An Elementary Continuation Technique

Anindya Chatterjee

September 9, 2002

1 Introduction

Let us begin with the differential equation

$$\ddot{x} + c\dot{x} + x + ax^3 + bx^5 = F \sin \omega t \,, \tag{1}$$

which we study using harmonic balance. Here, F, a, b and c are parameters, assumed known.

Assuming $x = A \sin \omega t + B \cos \omega t$, and applying harmonic balance, we obtain the two equations

$$-A\omega^2 + \frac{5}{8}bA^5 + \frac{3}{4}aA^3 + A + \frac{5}{4}bA^3B^2 + \frac{5}{8}bAB^4 + \frac{3}{4}aAB^2 - cB\omega - F \quad = \quad 0 \,, \tag{2}$$

$$\frac{3}{4}aB^3 + \frac{5}{8}bB^5 - B\omega^2 + cA\omega + \frac{3}{4}aA^2B + \frac{5}{4}bA^2B^3 + \frac{5}{8}bA^4B + B \quad = \quad 0 \,. \tag{3}$$

The above equations are hard to solve in general, but for $\omega = 0$ we find that a solution (and the one of interest) is given by $B = 0$ and $F = A + \frac{3}{4}aA^3 + \frac{5}{8}bA^5$. The latter equation can be solved for A in terms of F, or vice versa.

Given $\omega \neq 0$, we can hope to solve Eqs. 2 and 3 numerically, such as by the Newton-Raphson method. Slowly incrementing ω one step at a time, and in each case using the last solution obtained as an initial guess for the present one, we can find several solutions for several values of ω. The method runs into trouble if the solution turns around: incrementing ω beyond the turning point, we find no more solutions.

A simple way around this problem is as follows (arc-length based continuation). We define the vector

$$x = \left\{ \begin{array}{c} A \\ B \end{array} \right\}$$

and write Eqs. 2 and 3 abstractly as

$$f(x, \omega) = 0 \,.$$

In the above, the scalar quantity ω appears as a parameter. As ω is varied, the roots of the above equation trace out a curve in x-space.

Now we extend the vector x and write

$$y = \left\{ \begin{array}{c} A \\ B \\ \omega \end{array} \right\} .$$

We also introduce a vector y_1 and a reasonably small "arc-length" s, whose meaning and purpose will be clear shortly. Now, to Eqs. 2 and 3, we add on a third equation,

$$\|y_1 - y\| - s = 0 \,. \tag{4}$$

Now there are three equations (Eqs. 2, 3 and 4) for three unknowns (the three elements of y). We can solve these using the Newton-Raphson method. At each stage, starting with a "previous" solution y_1, we find a new solution y_2. Then we set y_1 equal to the newly obtained y_2, and repeat the process.

For greater robustness, we can supply two previous values y_0 and y_1. The advantage is that we can then use the linearly extrapolated $2y_1 - y_0$ as a useful initial guess for the Newton-Raphson method.

2 Some Matlab routines, and what they do

Here I provide for you some Matlab routines that I have written. You could copy these, or write your own, I don't particularly care which.

2.1 tempfile

Here, in the `typewriter font`, is a file that takes a global paremeter called μ, which for our specific case is identified with ω. It evaluates the left hand sides of Eqs. 3 and 4. If, for the value of ω set globally through μ, the choice of A and B is correct, then this function will return zero.

```
function z = tempfile(y)
global mu
omega=mu;

%parameters
c=0.1;
a=1.75; b=0.3;
F=0.5;

A=y(1); B=y(2);

z=[-A*omega^2+5/8*b*A^5+3/4*a*A^3-F+A+5/4*b*A^3*B^2+5/8*b*A*B^4+3/4*a*A*B^2-c*B*omega;
   3/4*a*B^3+5/8*b*B^5-B*omega^2+c*A*omega+3/4*a*A^2*B+5/4*b*A^2*B^3+5/8*b*A^4*B+B];
```

2.2 branch_follow

This file takes in initial data (the points y_0 and y_1 alluded to earlier, but split up as μ_0, μ_1, x_0, x_1) as well as a file name ("fname"), and computes a series of solution points. It calls another routine called "newton", which will in turn call "branch_aux" – these are supplied in subsequent subsections.

```
function x=branch_follow(fname,nsteps,mu0,mu1,x0,x1)
% The aim of this program is to follow solution branches to systems of nonlinear
% equations with one free parameter.

% You start with a file fname. This file uses a global parameter (called mu).
% Once mu is defined, given x, this file returns f(x). Note: x is n-dimensional.
% In the space of x, there  is a one-parameter family (or curve) of solutions
% parameterized by mu. Two nearby points on this curve (mu0,x0) and (mu1,x1) are
% initially specified (found manually with trial and error). Our aim is to generate
% a sequence of points (mu2,x2), (mu3,x3) etc. The distance between each point
% and the next one is to be kept fixed as we move along the curve. Some auxiliary
% files will be needed, and called as appropriate.

global mu tracking_file_name xc arc
tracking_file_name=fname;

x0=[mu0;x0];
xc=[mu1;x1];   % extended x, increasing the number of equations by 1.
x=[x0,xc];
arc=norm(x0-xc);
k=1;
c=1;
while (k<nsteps)*c
   xg=2*xc-x0;  % extrapolate
   [xx,c]=newton('branch_aux',xg);
   if c
      k=k+1
      x0=xc;
      xc=xx;
      x=[x,xx];
   end
end
```

2.3 newton

```
function [x,c]=newton(fun,x,showx)
% numerically estimates derivatives and implements Newton's method
% attempts to solve nonlinear equations
n=length(x);
epsil=(1e-5*max(1,norm(x)));
pert=eye(n)*epsil;
iter=0;
nmax=60;
c=1;

ee=feval(fun,x);
while (norm(ee)*max(1,norm(x))>1e-10)*(iter<nmax)

  iter=iter+1;
  for k=1:n
  D(:,k)=(feval(fun,x+pert(:,k))-ee)/epsil;
  end

  x=x-(D\ee);
  if nargin == 3
    disp(x)
  end
  ee=feval(fun,x);

end

disp(iter), disp('iterations, that took')
if (iter == nmax)+(abs(x)==inf)
c=0;
disp('did not converge')
end
```

2.4 branch_aux

```
function y=branch_aux(x)

global mu tracking_file_name xc arc

mu=x(1);
x=x(2:end);
y=feval(tracking_file_name,x);
y=[norm([mu;x]-xc)-arc;y];
```

3 Working with Matlab

In the Matlab environment, I first type "diary somefilename.txt". Then whatever I type in afterwards, as well as whatever Matlab gives me, goes into a file of that name. My diary file, demonstrating the continuation technique, appears below. The first few lines are inputs that I typed at the ">>" prompt. Notice that several commands can be typed into Matlab on the same line. At the end of it all, I also have an `eps` file called "conti.eps" – the figure is provided in the next section.

```
global mu
format compact
mu0=0; mu1=0.02;
mu=mu0; x0=newton('tempfile',[1;0]);
```

(some Matlab output removed from here)

```
mu=mu1; x1=newton('tempfile',x0);
```

(some Matlab output removed from here)

```
X=branch_follow('tempfile',650,mu0,mu1,x0,x1);
```

(what follows is Matlab output)

```
     2
iterations, that took
k =
     2
     2
iterations, that took
k =
     3
     2
iterations, that took
k =
     4
     2
iterations, that took
k =
     5
     2
```

(some Matlab output removed from here; k goes all the way up to 650)

```
k =
   647
     2
iterations, that took
k =
   648
     2
iterations, that took
k =
   649
     2
iterations, that took
k =
   650
```

(Matlab output ends here; input commands follow)

```
plot(X(1,:),sqrt(X(2,:).^2+X(3,:).^2));
xlabel('\omega','fontsize',18), ylabel('Amplitude','fontsize',18)
axis([0,5,0,2])
print -deps conti.eps
diary off
```

4 Results

The results obtained (amplitude versus forcing frequency) for $F = 0.5$, $a = 1.75$, $b = 0.3$ and $c = 0.1$ are shown in the figure.

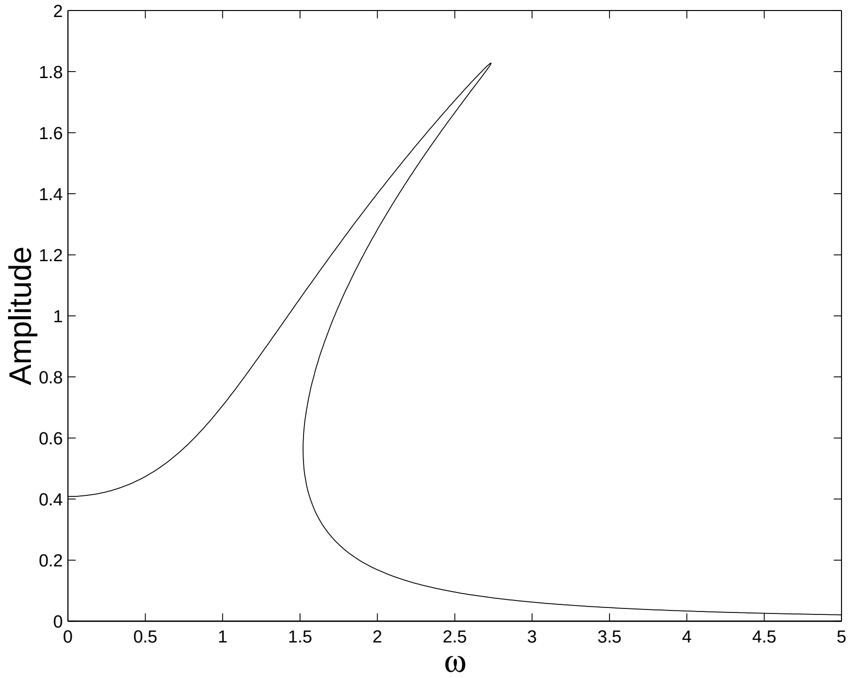

Figure 1: Amplitude versus forcing frequency.