.cos(1\*t)

x0=3.0\*np.sin(4\*t)\*np.cos(1\*t)

f = F\*np.cos(1\*t)\*np.sin(4\*t)

freq = np.fft.fftfreq(N-1, 1/(N-1))+.00000000001 #define Fourier Transform sample frequencies

# Function resFun defines a residual term that will be used in the optimization operation

def resFun(x):

X = np.fft.fft(x)

xddot = np.fft.ifft(-freq\*\*2\*X)

xdot = np.fft.ifft(1j\*freq\*X)

res = xddot + (x\*\*2 - 1)\*xdot + x - f

RES = np.sum(np.abs(res\*\*2))

return RES

# Goal of the function is to minimize error between solution guess x0 and response in order to

# converge to a solution

from scipy.optimize import minimize

sol = minimize(resFun, x0, method = 'BFGS', options={'maxiter':50000, 'disp':True})

#sol = minimize(resFun, x0)

print('Values of x after optimization:')

print(sol.x)

#Optional to plot Jacobian of the minimize function

#print(sol.jac)

#Numerical solution

from scipy.integrate import odeint

def deriv(x,t):

return np.array([x[1], -(x[0]\*\*2 - 1)\*x[1] - x[0] + F\*np.cos(1\*t)\*np.sin(4\*t)])

time=np.linspace(0.0,100,2000)

xinit=np.array([-2, 0])

x=odeint(deriv, xinit, time)

#Plotting of numerical and MHB solutions not shown

## Van der Pol Variation 2

import numpy as np

import matplotlib.pyplot as plt

import sympy as sp

N = 50 #define number of sample points

t=np.linspace(0, 2\*np.pi, N)

t=t[0:-1]

F = 2

#x0=1\*np.ones(N-1)

x0=1\*np.cos((2\*np.pi\*t)/1)

#x0=2.0\*np.sin(1\*t)\*np.cos(1\*t)

f = F\*np.cos((2\*np.pi\*t))

freq = np.fft.fftfreq(N-1, 1/(N-1))+.00000000001 #define Fourier Transform sample frequencies

# Function resFun defines a residual term that will be used in the optimization operation

def resFun(x):

X = np.fft.fft(x)

xddot = np.fft.ifft(-freq\*\*2\*X)

xdot = np.fft.ifft(1j\*freq\*X)

res = xddot + (x\*\*2 - 1)\*xdot + x - f

RES = np.sum(np.abs(res\*\*2))

return RES

# Goal of the function is to minimize error between solution guess x0 and response in order to

# converge to a solution

from scipy.optimize import minimize

sol = minimize(resFun, x0, method = 'BFGS', options={'maxiter':50000, 'disp':True})

#sol = minimize(resFun, x0)

print('Values of x after optimization:')

print(sol.x)

#Optional to plot Jacobian of the minimize function

#print(sol.jac)

#Numerical solution

from scipy.integrate import odeint

def deriv(x,t):

return np.array([x[1], -(x[0]\*\*2 - 1)\*x[1] - x[0] + F\*np.cos((2\*np.pi\*t))])

time=np.linspace(0.0,100,2000)

xinit=np.array([-2.0, 0])

x=odeint(deriv, xinit, time)

#Plotting of numerical and MHB solutions

## Closed Form Approximations

#Define function to be called in the "leastsq" function

#sol.x is the method of harmonic balance solution

def errorFun(p, t):

A0, A1, A2, A3, A4, A5, A6, A7, A8, omega, beta = p

err = sol.x - (A0 + A1\*np.cos(omega\*t + beta) + A2\*np.cos(2\*omega\*t + 2\*beta)\

+ A3\*np.cos(3\*omega\*t + 3\*beta) + A4\*np.cos(4\*omega\*t + 4\*beta)\

+ A5\*np.cos(5\*omega\*t + 5\*beta) + A6\*np.cos(6\*omega\*t + 6\*beta)\

+ A7\*np.cos(7\*omega\*t + 7\*beta) + A8\*np.cos(8\*omega\*t + 8\*beta))

return err

#Initial guess for coefficients

p0 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

#print(np.array(p0))

#Least square calculation

from scipy.optimize import leastsq

plsq = leastsq(errorFun, p0, args=(t))

#Function used for plotting purposes

def peval(t, p):

return p[0] + p[1]\*np.cos(p[9]\*t + p[10]) + p[2]\*np.cos(2\*p[9]\*t + 2\*p[10])\

+ p[3]\*np.cos(3\*p[9]\*t + 3\*p[10]) + p[4]\*np.cos(4\*p[9]\*t + 4\*p[10])\

+ p[5]\*np.cos(5\*p[9]\*t + 5\*p[10]) + p[6]\*np.cos(6\*p[9]\*t + 6\*p[10])\

+ p[7]\*np.cos(7\*p[9]\*t + 7\*p[10]) + p[8]\*np.cos(8\*p[9]\*t + 8\*p[10])

#Plot numerical and least square results

import matplotlib.pyplot as plt

fig = plt.figure(figsize=(12,8))

plt.plot(t, peval(t, plsq[0]), 'b--', linewidth = 4, label = 'Least\_Square\_Result')

plt.plot(t, sol.x, 'ro', label = 'Numerical\_Result')

plt.legend(loc = 'upper right')

plt.axis([-1, 7, -3, 3])

plt.grid()

plt.show()

coeff = plsq[0]

#Plot calculated results the coefficients

print('Coefficient values listed below')

print(plsq[0])

import sympy as sp

sp.init\_printing(use\_latex='mathjax')

time=sp.Symbol('t', real = True)

A0=sp.nsimplify(round(coeff[0], 2))

A1=sp.nsimplify(round(coeff[1], 2))

A2=sp.nsimplify(round(coeff[2], 2))

A3=sp.nsimplify(round(coeff[3], 2))

A4=sp.nsimplify(round(coeff[4], 2))

A5=sp.nsimplify(round(coeff[5], 2))

A6=sp.nsimplify(round(coeff[6], 2))

A7=sp.nsimplify(round(coeff[7], 2))

A8=sp.nsimplify(round(coeff[8], 2))

omega=sp.nsimplify(round(coeff[9], 2))

beta=sp.nsimplify(round(coeff[10], 2))

ans = A0 + A1\*sp.cos(omega\*time + beta) + A2\*sp.cos(2\*omega\*time + 2\*beta)\

+ A3\*sp.cos(3\*omega\*time + 3\*beta) + A4\*sp.cos(4\*omega\*time + 4\*beta)\

+ A5\*sp.cos(5\*omega\*time + 5\*beta) + A6\*sp.cos(6\*omega\*time + 6\*beta)\

+ A7\*sp.cos(7\*omega\*time + 7\*beta) + A8\*sp.cos(8\*omega\*time + 8\*beta)

# APPENDIX

# MATLAB FIle for duffing oscillator

## Define number of sample points

N = 100;  
t = linspace(0,2\*pi,N)';  
t(end) = [];

## [Defining forcing function](Defining)

%x"+x'+x+x^3=cos(2t)sin(5t)  
global func freq %Defining global variables  
func = cos(2.\*t).\*sin(5.\*t);

## Frequency Spectrum

dt = 1/N;  
df=1./(N\*dt); % Interval in freq  
fpos=[0:N/2-1]'\*df; % Freqs of positive half of spectrum  
fneg=[N/2-1:-df:1]'\*df\*-1; %Freqs of negative half of spectrum  
freq =[fpos;fneg]; %Full frequency spectrum

## [Initial solution for x](Guessing)

x0 = ones(N-1,1); %x=1

## [Optimization toolbox to minimize residual and obtain x solution](Using)

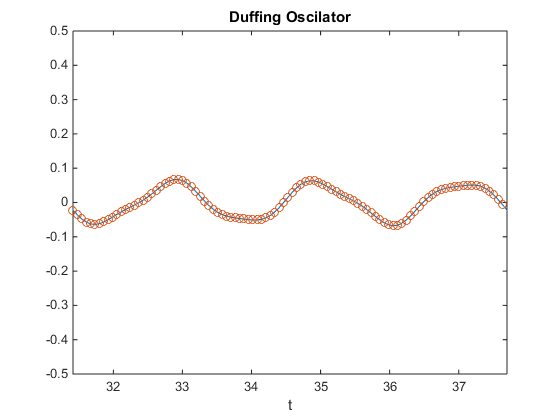
%See associated function 'resFUN\_Duff.m'  
options = optimset('MaxFunEvals',50000);  
[x\_HarBal,RES] = fminunc(@resFUN\_Duff,x0,options);

## Using ODE45 to numerically solve for the ode

[t\_num,x\_num]=ode45('derivFUN\_Duff',[0 100],[0 0])

## Solution with numerical solution

t\_shift = t+(2\*pi\*5);  
figure  
plot(t\_num,x\_num(:,1),t\_shift,x\_HarBal,'o')  
axis([2\*pi\*5 2\*pi\*6 -0.5 0.5]) %Shifting to steady state of solution  
xlabel('t')  
title('Duffing Oscilator')



**Function Files**

%Function written for ODE45  
function xp = derivFUN\_Duff(t,x)  
xp = zeros(2,1);  
xp(1) = x(2);  
xp(2) = -x(2)-x(1)-x(1)^3 + cos(2\*t).\*sin(5\*t);

%Defining residual function for minimization scheme  
function RES = resFUN\_Duff(x)  
% <Importing global variables>  
 global freq  
 global func  
% <FFT for given x to solve for derivatives xddot and xdot>  
 X = fft(x);  
 xdot = ifft(1i\*freq.\*X);  
 xddot = ifft(-freq.^2.\*X);  
  
% <Identifying residual function>  
 res = xddot+xdot+x+x.^3-func;  
 RES = sum(abs(res.^2));

[*Published with MATLAB® R2014b*](http://www.mathworks.com/products/matlab)