2 SOLVING EQUATIONS

Contents

Initialization

Things to do before an interactive section. (Emacs users also set C-x f fill column to 57.)

```
% reduce needless whitespace
format compact
% reduce irritations
more off
% start a diary
%diary lecture?.txt
```

THE PROBLEM WE WANT TO SOLVE

We want to find the frequencies (tones) of a string with one end rigidly attached and the other end flexibly attached.
It can be shown that all valid frequencies omega must satisfy the equation

$$-k\omega = \tan(\omega)$$

Here k is a constant depending on the string properties. The above equation does not have an analytic solution (unlike a quadratic equation, say.)
Our problem is to figure out what those valid frequencies are using Matlab function `fzero`. We will also learn how to plot functions using `plot`.

FUNCTIONS DESCRIBING ERRORS IN EQUATIONS

If we should have that

$$-k\omega = \tan(\omega)$$

then

$$\text{Error}(\omega) = \tan(\omega) + k\omega$$

is the error in the equation. We can put that in a Matlab function file. For now, we will assume that k equals 1. For that reason, we will call the function `freqEq1Error`. Its contents are:

```
function error = freqEq1Error(omega)

% This function returns the error in the equation
% satisfied by the frequencies of a string with one end
% flexibly attached.  The scaled attachment flexibility k
% is assumed to be 1.
%
% Input:
```

```
%    omega: the frequency to test, in radians
% Output:
%    error: zero if omega is a correct frequency (tone)
%           of the string, nonzero if it is not.
%
% Advanced analysis taught in Analysis in Mechanical
% Engineering II shows that the equation the frequencies
% must satisfy is:
%                 - k omega = tan(omega)
% So if the frequency is not right, the error in the
% equation (difference between the right and left hand
% sides) is:
%               error = tan(omega) + k omega

% Note that omega is in radians and do not forget the
   semi-colon
error = tan(omega) + omega;

end
```

Play a bit with the function

```
% see whether matlab can see the function
%help freqEq1Error

% for omega=0 the error is zero but then there is no
   sound!
err0=freqEq1Error(0)

% for omega=1 the error is not zero, so omega=1 is _not_
% a frequency of vibration of this string
err1=freqEq1Error(1)

% how about 2?
err2=freqEq1Error(2)

% how about 1.9 or 2.1?
err1p9=freqEq1Error(1.9)
err2p1=freqEq1Error(2.1)
```

```
err0 = 0
err1 =  2.5574
err2 = -0.18504
err1p9 = -1.0271
err2p1 =  0.39015
```

PLOT TO UNDERSTAND THE PROBLEM BETTER

Somehow we must find the locations where the error is zero. That is not that straightforward. So maybe we should first examine the functions in the right and left hand sides by plotting them.

Plot the error for 0 < omega < 10

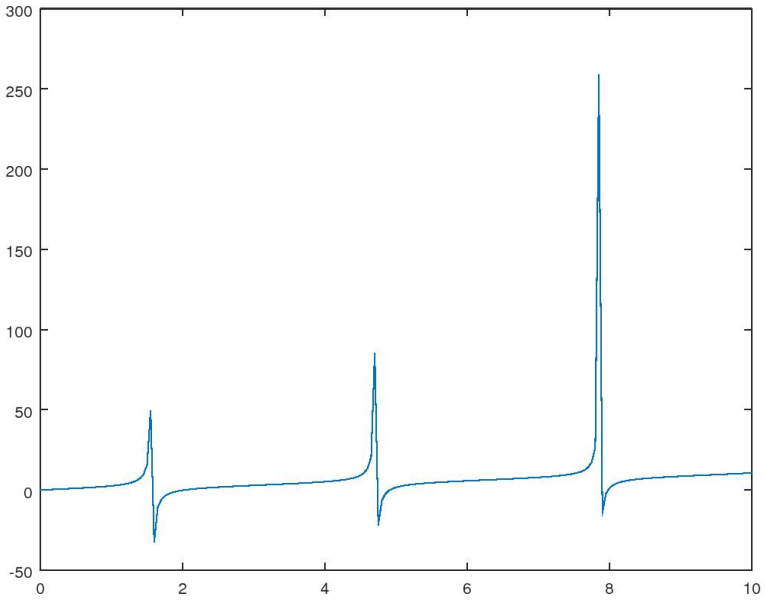
The Matlab 'plot' function plots curves given enough points on the curves.

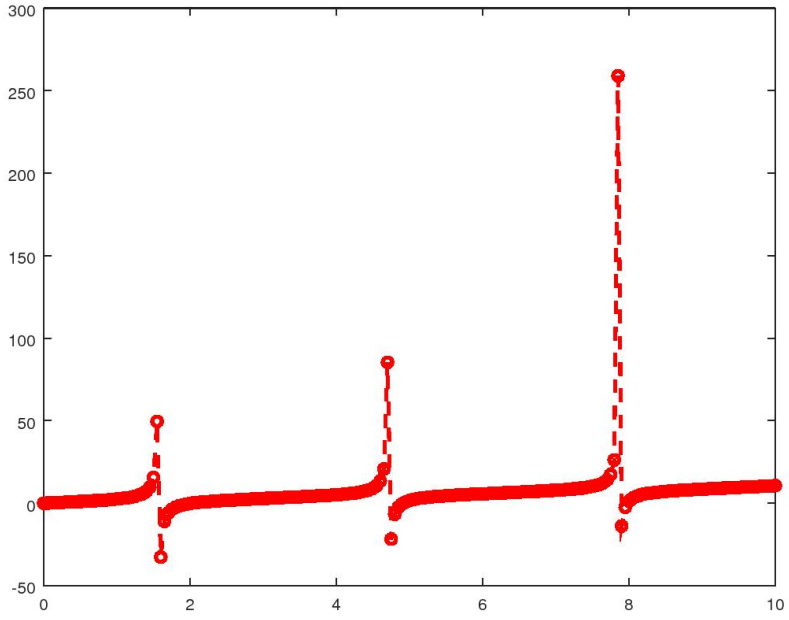
```
% generate 201 omega values between 0 and 10
omegaVals=[0:0.05:10];

% this makes omegaVals a row of numbers
%omegaVals

% another way to do the same thing
omegaVals=linspace(0,10,201);
%omegaVals

% compute the corresponding errors
errorVals=freqEq1Error(omegaVals);
%errorVals

% plot (unmaximize this window to have the plot visible)
plot(omegaVals',errorVals')

% Note: The reason for the quotes is that plot likes to
% have its numbers as columns instead as rows.
```

Try improving the plot

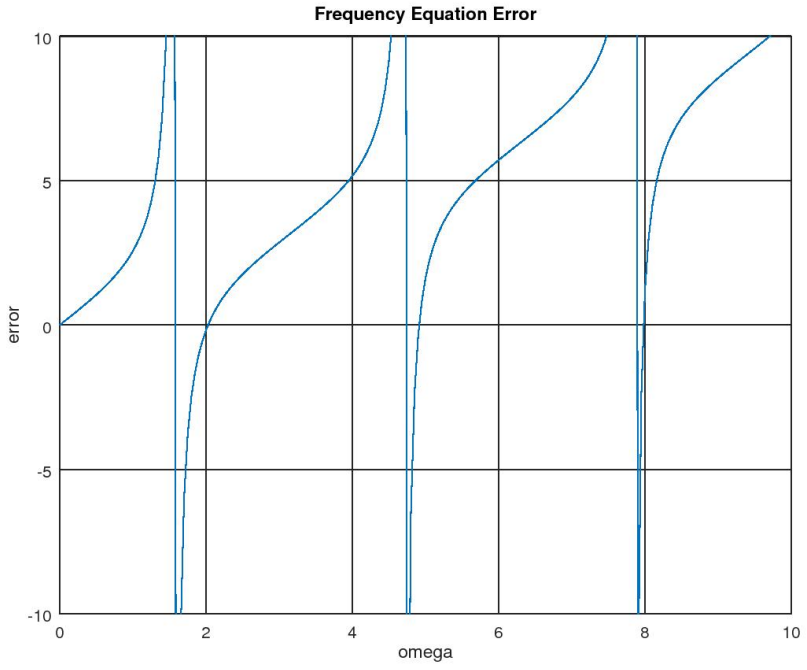
```
% to find out how to modify the plot
%help plot
% (also google 'matlab chart line properties')

% --: dashed line, o: circle symbols, r: red line
plot(omegaVals',errorVals','--or','LineWidth',2)
```

Try, try, again


300
250
200
150
100
50
0
-50
0
2
4
6
8
10


300
250
200
150
100
50
0
-50
0
2
4
6
8
10

```
% redo from scratch
plot(omegaVals',errorVals')

% now reduce the vertical extent of the plot
axis([-0 10 -10 10])

% and add a grid
grid on

% put the x-axis at y=0
%set(gca)
set(gca,'xaxislocation','origin')

% add labels on the x- and y-axes
xlabel('omega')
ylabel('error')

% add a title
title('Frequency Equation Error')
```



FINDING ACCURATE VALUES FOR THE FREQUENCIES

To keep it simple, let's keep k=1 for now and find the lowest frequency first.

Finding the value of the lowest frequency

We want to find the lowest positive frequency, call it `omega1`, where `freqEq1Error` is zero.
Matlab can find zeros ('roots') of functions using the `fzero` library function.
Warning: The word "root" as used here has nothing to do with a square root. It simply means the position where a function is zero. For example if $f(x) = 0$ when $x = a$, then a is a "root" of the equation $f(x) = 0$.

```
% Get a clue how to use fzero first
%help fzero

% tell fzero to start searching from omega=2
disp(' ')
omega1Guess=2
omega1=fzero('freqEq1Error',omega1Guess)
disp('This happens to be OK.')
disp('But it might just as well have failed.')
disp('It only works started close enough to the answer.')

% suppose we start at .5 pi
disp(' ')
omega1Guess=0.5*pi
omega1=fzero('freqEq1Error',omega1Guess)
disp('Oops.  In fact we could have ended up *anywhere*!')
disp('(If there is no singularity, Matlab usually gets')
disp('it right.  But Octave frequently gets it wrong.)')

% The _safe_ way is to tell fzero to search in a small
% interval that contains only the root we want, like from
% 1.9 to 2.1.  Note from above that the errors are of
% opposite sign at those two values, so it *must* be
% zero somewhere in between.

% let fzero search between 1.9 and 2
disp(' ')
omega1Interval=[1.9 2.1]
disp('The end points must be of different sign.')
disp('One root, and no singularities, inside.')
freqEq1Error(omega1Interval)
disp('That seems to be OK!')
omega1=fzero('freqEq1Error',omega1Interval)
```

```
disp('Same as before.')
disp('But this method was absolutely safe!')
```

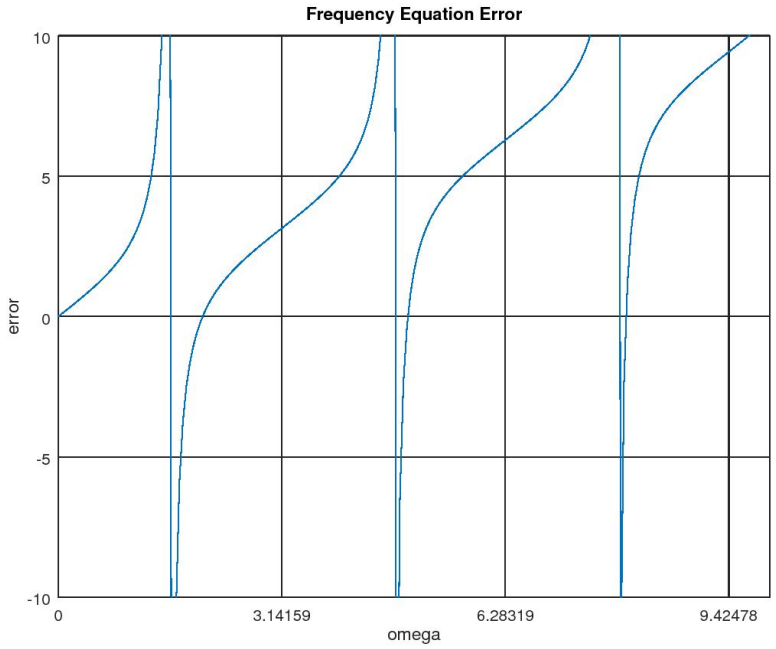
```
omega1Guess =  2
omega1 =  2.0288
This happens to be OK.
But it might just as well have failed.
It only works started close enough to the answer.

omega1Guess =  1.5708
omega1 =  1.5708
Oops.  In fact we could have ended up *anywhere*!
(If there is no singularity, Matlab usually gets
it right.  But Octave frequently gets it wrong.)

omega1Interval =
   1.9000   2.1000
The end points must be of different sign.
One root, and no singularities, inside.
ans =
  -1.02710   0.39015
That seems to be OK!
omega1 =  2.0288
Same as before.
But this method was absolutely safe!
```

Find many more frequencies

How about the other frequencies? This is going to be messy. Let's look a bit better at the plot first.

```
% redo from scratch (for publishing purposes)
plot(omegaVals',errorVals')

% now reduce the vertical extent of the plot
axis([-0 10 -10 10])

% and add a grid
grid on

% put the x-axis at y=0
%set(gca)
set(gca,'xaxislocation','origin')

% add labels on the x- and y-axes
```

```
xlabel('omega')
ylabel('error')

% add a title
title('Frequency Equation Error')

% set the tick marks at multiples of pi
set(gca,'xtick',[0:pi:3*pi])
```



Trick: modify the equation

The conclusio from the graphs seems to be that if you want to find more frequencies, it would be simplest to start `fzero` at odd values of $\pi/2$. But that does not work because the `tan` is infinite there.
But suppose we multiply the original equation

$$-k\omega = tan(\omega)$$

by $\cos(\omega)$:

$$-k\omega\cos(\omega) = \sin(\omega)$$

Then there is no longer a singularity at any omega. The error becomes:

$$\text{Error}(\omega) = \sin(\omega) + \text{k}\omega\cos(\omega)$$

So we define a new function:

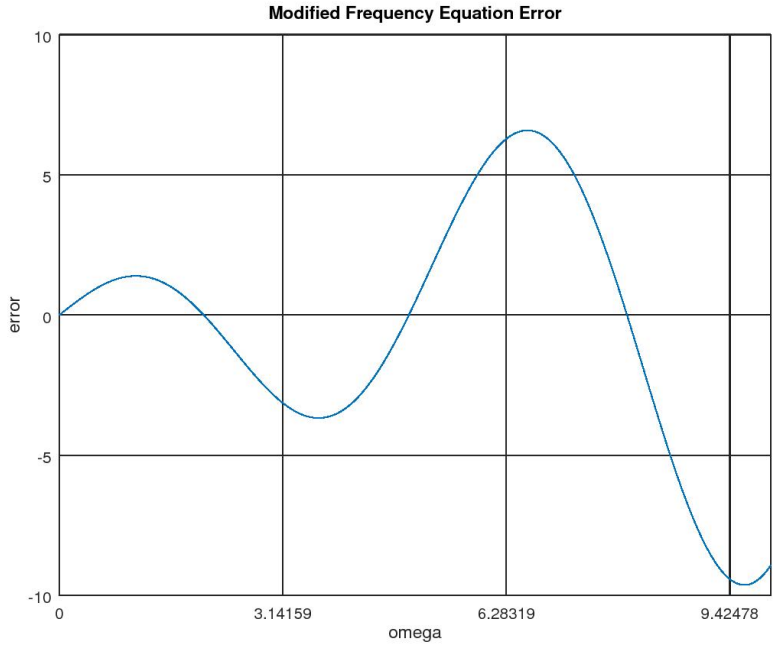
```
function error = freqEq1ErrorMod(omega)

% This function returns the error in the equation
% satisfied by the frequencies of a string with one end
% flexibly attached.  The scaled attachment flexibility k
% is assumed to be 1.
%
% Input:
%    omega: the frequency to test
% Output:
%    error: zero if omega is a correct frequency (tone)
%           of the string, nonzero if it is not.
%
% Advanced analysis taught in Analysis in Mechanical
% Engineering II shows that the equation the frequencies
% must satisfy is:
%                - k omega = tan(omega)
% However, the tan is infinite at any odd amount of pi/2,
% and that is a numerical problem.  So we multiply both
% sides by the cosine:
%             - k omega cos(omega) = sin(omega)
% Then if the frequency is not right, the error in the
% equation (difference between the right and left hand
% sides) is:
%          error = sin(omega) + k omega cos(omega)

% Note that omega is in radians.
% Do not forget the . before * and semi-colon.
error = sin(omega) + omega.*cos(omega);

end
```

```
% let's plot it
errorVals=freqEq1ErrorMod(omegaVals);
plot(omegaVals',errorVals')

% add a grid
grid on

% put the x-axis at y=0
```

```
set(gca,'xaxislocation','origin')

% add labels on the x- and y-axes
xlabel('omega')
ylabel('error')

% add a title
title('Modified Frequency Equation Error')

% set the tick marks at multiples of pi
set(gca,'xtick',[0:pi:3*pi])

% let's try it out
disp(' ')
omega1Interval=[0.5*pi 1.5*pi]
omega1=fzero('freqEq1ErrorMod',omega1Interval)
% yes, that produced the correct root

% seems to work OK:
disp(' ')
omega2Interval=[1.5*pi 2.5*pi]
omega2=fzero('freqEq1ErrorMod',omega2Interval)

% try the next one
disp(' ')
omega3Interval=[2.5*pi 3.5*pi]
omega3=fzero('freqEq1ErrorMod',omega3Interval)

% and the next
disp(' ')
omega4Interval=[3.5*pi 4.5*pi]
omega4=fzero('freqEq1ErrorMod',omega4Interval)
disp('We may approximate the rest as ((2n-1)/2)*pi')
% at some point, the frequencies will get so close to the
% odd multiple of pi/2 that we can ignore the difference.
```

```
omega1Interval =
    1.5708    4.7124
omega1 =  2.0288

omega2Interval =
    4.7124    7.8540
omega2 =  4.9132

omega3Interval =
```

```
    7.8540   10.9956
omega3 =  7.9787

omega4Interval =
   10.996   14.137
omega4 =  11.086
We may approximate the rest as ((2n-1)/2)*pi
```



HOW ABOUT IF THE STIFFNESS IS NOT 1??

So far we assumed that k was 1 in

$$-k\omega\cos(\omega) = \sin(\omega)$$

What if it is not? Surely we cannot create a new function for every possible value of k??
So we must create a function that accepts k as an input argument. Then we can use that function for *any* k we want:

```
function error = freqEqError(omega,k)

% Function used to find the natural frequencies of a
% string that has one end rigidly attached to the musical
```

```
% instrument but the other end attached to a flexible
% strip.
%
% Input:
%    omega: The natural frequency in radians.
%    k:     The bending flexibility of the strip.
%    Both are suitably nondimensionalized in a way not
%    important here.
%
% Output:
%    error: If error is zero, then the frequency is a
%           valid one for that value of k.  Note that a
%           string can vibrate with infinitely many
%           frequencies (theoretically at least)
%
% Advanced analysis taught in Analysis in Mechanical
% Engineering II shows that the equation the frequencies
% must satisfy is:
%                 - k omega = tan(omega)
% However, the tan is infinite at any odd amount of pi/2,
% and that is a numerical problem.  So we multiply both
% sides by the cosine:
%           - k omega cos(omega) = sin(omega)
% Then if the frequency is not right, the error in the
% equation (difference between the right and left hand
% sides) is:
%          error = sin(omega) + k omega cos(omega)

% Note that omega is in radians.
% Do not forget the . before * and the semi-colon.
error = sin(omega) + k*omega.*cos(omega);

end
```

But how do we tell fzero what k to use??

There is no way to tell `fzero` to use a second input argument in a function. Instead we must tell Matlab itself to provide `fzero` a new function that has the desired value of `k` already in it.
The convenient way to do that is to tell matlab to create an anonymous (nameless) function `(x)` of `x` that for given `x` returns `freqEqError(x,k)`, with `k` the value we want. That can be done as

```
@(omega) freqEqError(omega,k)
```

(The "@" is *not* a function name. It tells matlab to create a "handle" to that function for fzero to get hold of it.)

```
% let's first try it for the current value k = 1
disp(' ')
k=1
disp(['Current k-value: ', num2str(k)])
omega1Interval=[0.5*pi 1.5*pi]
omega1=fzero(@(omega) freqEqError(omega,k),omega1Interval
    )
omega2Interval=[1.5*pi 2.5*pi]
omega2=fzero(@(omega) freqEqError(omega,k),omega2Interval
    )
disp('Seems to work OK.')

% how about another value of k now?
disp(' ')

% notify about the new k value
k=2;
disp(['New k-value: ', num2str(k)])

% compute the new frequencies
omega1Interval=[0.5*pi 1.5*pi]
omega1=fzero(@(omega) freqEqError(omega,k),omega1Interval
    )
omega2Interval=[1.5*pi 2.5*pi]
omega2=fzero(@(omega) freqEqError(omega,k),omega2Interval
    )
omega3Interval=[2.5*pi 3.5*pi]
omega3=fzero(@(omega) freqEqError(omega,k),omega3Interval
    )
omega4Interval=[3.5*pi 4.5*pi]
omega4=fzero(@(omega) freqEqError(omega,k),omega4Interval
    )
disp('Seems to work OK.')
```

```
k =  1
Current k-value: 1
omega1Interval =
    1.5708    4.7124
omega1 =  2.0288
omega2Interval =
    4.7124    7.8540
omega2 =  4.9132
```

```
Seems to work OK.

New k-value: 2
omega1Interval =
    1.5708    4.7124
omega1 =  1.8366
omega2Interval =
    4.7124    7.8540
omega2 =  4.8158
omega3Interval =
     7.8540    10.9956
omega3 =  7.9171
omega4Interval =
    10.996    14.137
omega4 =  11.041
Seems to work OK.
```

PRINT OUT THE FREQUENCIES NICELY

The `fprintf` function allows you to print out numbers in your own way. Use it as

```
fprinf('FORMAT_STRING',VALUES)
```

Function `fprintf` uses the following symbols in FORMAT_STRING:

- %i: integer (also %d)
- %f: floating point number
- %e: floating point number in exponential notation
- %g: best choice of %f or %e

More advanced formatting:

- %PRINTPOSITIONSi
- %PRINTPOSITIONS.DIGITSBEHINDPOINTf

```
Warning:
You need a \n at the end to go to the next line.
```

```
% the first %f gets replaced by k
% the %i gets replaced by the frequency number
% the second %f gets replaced by omega
disp(' ')
```

```
fprintf('for k =%f, omega%i equals: %f\n',k,1,omega1)
fprintf('for k =%f, omega%i equals: %f\n',k,2,omega2)
fprintf('for k =%f, omega%i equals: %f\n',k,3,omega3)
fprintf('for k =%f, omega%i equals: %f\n',k,4,omega4)
disp('From now on, all FINAL numbers MUST use fprintf!')
disp('ILLEGAL to put ANY printed numbers in the STRING!')

% take control of the formatting
disp(' ')
disp('You MUST take control of the formatting!')
fprintf('for k =%5.2f, omega%1i equals:%6.3f\n',k,1,
    omega1)
fprintf('for k =%5.2f, omega%1i equals:%6.3f\n',k,2,
    omega2)
fprintf('for k =%5.2f, omega%1i equals:%6.3f\n',k,3,
    omega3)
fprintf('for k =%5.2f, omega%1i equals:%6.3f\n',k,4,
    omega4)
disp('Note that %f performs rounding!')
```

```
for k =2.000000, omega1 equals: 1.836597
for k =2.000000, omega2 equals: 4.815842
for k =2.000000, omega3 equals: 7.917053
for k =2.000000, omega4 equals: 11.040830
From now on, all FINAL numbers MUST use fprintf!
ILLEGAL to put ANY printed numbers in the STRING!

You MUST take control of the formatting!
for k = 2.00, omega1 equals: 1.837
for k = 2.00, omega2 equals: 4.816
for k = 2.00, omega3 equals: 7.917
for k = 2.00, omega4 equals:11.041
Note that %f performs rounding!
```

GETTING THE K VALUE DIRECTLY FROM THE KEYBOARD

To create a script `findFrequencies.m` that allows *any* Matlab user to compute any frequency for any stiffness, do it as follows:

```
%% String frequencies script
%
% This script will compute the natural frequencies of a
% string that has one end rigidly attached to the musical
```

```
% instrument but the other end attached to a flexible
% strip.  It gets the strip stiffness and frequency
% number directly from the user.

% keep doing this until the user enters 0 for k
while (1==1)

    % get the stiffness
    k=input('Enter the stiffness k (0 to quit): ');

    % quit if not positive
    if (k <= 0)
        break
    end

    % keep finding frequencies until the user enters 0
    while (1==1)

        % get the root number
        n=input('Desired root number (0 to change k): ');

        % quit the inner loop if not positive
        if (n <= 0)
            break
        end

        % find the frequency
        omegaInterval=[(2*n-1)/2*pi (2*n+1)/2*pi];
        omega=fzero(@(omega) freqEqError(omega,k),...
                    omegaInterval);

        % print it out nicely
        fprintf(...
            'for k =%6.3f, frequency%2i equals: %f\n',...
            k,n,omega)

    end

end
```

This uses the `input` function to get the desired values from the user. Note that using the `input` function prevents publishing in at least Octave.

ADDITIONAL REMARKS

To find the smallest or largest value of a function instead of a zero value, you could find a zero for the derivative. Alternatively, you can directly search for a minimum by using Matlab function `fminbnd` instead of `fzero`. To search for a maximum, search for a minimum of minus the function.
If you have more than one variable, things get messier. Try `fzero` or `fminunc`.

End lesson 2