Lecture19

March 27, 2018

1 AMAT503: Lecture 19

March 22, 2018 and March 26.

Michael Lamoureux

We are doing multiresolution analysis, applied to real functions on the line.

```
In [1]: ## Some startup commands

        %matplotlib inline
        from numpy import *
        from scipy import *
        from matplotlib.pyplot import *
        from pywt import *
```

1.1 1. Multiresolution Analysis - applied

We want to do a multiresolution analysis on some real functions. You might need to look at the last class's lecture notes to see the details.

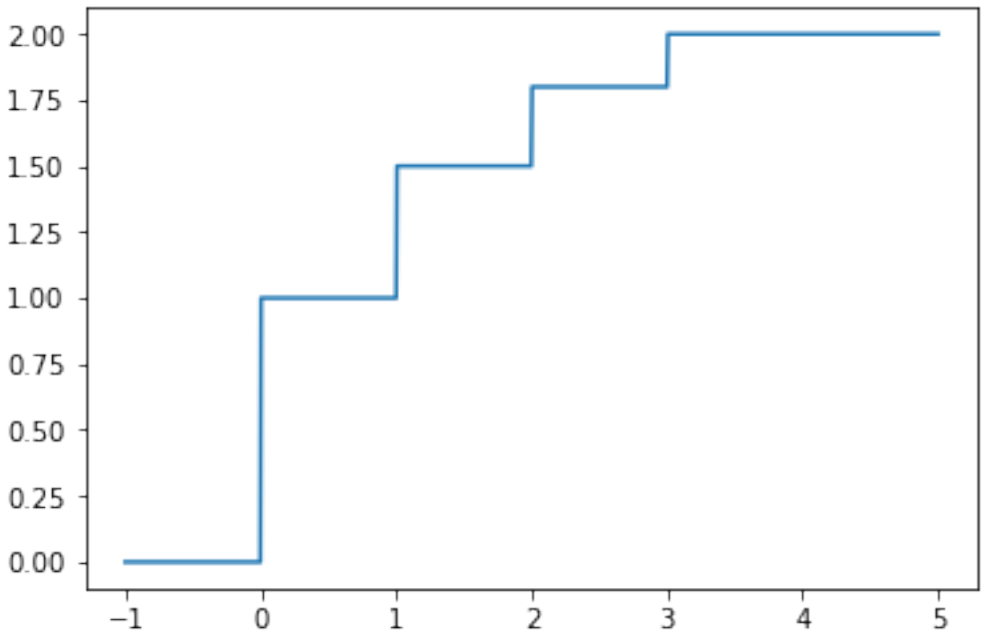
We start with a function $f(t)$ and a scaling function $\phi(t)$ in the space V_0 and write down some coefficients

$$y_k = \int f(t)\overline{\phi(t-k)}\,dt.$$

We then apply the corresponding discrete wavelet transform to get coefficients for the multiresolution analysis. This way, function f is expressed as a linear combination of scaled versions of the wavelet functions (and a scaled version of the scaling function.)

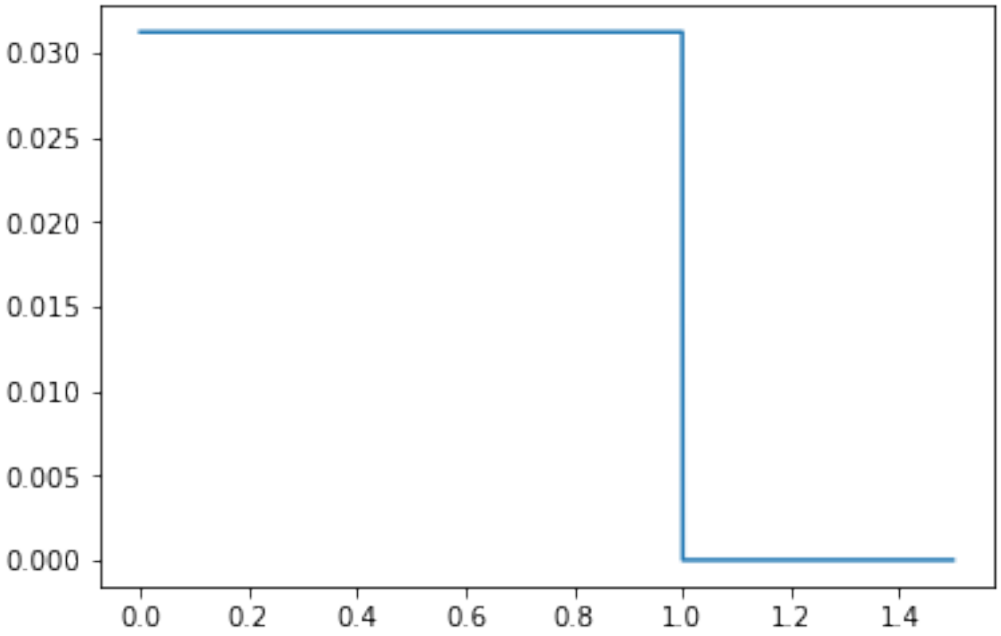
We start by defining a function:

```
In [2]: def f(t):
            return (t>0)+ .5*(t>1) + .3*(t>2) + .2*(t>3)

In [3]: t = linspace(-1,5,1000)
        plot(t,f(t));
```



Let's get our scaling function, and wavelet function.

```
In [8]: w = Wavelet('db1')
        h = w.dec_lo
        z = [1]
        for k in range(10):
            x = convolve(z,h)
            z = zeros(2*size(x))
            z[0:size(z):2] = x

        phi = x
        plot(linspace(0,size(phi)/1024,size(phi)),phi);
```




```
In [5]: size(phi)
```

```
Out[5]: 1535
```

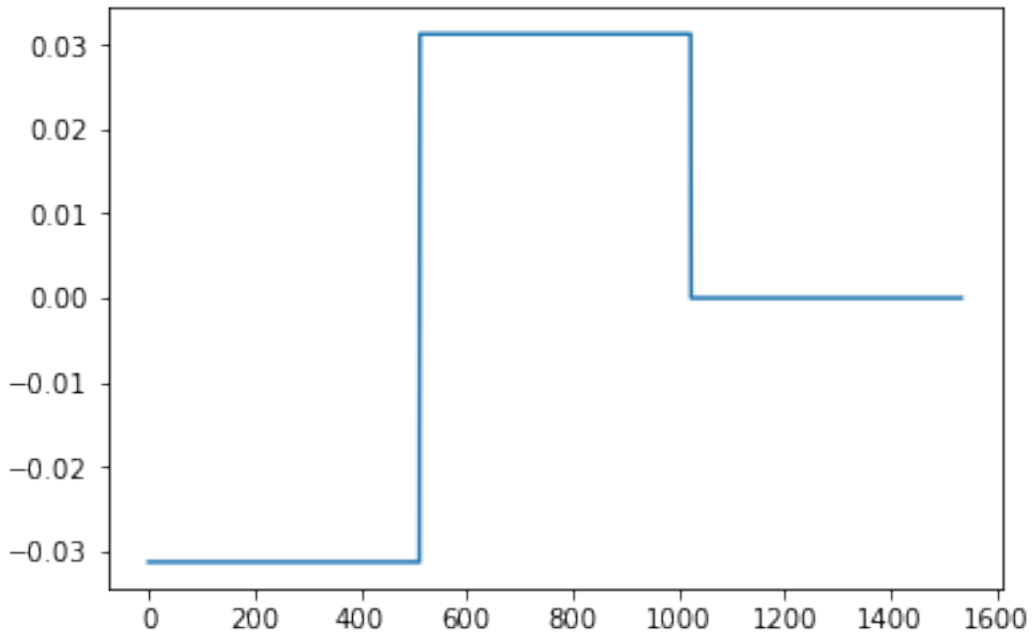
```
In [6]: h
```

```
Out[6]: [0.7071067811865476, 0.7071067811865476]
```

```
In [9]: # Check normalization, sum of squares should be one.
        sum(phi**2)
```

```
Out[9]: 1.0000000000000002
```

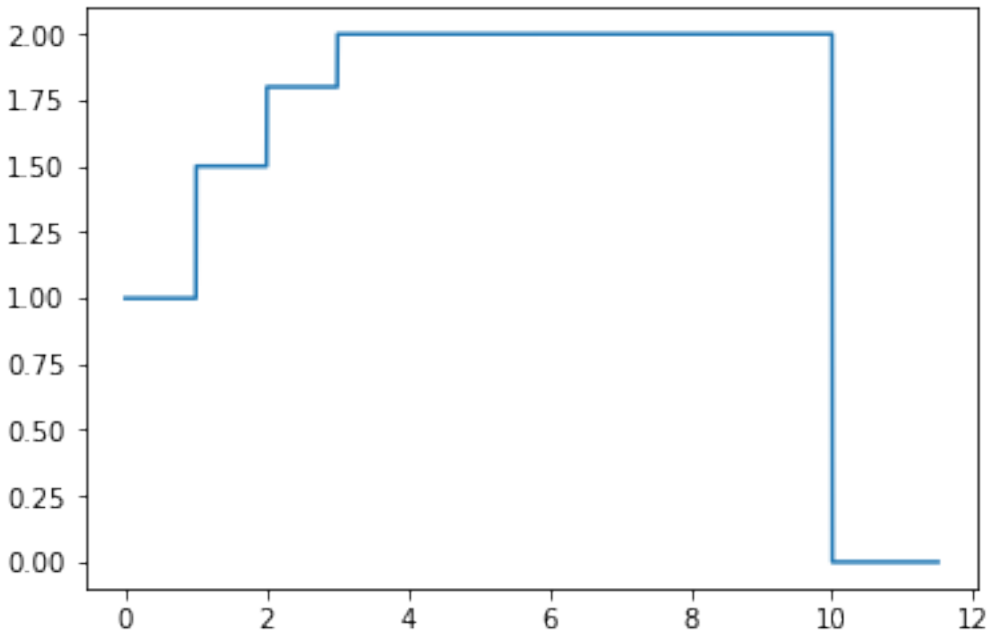
```
In [10]: w = Wavelet('db1')
         h = w.dec_lo
         g = w.dec_hi
         z = [1]
         x = convolve(z,g)
         z = zeros(2*size(x))
         z[0:size(z):2] = x

         for k in range(9):
             x = convolve(z,h)
             z = zeros(2*size(x))
             z[0:size(z):2] = x

         psi = x
         plot(psi);
```



In [13]:
```
# Compute the initial scaling coefficient.

N = 10
y = zeros(N,float_)
t = linspace(0,1,size(phi))
dt = 1/size(phi)
for k in range(N):
    y[k] = sum(f(t+k)*phi)

y
```

Out[13]:
```
array([ 31.96875 ,  47.984375,  57.590625,  63.99375 ,  64.      ,
        64.      ,  64.      ,  64.      ,  64.      ,  64.      ])
```

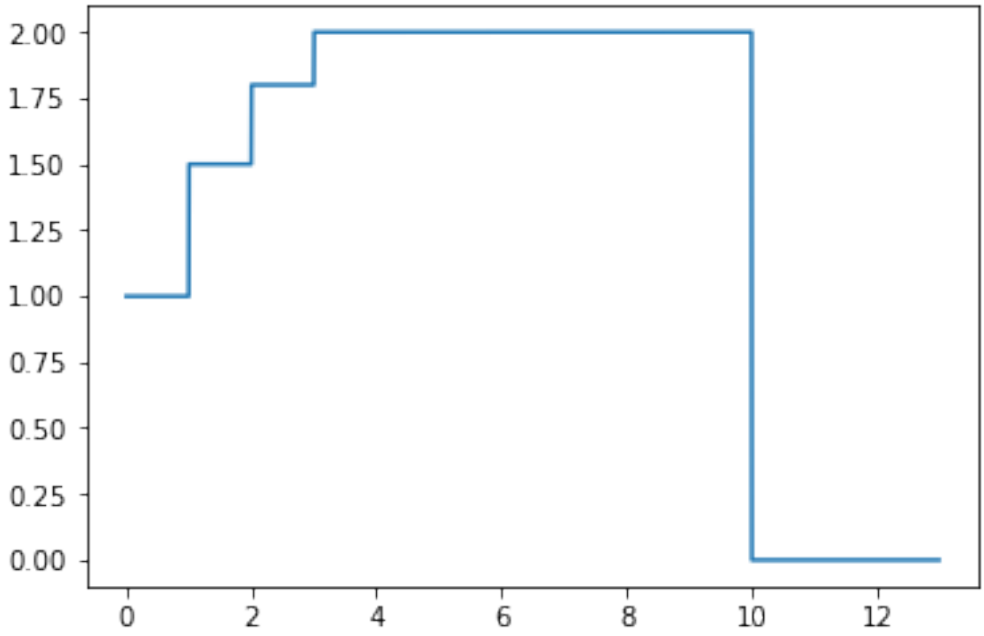
In [14]:
```
# Let's build a reconstruction of f from these coefficients

f_len = 1024*size(y) + size(phi)
f_recon = zeros(f_len, float_)

for k in range(size(y)):
    f_recon[(1024*k):(1024*k+size(phi))] = f_recon[(1024*k):(1024*k+size(p

plot(linspace(0,size(f_recon)/1024,size(f_recon)),f_recon);
```




```
In [15]: ## Let's compute some wavelet coefficients

         (cA, cD) = dwt(y, 'db1')
         cA, cD

Out[15]: (array([ 56.53539686,  85.97313605,  90.50966799,  90.50966799,  90.509667
          array([-11.32475704,  -4.52769311,   0.        ,   0.        ,   0.
```

We should be able to reconstruct f using these coefficients.

```
In [18]: fw_len = 1024*size(cA) + size(phi)
         fw_recon = zeros(fw_len, float_)

         for k in range(size(cA)):
             fw_recon[(1024*k):(1024*k+size(phi))] = fw_recon[(1024*k):(1024*k+size
                 cA[k]*phi - cD[k]*psi

         plot(linspace(0,size(fw_recon)/512,size(fw_recon)),fw_recon/sqrt(2));
```



Notice I had to put a sqrt(2) in the plot command to get this to work out correctly. You might want to think about that. (So do I.)

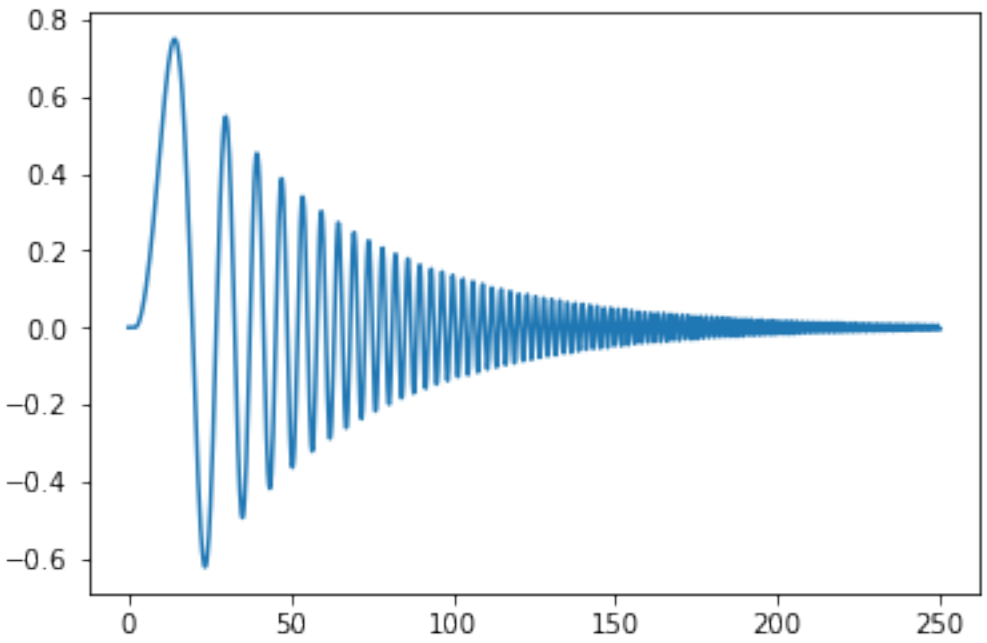
Now, of course we could repeat this discrete wavelet transform several times. The point is, we never have to computer inner products anymore to get the wavelet coefficients for the function. They just come out of the discrete wavelet algorithm. This is a very powerful idea.

1.1.1 More complicated functions, wavelets

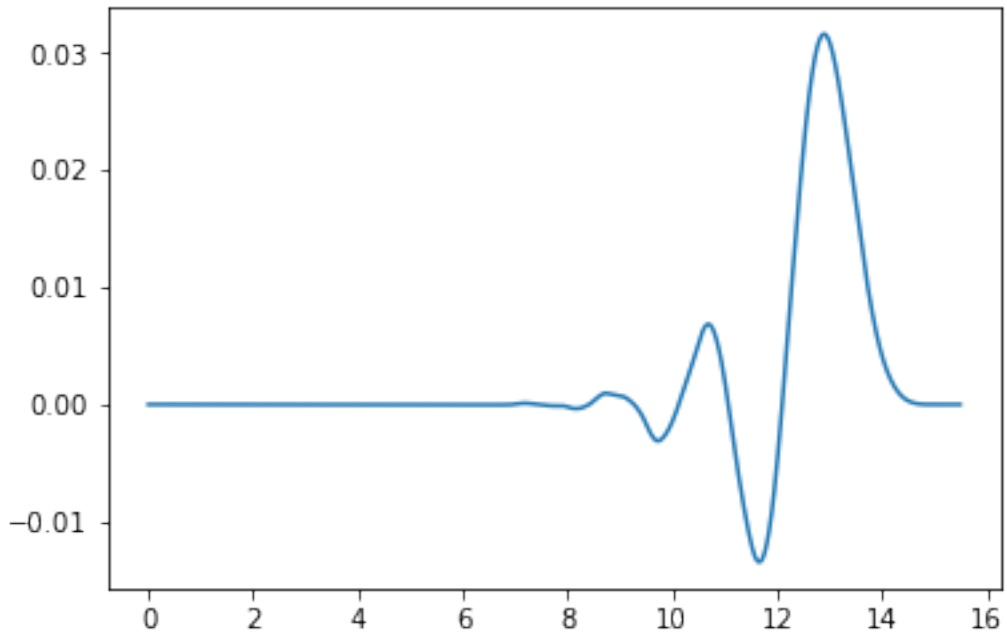
Let's see what happens if we use a more complicated function. Something with lots of wiggles and changes. I suggest we use a sinusoid that decays quickly. Maybe we can let the frequency change a lot.

We should then try to approximate using a wiggly Daubechies wavelets.

```
In [125]: def f(t):
              return (t>2)*exp(-t/50)*sin(.01*(t-2)**2)

          t=linspace(0,250,1000)
          plot(t,f(t));
```

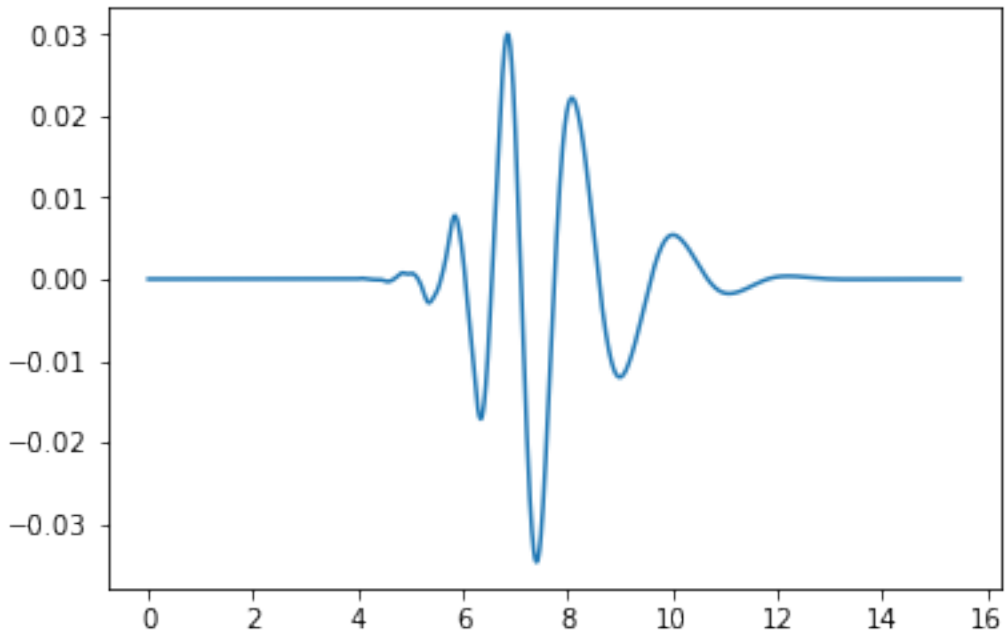



```
In [126]: w = Wavelet('db8')
          h = w.dec_lo
          z = [1]
          for k in range(10):
              x = convolve(z,h)
              z = zeros(2*size(x))
              z[0:size(z):2] = x

          phi = x
          plot(linspace(0,size(phi)/1024,size(phi)),phi);
```



Notice this scaling function looks backwards (it is backwards!)

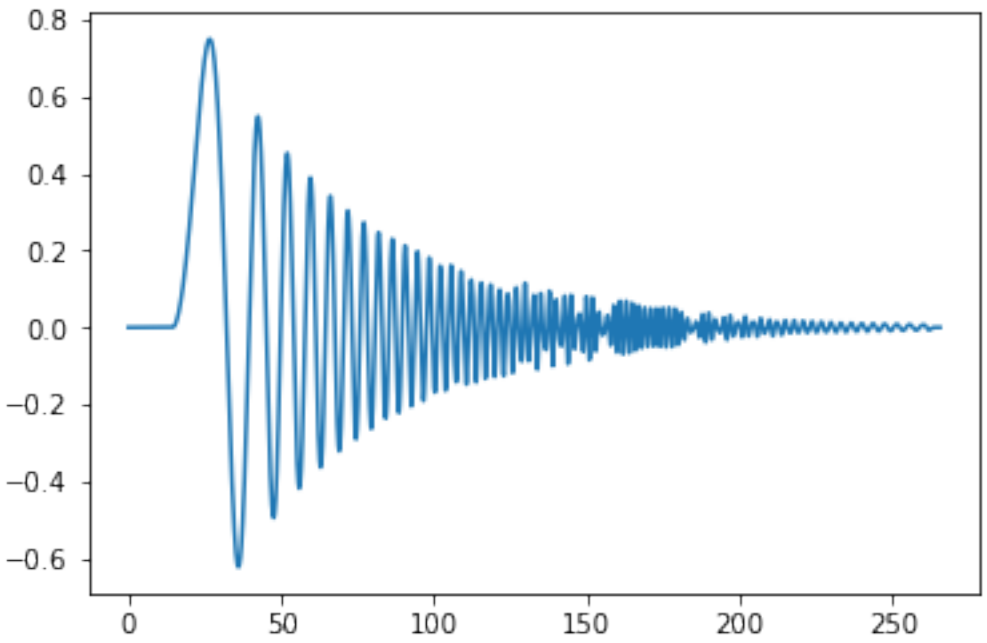
```
In [127]: w = Wavelet('db8')
          h = w.dec_lo
          g = w.dec_hi
          z = [1]
          x = convolve(z,g)
          z = zeros(2*size(x))
          z[0:size(z):2] = x

          for k in range(9):
              x = convolve(z,h)
              z = zeros(2*size(x))
              z[0:size(z):2] = x

          psi = x
          plot(linspace(0,size(psi)/1024,size(psi)),psi);
```



This wavelet function is also backwards, at least according to Daubechies' book, and Wikipedia.

In [128]:
```
# Compute the initial scaling coefficient.

N = 250
y = zeros(N,float_)
t = linspace(0,1,size(phi))
dt = 1/size(phi)
for k in range(N):
    y[k] = sum(f(t+k)*phi)
```

In [129]:
```
# Let's build a reconstruction of f from these coefficients

f_len = 1024*size(y) + size(phi)
f_recon = zeros(f_len, float_)

for k in range(size(y)):
    f_recon[(1024*k):(1024*k+size(phi))] = f_recon[(1024*k):(1024*k+size

plot(linspace(0,size(f_recon)/1024,size(f_recon)),f_recon);
```

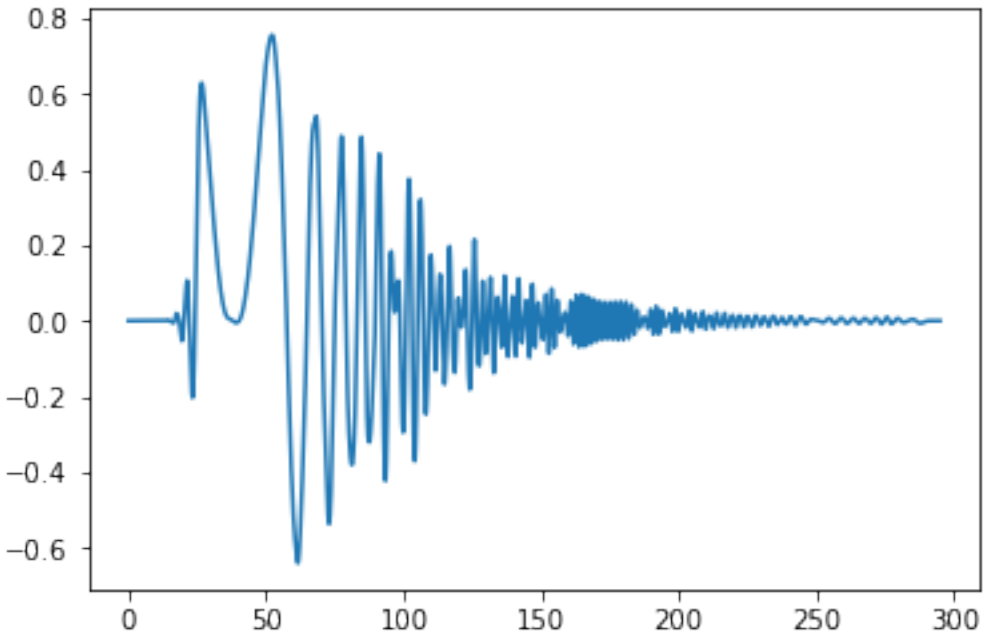



```
In [130]: ## Let's compute some wavelet coefficients

          (cA, cD) = dwt(y, 'db8')
          cA, cD

Out[130]: (array([  3.01750674e+01,   2.24816532e+01,   1.39375190e+01,
                    6.69155231e+00,   1.77825217e+00,   1.45125108e-01,
                   -4.90575612e-02,   1.45893530e-01,   2.82412388e+00,
                    8.39090792e+00,   1.61713237e+01,   2.47966212e+01,
                    3.18398238e+01,   3.38556104e+01,   2.74127217e+01,
                    1.14476991e+01,  -9.81804628e+00,  -2.60118140e+01,
                   -2.53542535e+01,  -5.33859009e+00,   1.89105167e+01,
                    2.29576801e+01,   1.39871525e-01,  -2.17278328e+01,
                   -1.07771646e+01,   1.67690478e+01,   1.29663404e+01,
                   -1.52909071e+01,  -9.72579016e+00,   1.68379894e+01,
                    1.15503654e+00,  -1.57698427e+01,   1.07682687e+01,
                    4.27332003e+00,  -1.36330230e+01,   1.18190202e+01,
                   -3.29345716e+00,  -5.24806152e+00,   1.00489185e+01,
                   -1.08252647e+01,   9.06780832e+00,  -6.43415840e+00,
                    4.01888852e+00,  -2.30076076e+00,   1.35643308e+00,
                   -1.07633333e+00,   1.29202049e+00,  -1.81985724e+00,
                    2.45710679e+00,  -2.97030348e+00,   3.11221817e+00,
                   -2.69221112e+00,   1.69302755e+00,  -3.71264138e-01,
                   -7.70463057e-01,   1.22205803e+00,  -8.33750313e-01,
                    1.64717399e-02,   5.23991845e-01,  -4.09754093e-01,
                   -5.07698887e-02,   2.63439713e-01,  -7.99239721e-02,
```

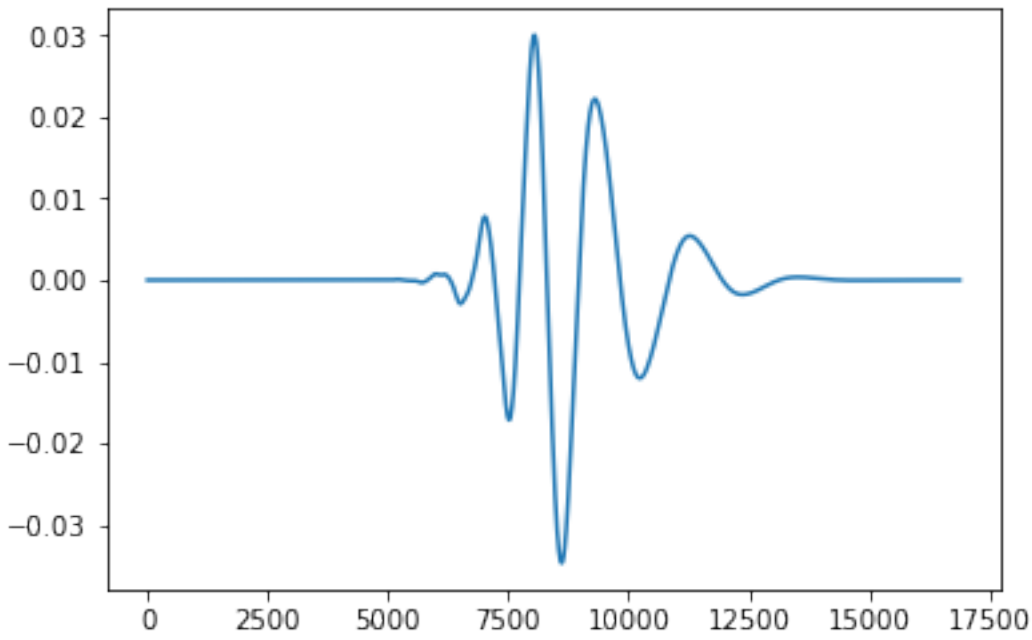
```
               -1.17080385e-01,   6.35448167e-02,   5.22330773e-02,
               -2.97731785e-02,  -2.67703896e-02,   8.13762958e-03,
                1.34017952e-02,   1.17563506e-03,  -4.75382283e-03,
               -2.62595525e-03,   3.74064984e-04,   1.13229577e-03,
                5.62611295e-04,  -8.24468785e-06,  -1.86902744e-04,
               -1.32693849e-04,  -4.27768047e-05,   6.54619767e-06,
                1.84155961e-05,   1.46224137e-05,   9.41138884e-06,
                6.84760854e-06,   4.38325112e-06,  -4.09296934e-06,
               -2.38900585e-05,  -4.91994233e-05,  -4.86067195e-05,
                3.14364818e-05,   2.02791154e-04,   3.10913459e-04,
                2.09183834e-05,  -7.62638544e-04,  -1.16148321e-03,
                3.30573065e-04,   3.01810443e-03,   2.03068232e-03,
               -4.86823537e-03,  -6.73081827e-03,   6.98536801e-03,
                1.36491063e-02,  -1.24577070e-02,  -2.08768093e-02,
                2.77200352e-02,   1.81928686e-02,  -5.41155858e-02,
                1.87715877e-02,   5.71810955e-02,  -8.88673353e-02,
                3.56741144e-02,   6.32115428e-02,  -1.35162554e-01,
                1.35181702e-01,  -6.74383109e-02,  -3.26556553e-02,
                1.27273544e-01,  -1.93960598e-01,   2.28242756e-01,
               -2.37500808e-01,   2.33429442e-01,  -2.26856012e-01,
                2.25360748e-01,  -2.32582847e-01,   2.47897325e-01,
               -2.65521885e-01,   2.73146261e-01,  -2.51891320e-01,
                1.79073117e-01,  -2.75947792e-02,  -2.99854272e-01]),
       array([ -1.11040541e-02,   2.76636794e-02,  -6.79845512e-02,
                3.84632165e-02,  -4.67667119e-03,  -5.01230786e-03,
                1.24909834e-03,   7.78636755e-05,   8.99996997e-04,
                2.10271706e-03,   2.11827732e-03,  -1.51277071e-03,
               -8.75590841e-03,  -1.14657346e-02,   4.24334469e-03,
                3.45145347e-02,   3.25509144e-02,  -4.50743561e-02,
               -1.10558500e-01,   2.09463722e-02,   2.33464797e-01,
                2.69782397e-02,  -4.29480259e-01,  -1.09161867e-02,
                7.32275865e-01,  -3.07776367e-01,  -9.55065898e-01,
                1.20499657e+00,   2.54334710e-01,  -1.94679932e+00,
                2.02164133e+00,  -1.98918438e-01,  -2.21447849e+00,
                3.66388492e+00,  -3.44714112e+00,   1.84863858e+00,
                3.65929642e-01,  -2.49061485e+00,   4.13967498e+00,
               -5.23976589e+00,   5.89778399e+00,  -6.26259724e+00,
                6.42886769e+00,  -6.38430641e+00,   5.99152341e+00,
               -5.01066529e+00,   3.19104181e+00,  -4.63571075e-01,
               -2.78538601e+00,   5.51242036e+00,  -6.20925952e+00,
                3.79598150e+00,   1.00268169e+00,  -5.04225009e+00,
                4.61595070e+00,   4.42513699e-01,  -4.76821837e+00,
                2.68224702e+00,   3.15267219e+00,  -3.54366729e+00,
               -2.30315955e+00,   3.28408612e+00,   2.57633363e+00,
               -2.20624206e+00,  -3.31679999e+00,  -9.45728673e-03,
                2.94075001e+00,   2.58181192e+00,   1.78036814e-02,
               -2.21667712e+00,  -2.74787052e+00,  -1.78529588e+00,
               -2.44601073e-01,   1.09663784e+00,   1.90389535e+00,
```

```
 2.19756842e+00,   2.15484088e+00,   1.96324292e+00,
 1.75727295e+00,   1.61019157e+00,   1.54704038e+00,
 1.55809704e+00,   1.60537843e+00,   1.62386167e+00,
 1.52526841e+00,   1.21658888e+00,   6.44916424e-01,
-1.33282469e-01,  -8.93461544e-01,  -1.27160968e+00,
-9.58026268e-01,  -1.97557253e-02,   9.15603633e-01,
 1.00376862e+00,   5.91321506e-02,  -9.12267574e-01,
-6.49271358e-01,   5.33517445e-01,   7.95958187e-01,
-3.35339823e-01,  -7.53085958e-01,   3.81520129e-01,
 5.93790511e-01,  -5.77216487e-01,  -2.28486209e-01,
 6.68049314e-01,  -3.29586201e-01,  -2.83660993e-01,
 5.80748853e-01,  -4.20655149e-01,   2.98361649e-02,
 3.12147081e-01,  -4.64613591e-01,   4.34494913e-01,
-2.99816450e-01,   1.39839795e-01,  -3.83542005e-03,
-8.98032356e-02,   1.42079954e-01,  -1.61761628e-01,
 1.58077902e-01,  -1.37717127e-01,   1.04665362e-01,
-6.15695514e-02,   1.17457504e-02,   7.18452021e-03,
-2.84689639e-02,   6.08791218e-02,  -4.77027262e-02,
 4.22241287e-03,   4.78380943e-02,  -9.44417179e-02]))
```

In [131]:
```
fw_len = 1024*size(cA) + size(phi)
fw_recon = zeros(fw_len, float_)

for k in range(size(cA)):
    fw_recon[(1024*k):(1024*k+size(phi))] = fw_recon[(1024*k):(1024*k+siz
        cA[k]*phi - cD[k]*psi

plot(linspace(0,size(fw_recon)/512,size(fw_recon)),fw_recon/sqrt(2));
```



```
In [132]: ## Let's explore orthogonality

          phiOne = zeros(size(phi)+1024);
          phiTwo = zeros(size(phi)+1024);

          phiOne[0:size(phi)] = phi
          phiTwo[1024:(1024+size(phi))] = phi

In [133]: sum(phiOne*phiTwo)

Out[133]: 1.6050109680166517e-17

In [134]: psiOne = zeros(size(psi)+1024);
          psiTwo = zeros(size(psi)+1024);
          psiOne[0:size(psi)] = psi
          psiTwo[1024:(1024+size(psi))] = psi

In [135]: sum(psiOne*psiTwo)

Out[135]: 0.0

In [136]: sum(psiOne*phiOne)

Out[136]: -2.1684043449710089e-18

In [137]: plot(psiTwo)

Out[137]: [<matplotlib.lines.Line2D at 0x10c04b5c0>]
```



1.1.2 Comments.

Well, some of those reconstructions are terrible!

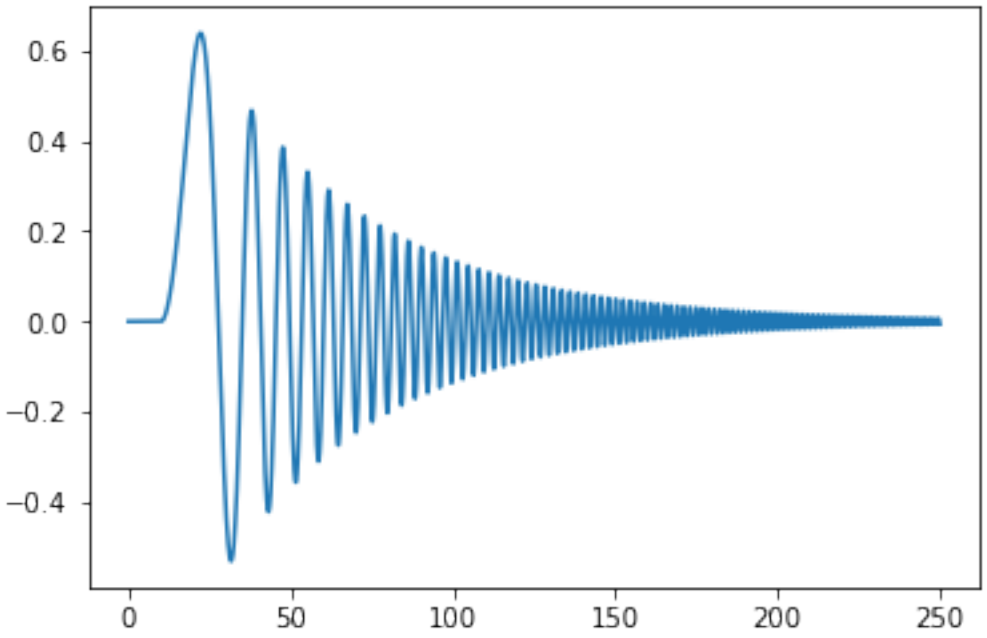
Let's explore in class, might be interesting.

1.2 Second attempt.

Here is what we did in class, to fix things.

First, we shift the function away from the boundary. Wavelets mess up at boundaries, unless you are very careful. We don't want to be that careful, so we just avoid the boundary

```
In [150]: def f(t):
              return (t>10)*exp(-t/50)*sin(.01*(t-10)**2)   # start at t=10

          t=linspace(0,250,10000)
          plot(t,f(t));
```

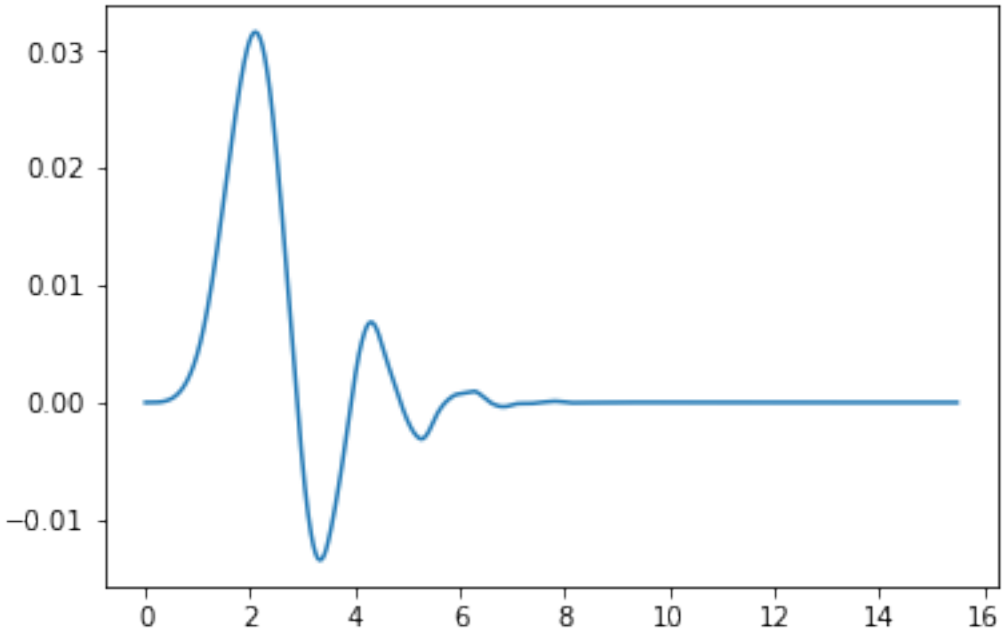


1.2.1 Flip wilson

The wavelets we got looked backwards, compared to what Wikipedia says our wavelets should look like. So let's flip the coefficients around. These will give scaling functions and wavelets that are minimum phase. I.e. most of the energy is concentrated at the front of the signal.

```
In [138]: w = Wavelet('db8')
          h = w.dec_lo
          h = h[::-1]
```

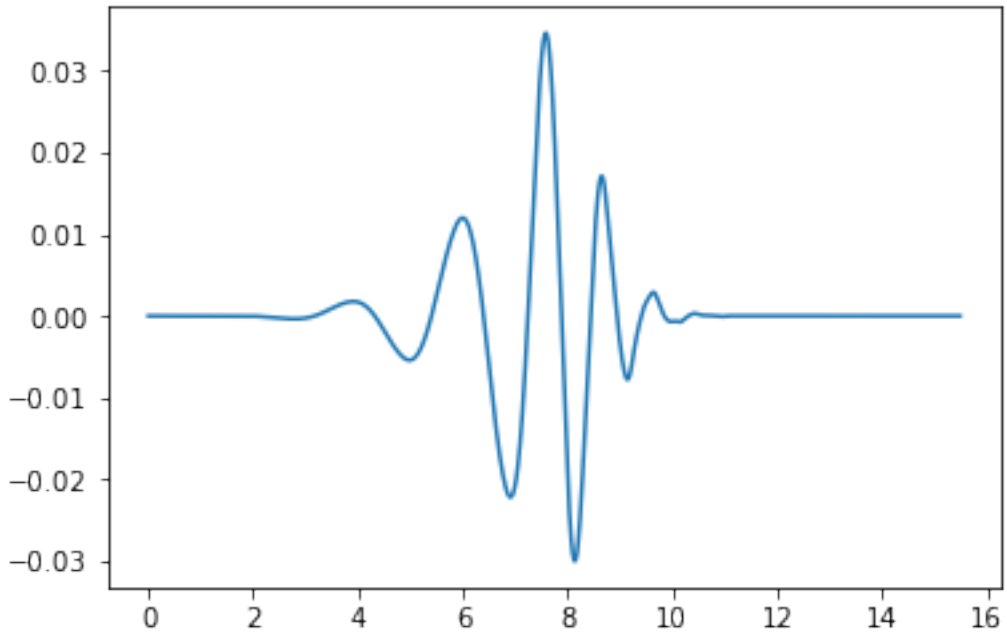
```
z = [1]
for k in range(10):
    x = convolve(z,h)
    z = zeros(2*size(x))
    z[0:size(z):2] = x

phi = x
plot(linspace(0,size(phi)/1024,size(phi)),phi);
```



For the wavelet, we reverse order and also make it negative. (z=[-1] initially). This way it looks like the Daubechies wavelet.

```
In [140]: w = Wavelet('db8')
          h = w.dec_lo
          h = h[::-1]
          g = w.dec_hi
          g = g[::-1]
          z = [-1]
          x = convolve(z,g)
          z = zeros(2*size(x))
          z[0:size(z):2] = x

          for k in range(9):
              x = convolve(z,h)
              z = zeros(2*size(x))
              z[0:size(z):2] = x
```

```
psi = x
plot(linspace(0,size(psi)/1024,size(psi)),psi);
```



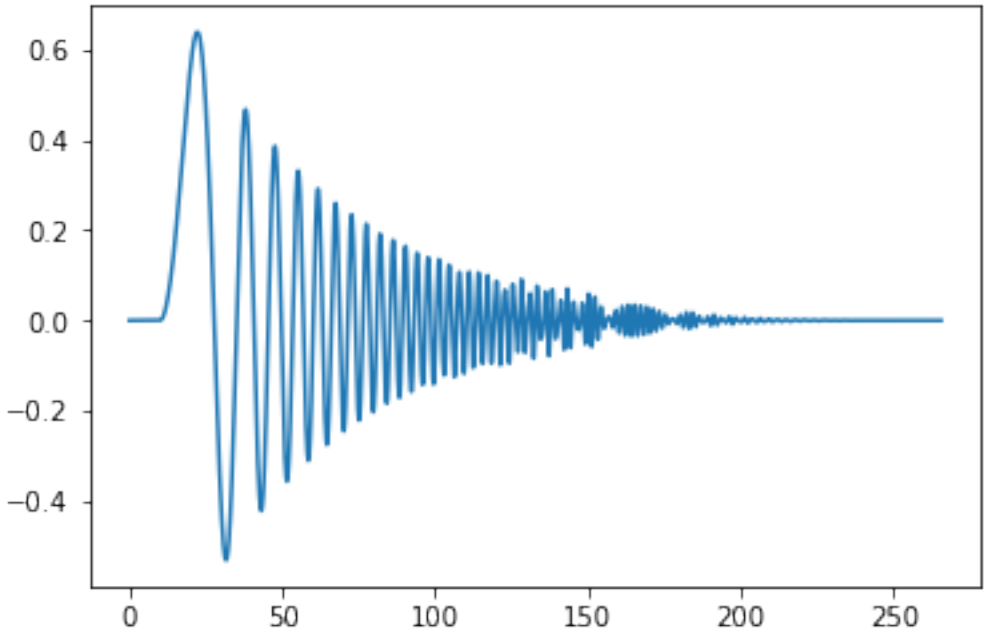
1.3 Another error.

We want to compute the integrals $y_k = \int f(t)\phi(t-k)dt$. We need to be careful about the range of t values that we use. The function $\phi(t)$ has support bigger than just the interval $[0,1]$. In the figure above, we see the support is really about $[0,1]$. The range of samples goes from 0 to size(phi)/1024.

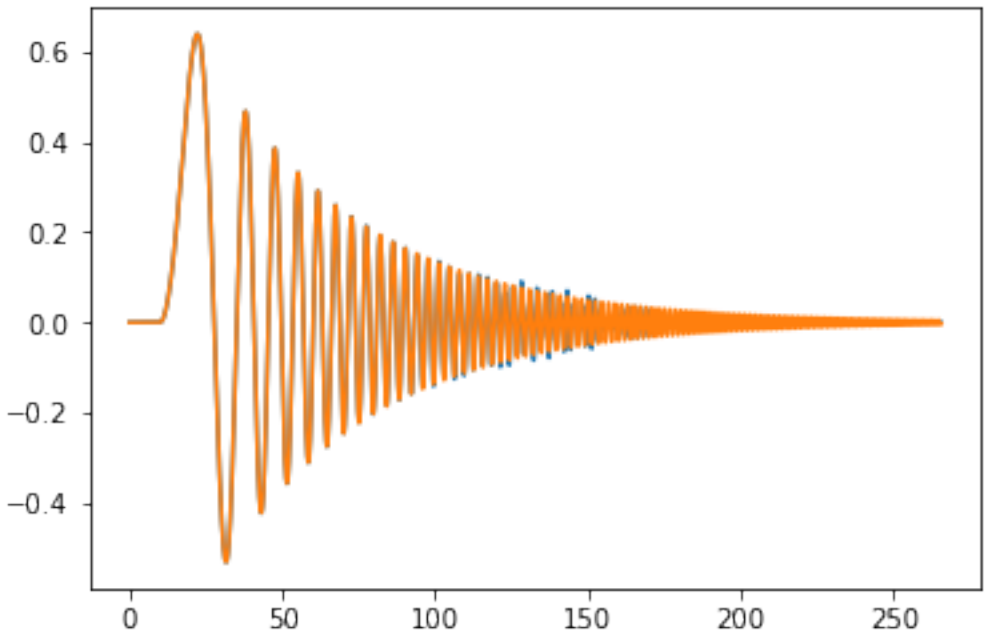
We have to use that in the t variable in the following code:

```
In [210]: # Compute the initial scaling coefficient.

          N = 250
          y = zeros(N,float_)
          t = linspace(0,size(phi)/1024,size(phi))
          dt = 1/size(phi)
          for k in range(N):
              y[k] = sum(f(t+k)*phi)

In [213]: # Let's build a reconstruction of f from these coefficients

          f_len = 1024*size(y) + size(phi)
          f_recon = zeros(f_len, float_)

          for k in range(size(y)):
              f_recon[(1024*k):(1024*k+size(phi))] = f_recon[(1024*k):(1024*k+size
```

```
## We plot just the reconstruction
tt = linspace(0,size(f_recon)/1024,size(f_recon))
plot(tt,f_recon);
```



In [214]:
```
## We plot the original function f and its reconstruction, one on top of
tt = linspace(0,size(f_recon)/1024,size(f_recon))
plot(tt,f_recon,tt,f(tt));
```



1.3.1 Another error

PyWavelets tries to do something sensible when dealing with finite lenght signals.

The issue is whether to wrap around, make the function periodic, make a reflection at the boundary, etc.

Choose poorly, and you get errors.

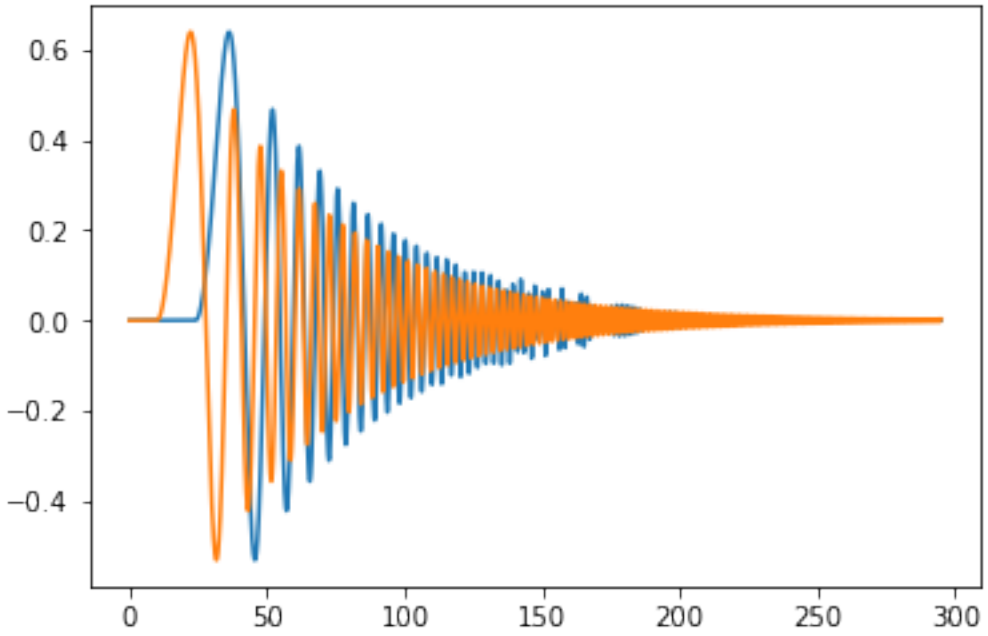
For us, we want to use zero padding, to treat the signal as if it is zero outside the samples that have been recorded and stored in our finite vector. When we call "dwt", we have to include the parameter 'zero' as follows.

```
In [215]: ## Let's compute some wavelet coefficients

          (cA, cD) = dwt(y, 'db8', 'zero')  # notice the keywork 'zero' to select z
          cA, cD

Out[215]: (array([ -1.87861998e-11,  -8.72081725e-09,  -4.11790461e-07,
                   -1.45919911e-06,  -1.65158340e-05,   3.44339669e-04,
                   -2.04590689e-03,   7.18892957e-03,  -1.88829389e-02,
                    4.51757875e-02,  -1.31922854e-01,   8.30103781e-01,
                    4.26611395e+00,   9.96065464e+00,   1.71110742e+01,
                    2.41728359e+01,   2.86404694e+01,   2.74031252e+01,
                    1.81352250e+01,   1.69158667e+00,  -1.58196170e+01,
                   -2.41348079e+01,  -1.55081523e+01,   5.67713623e+00,
                    2.08245441e+01,   1.27426520e+01,  -1.06308519e+01,
                   -1.81539626e+01,   2.43681217e+00,   1.75875658e+01,
                   -5.10133285e-01,  -1.61717304e+01,   3.99281150e+00,
```

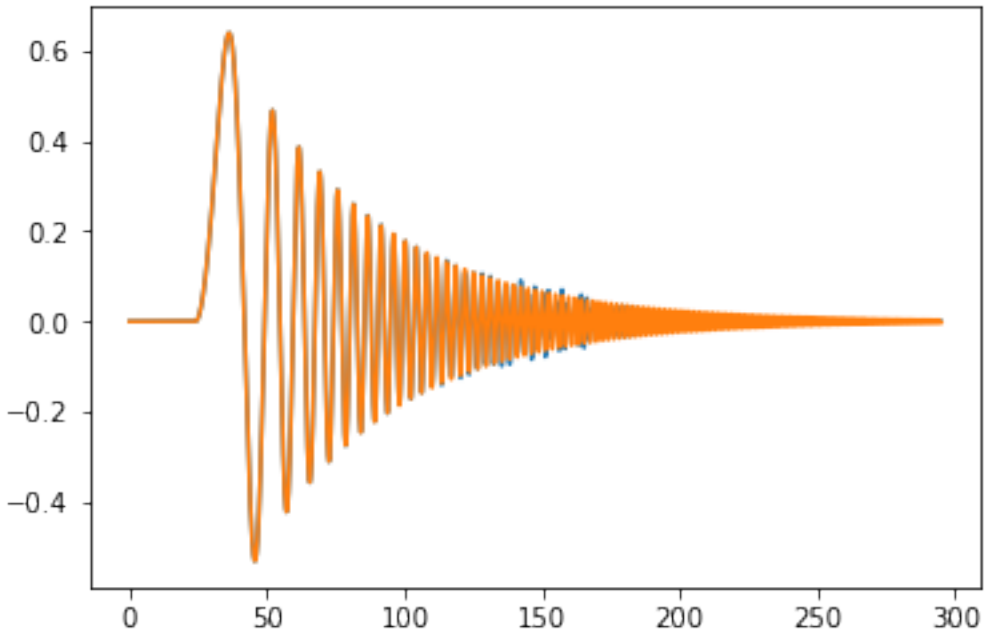
```
                1.31776724e+01,  -1.05363687e+01,  -4.62305953e+00,
                1.30550940e+01,  -8.39361081e+00,  -2.15413404e+00,
                9.62681353e+00,  -1.06504034e+01,   6.93202374e+00,
               -1.72741645e+00,  -2.63062114e+00,   5.29795272e+00,
               -6.43522617e+00,   6.57085616e+00,  -6.20013894e+00,
                5.63742875e+00,  -5.01455144e+00,   4.33350888e+00,
               -3.52983686e+00,   2.53723459e+00,  -1.35389303e+00,
                9.94744538e-02,   9.70810606e-01,  -1.53998506e+00,
                1.40367274e+00,  -6.66867236e-01,  -2.11338885e-01,
                6.68006796e-01,  -4.74949418e-01,  -3.88661801e-02,
                3.11068841e-01,  -1.50448287e-01,  -1.06337503e-01,
                1.14827227e-01,   2.93433006e-02,  -5.96714654e-02,
               -1.28331147e-02,   2.59507204e-02,   9.73379733e-03,
               -8.43735926e-03,  -6.52024796e-03,   7.60910903e-04,
                2.74077671e-03,   1.08646578e-03,  -3.49790770e-04,
               -5.68611537e-04,  -2.46876406e-04,   1.31761007e-05,
                8.50738024e-05,   5.87786015e-05,   1.98583249e-05,
               -1.62881922e-06,  -7.44416998e-06,  -5.85064971e-06,
               -1.75711794e-06,   3.41707726e-06,   8.56841275e-06,
                9.45988495e-06,  -2.05096641e-06,  -3.07409686e-05,
               -5.99738248e-05,  -3.89592097e-05,   7.98750354e-05,
                2.28668076e-04,   1.59055302e-04,  -2.98231860e-04,
               -6.88785187e-04,  -7.59690599e-05,   1.27659335e-03,
                9.58623845e-04,  -1.79757183e-03,  -2.12329085e-03,
                2.59845574e-03,   3.07792792e-03,  -4.37739615e-03,
               -2.59862059e-03,   7.04797733e-03,  -1.41930452e-03,
               -7.24551584e-03,   8.27409261e-03,  -6.90432216e-04,
               -7.82346869e-03,   1.03620333e-02,  -6.21843634e-03,
               -8.02010681e-04,   6.60095008e-03,  -9.25375720e-03,
                8.99645979e-03,  -7.07255305e-03,   4.71229318e-03,
               -2.68525983e-03,   1.29473201e-03,  -5.45531141e-04,
                3.06009953e-04,  -4.12307873e-04,   7.29061454e-04,
               -1.14798888e-03,   1.43456827e-03,  -1.37438071e-04]),
       array([ -8.70186303e-09,  -4.11855229e-06,  -2.27809979e-04,
               -2.58671565e-03,  -2.27853991e-02,   2.91142415e-02,
               -2.74565774e-02,   2.35266841e-02,  -1.03729886e-02,
                5.75076365e-04,   2.88175844e-04,   2.95996987e-04,
                1.24544896e-03,   2.05494847e-03,   9.01859832e-04,
               -3.92378250e-03,  -9.75340436e-03,  -6.19108982e-03,
                1.53625256e-02,   3.56642051e-02,   4.94753434e-03,
               -7.57090185e-02,  -6.92092284e-02,   1.16599844e-01,
                1.82624946e-01,  -1.84912892e-01,  -3.33618561e-01,
                3.79745430e-01,   4.21448745e-01,  -8.29950559e-01,
               -6.87530161e-02,   1.28249595e+00,  -1.19377460e+00,
               -3.78858174e-01,   2.05083716e+00,  -2.38494337e+00,
                1.09015591e+00,   1.04752403e+00,  -2.97238204e+00,
                4.02738650e+00,  -4.12043121e+00,   3.53454658e+00,
               -2.66353513e+00,   1.84079364e+00,  -1.28608684e+00,
```

```
 1.12300009e+00,  -1.41156109e+00,   2.15810555e+00,
-3.28457401e+00,   4.56103275e+00,  -5.53914010e+00,
 5.57429080e+00,  -4.05947652e+00,   9.27712602e-01,
 2.78937262e+00,  -4.96881813e+00,   3.66694379e+00,
 6.53276950e-01,  -4.19915561e+00,   2.92941274e+00,
 1.94355943e+00,  -3.75766905e+00,  -4.22605921e-01,
 3.52897479e+00,   2.35271203e-01,  -3.15350059e+00,
-1.07562283e+00,   2.33617997e+00,   2.29054353e+00,
-3.88540436e-01,  -2.30583605e+00,  -1.93265010e+00,
-1.85797390e-01,   1.38041087e+00,   1.95732456e+00,
 1.61977793e+00,   8.37386087e-01,   3.72948554e-02,
-5.60008353e-01,  -9.12468131e-01,  -1.06926911e+00,
-1.10043531e+00,  -1.06091317e+00,  -9.80155672e-01,
-8.64543287e-01,  -7.05040179e-01,  -4.87931283e-01,
-2.09127665e-01,   1.08785689e-01,   4.03184093e-01,
 5.75916164e-01,   5.32127259e-01,   2.53473937e-01,
-1.35513689e-01,  -3.90138924e-01,  -3.12219941e-01,
 3.07175328e-02,   2.89369167e-01,   1.76943795e-01,
-1.38921173e-01,  -2.02103996e-01,   5.52397092e-02,
 1.67864190e-01,  -4.36273559e-02,  -1.23800195e-01,
 6.72635796e-02,   6.61894111e-02,  -8.51795497e-02,
 6.79775050e-03,   5.64540842e-02,  -5.40021719e-02,
 1.27866925e-02,   2.36161056e-02,  -3.55547441e-02,
 2.74898406e-02,  -1.20959515e-02,  -1.18762148e-03,
 8.90277822e-03,  -1.15954945e-02,   1.12008171e-02,
-9.48004960e-03,   7.52380311e-03,  -5.82159889e-03,
 4.49164491e-03,  -3.47977862e-03,   3.31247022e-03,
-1.24735033e-03,   5.11852713e-04,  -1.75242600e-04,
 4.20562524e-05,  -5.79161379e-06,   2.96711066e-07]))
```

In [208]:
```
fw_len = 1024*size(cA) + size(phi)
fw_recon = zeros(fw_len, float_)

for k in range(size(cA)):
    fw_recon[(1024*k):(1024*k+size(phi))] = fw_recon[(1024*k):(1024*k+siz
        cA[k]*phi - cD[k]*psi

#We plot the two functions, one on top of the other, to see if they are c
tt = linspace(0,size(fw_recon)/512,size(fw_recon))
plot(tt,fw_recon/sqrt(2),tt,f(tt));
```



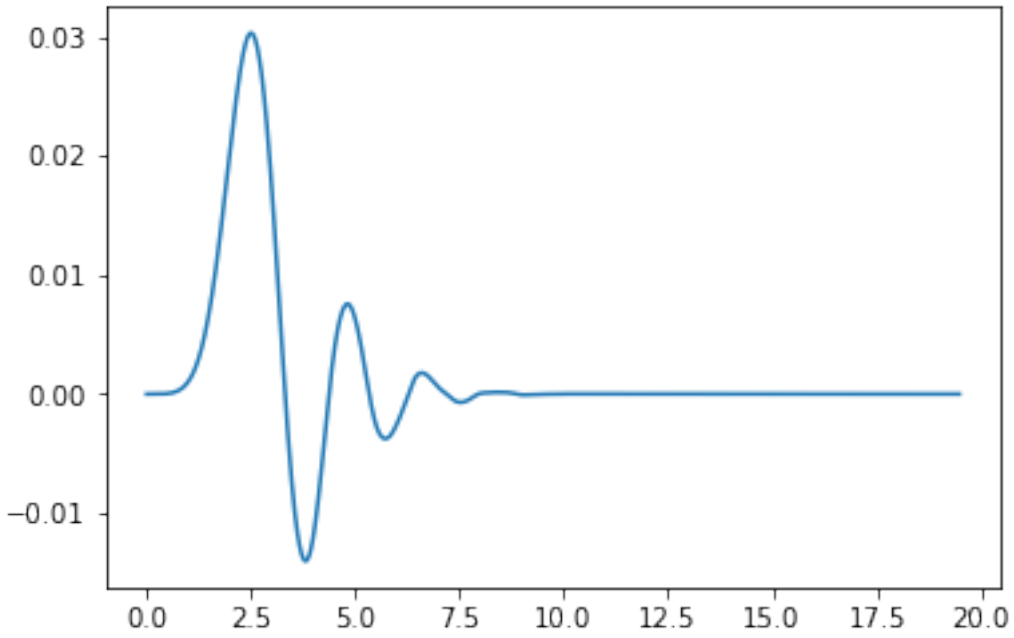
In [209]:
```
#We plot the two functions, one on top of the other, with one of them shi
tt = linspace(0,size(fw_recon)/512,size(fw_recon))
plot(tt,fw_recon/sqrt(2),tt,f(tt-14));
```



So, why 14? Maybe this has something to do with the wrap around and zero padding. The 14 might have to do with the fact that db8 has 16 coefficients. 14 = 16-2.

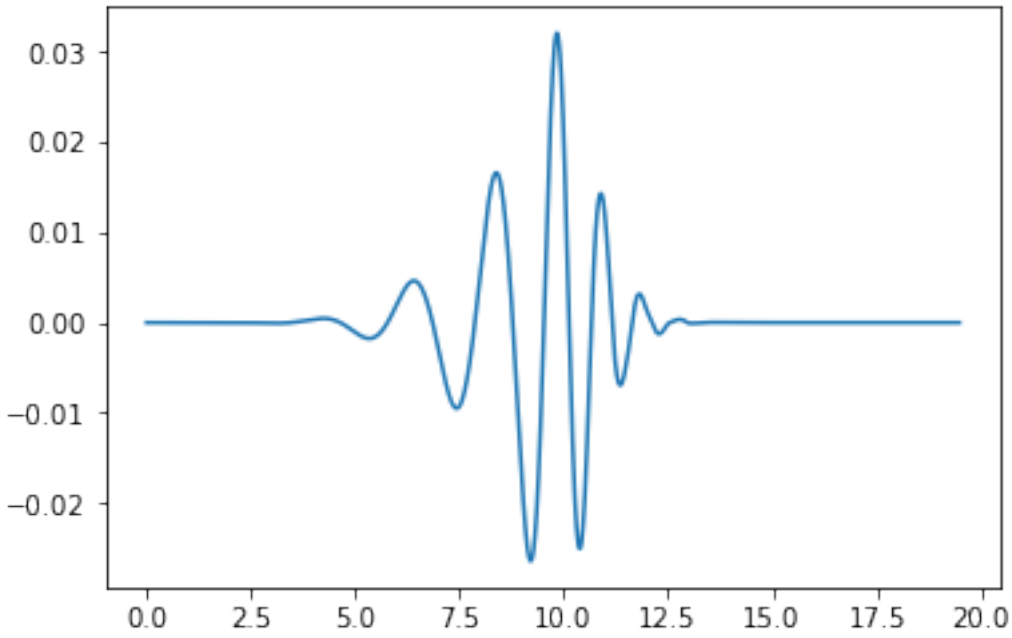
Let's try this with db10 to confirm.

```
In [217]: w = Wavelet('db10')
          h = w.dec_lo
          h = h[::-1]
          z = [1]
          for k in range(10):
              x = convolve(z,h)
              z = zeros(2*size(x))
              z[0:size(z):2] = x

          phi = x
          plot(linspace(0,size(phi)/1024,size(phi)),phi);
```




```
In [219]: w = Wavelet('db10')
          h = w.dec_lo
          h = h[::-1]
          g = w.dec_hi
          g = g[::-1]
          z = [-1]
          x = convolve(z,g)
          z = zeros(2*size(x))
          z[0:size(z):2] = x
```

```
for k in range(9):
    x = convolve(z,h)
    z = zeros(2*size(x))
    z[0:size(z):2] = x

psi = x
plot(linspace(0,size(psi)/1024,size(psi)),psi);
```

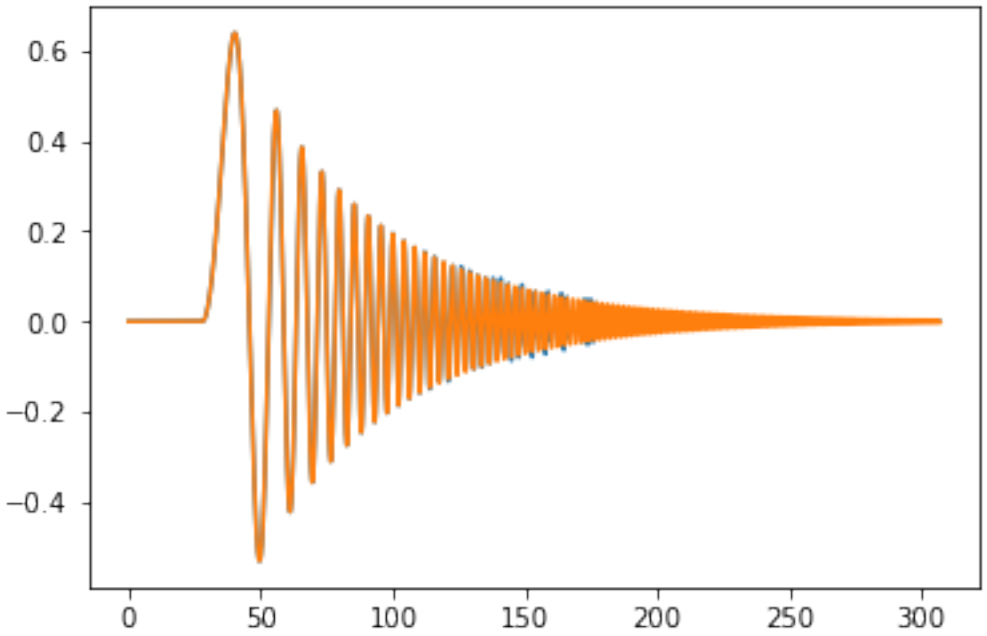


In [220]:
```
# Compute the initial scaling coefficient.

N = 250
y = zeros(N,float_)
t = linspace(0,size(phi)/1024,size(phi))
dt = 1/size(phi)
for k in range(N):
    y[k] = sum(f(t+k)*phi)
```

In [222]:
```
## Let's compute some wavelet coefficients

(cA, cD) = dwt(y, 'db10', 'zero')  # notice the keywork 'zero' to select
cA, cD;
```

In [225]:
```
## Plot the wavelet reconstrunction, and shift by 18 the original functi

fw_len = 1024*size(cA) + size(phi)
fw_recon = zeros(fw_len, float_)
```

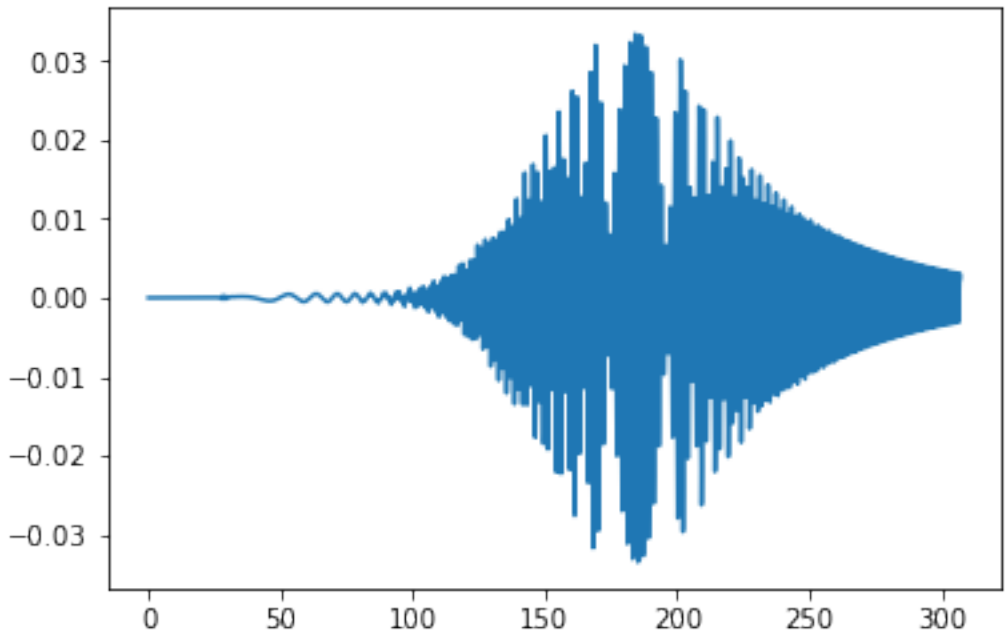
```
for k in range(size(cA)):
    fw_recon[(1024*k):(1024*k+size(phi))] = fw_recon[(1024*k):(1024*k+siz
        cA[k]*phi - cD[k]*psi

#We plot the two functions, one on top of the other, to see if they are c
tt = linspace(0,size(fw_recon)/512,size(fw_recon))
plot(tt,fw_recon/sqrt(2),tt,f(tt-18));
```



1.3.2 Verified! We had to shift by 18 = 20-2.

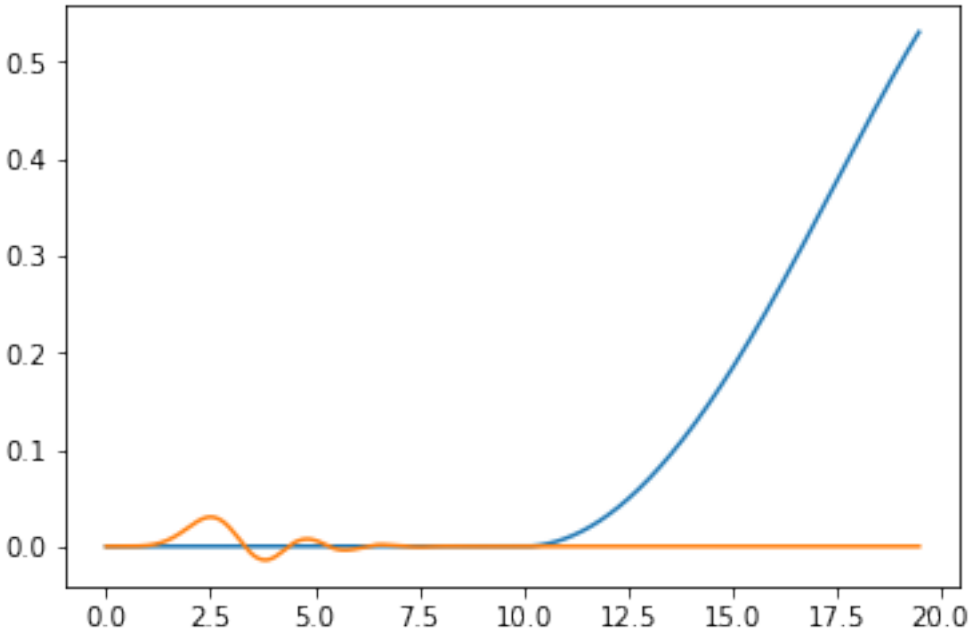
I don't know why. But it sure is interesting.

```
In [236]: #We plot the difference of the two functions. We see the error gets prett
          tt = linspace(0,size(fw_recon)/512,size(fw_recon))
          plot(tt,fw_recon/sqrt(2)-f(tt-18));
```




```
In [226]: ## Let's explore orthogonality

          phiOne = zeros(size(phi)+1024);
          phiTwo = zeros(size(phi)+1024);

          phiOne[0:size(phi)] = phi
          phiTwo[1024:(1024+size(phi))] = phi

In [227]: sum(phiOne*phiTwo)

Out[227]: 3.5182711567141243e-17

In [228]: f(9)

Out[228]: 0.0

In [152]: f(10)

Out[152]: 0.0

In [153]: f(11)

Out[153]: 0.0080250542271605545

In [154]: f(12)

Out[154]: 0.031456724416707632
```

```
In [195]: y[1]

Out[195]: -2.6134435921700234e-07

In [229]: t = linspace(0,size(phi)/1024,size(phi))
          plot(t,f(t),t,phi)

Out[229]: [<matplotlib.lines.Line2D at 0x10cc32978>,
           <matplotlib.lines.Line2D at 0x10cc32828>]
```




```
In [230]: sum(f(t)*phi)

Out[230]: -1.2604318363800155e-06

In [231]: y[0]

Out[231]: -1.2604318363800155e-06

In [232]: y[0]

Out[232]: -1.2604318363800155e-06

In [233]: cA[0]

Out[233]: -2.3460683354070367e-10

In [234]: y[0:16]
```

```
Out[234]: array([ -1.26043184e-06,   8.79394675e-06,  -2.24220372e-05,
                  -8.84242168e-06,   2.39320476e-04,  -8.71779364e-04,
                   1.94036289e-03,  -2.66437928e-03,  -7.00305779e-03,
                   3.27418006e-01,   1.13871118e+00,   2.40698919e+00,
                   4.09458596e+00,   6.15142734e+00,   8.50680512e+00,
                   1.10603837e+01])

In [ ]:
```