In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

On the irregular time sampling

Sampling times

In [117]:
```
time = np.arange(0,1,1/512) # regular sampling times
itime = time+(np.random.rand(len(time))-.5)*.00024 # irregular samp
ling times
```

In [169]:
```
f = 150.125 # signal frequency (more realistic if not exactly multi
ple of fs/N)
amp = 1 # signal amplitude

rsin = amp*np.sin(2*np.pi*f*time) # sinusoid regularly sampled
isin = amp*np.sin(2*np.pi*f*(itime)) # sinusoid irregularly sampled
SNR = np.sqrt(sum(rsin**2)/sum((rsin-isin)**2))
print('SNR: ' + str(SNR))
```

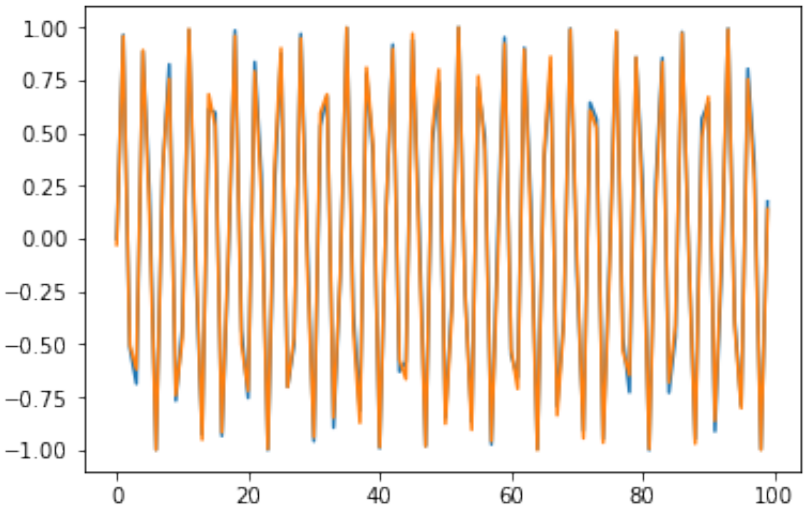
```
SNR: 15.2315789898
```

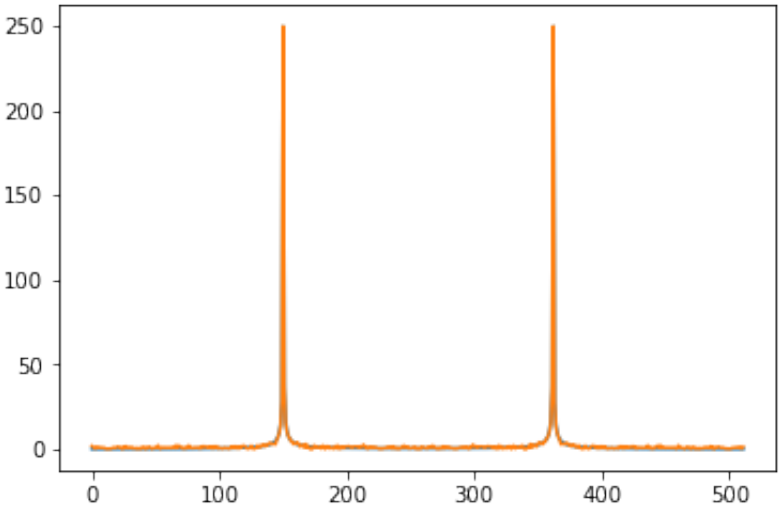
In [170]:
```
noise = 1/SNR*amp*np.random.randn(len(time)) # some noise for later
```

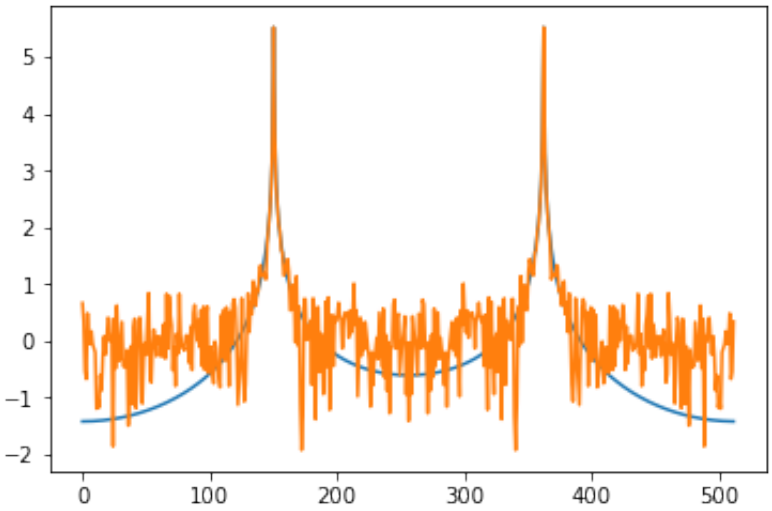
So it's a small effect even at 150 Hz, now let's see if it is too wierd looking...

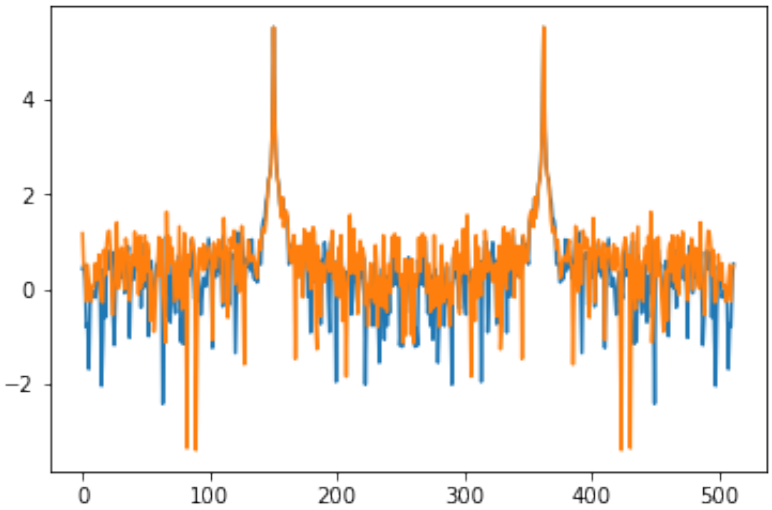
In [171]:
```
plt.figure()
plt.plot(rsin[:100])
plt.plot(isin[:100])
plt.figure()
plt.plot((abs(np.fft.fft(rsin))))
plt.plot((abs(np.fft.fft(isin))))
plt.figure()
plt.plot(np.log(abs(np.fft.fft(rsin))))
plt.plot(np.log(abs(np.fft.fft(isin))))
plt.figure()
plt.plot(np.log(abs(np.fft.fft(rsin+noise))))
plt.plot(np.log(abs(np.fft.fft(isin+noise))))
```

Out[171]:
```
[<matplotlib.lines.Line2D at 0x11993ef98>]
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80
100


250
200
150
100
50
0
0
100
200
300
400
500


5
4
3
2
1
0
−1
−2
0
100
200
300
400
500

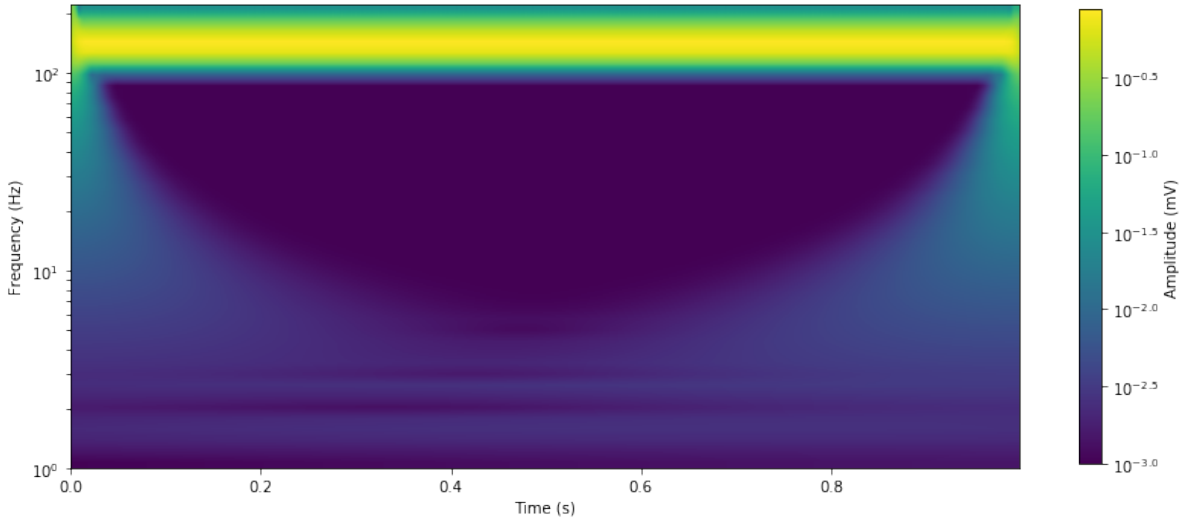


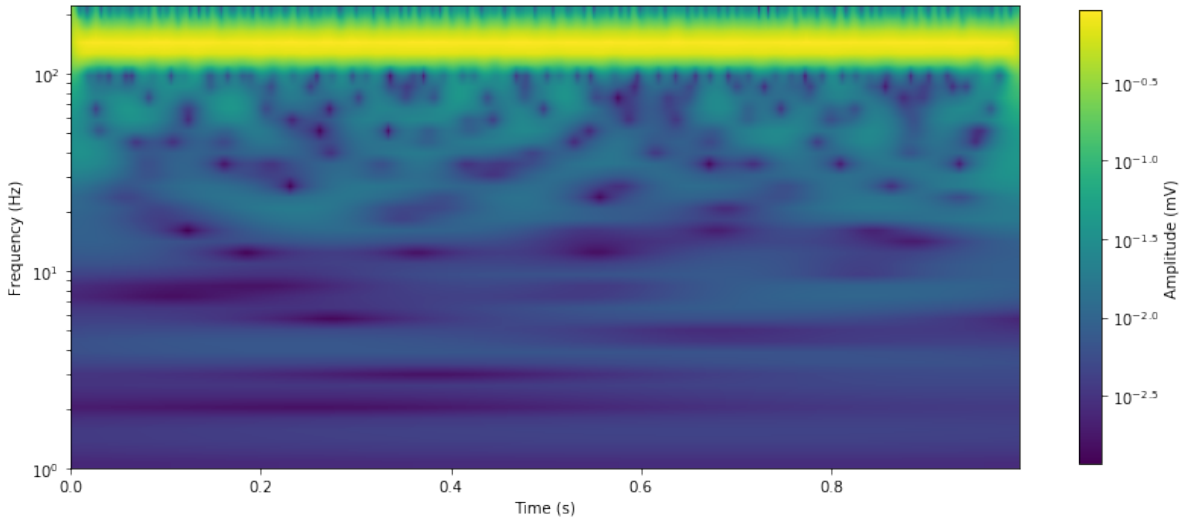
Doesn't look too bad... just a final check:

In [172]:
```
import scipy.signal as sg

def morlet_wavelet(input_signal,dt=1,R=7,freq_interval=(),drawplot=
1,eps=.0001,quick = False):
    Ns = len(input_signal)
    try:
        minf = max(freq_interval[0],1/(Ns*dt)) # avoid wavelets wit
h periods longer than the signal
    except:
        minf = 1/(Ns*dt)
    try:
        maxf = min(freq_interval[1],.5/dt) # avoid wavelets above t
he Nyquist frequency
    except:
        maxf = .5/dt
    try:
        Nf   = freq_interval[2]
    except:
        Nf = int(np.ceil(np.log(maxf/minf)/np.log(1/R+1))) # make s
pacing aproximately equal to sigma f

    alfa = (maxf/minf)**(1/Nf)-1; # According to the expression ach
ived by fn = ((1+1/R)^n)*f0 where 1/R = alfa
    vf   = ((1+alfa)**np.arange(0,Nf))*minf;
    result = np.zeros((Nf,Ns),dtype='complex')


    # These for loops reaaally should be paralelied...
    if quick:
         for k in range(Nf):
            result[k,:] = exp_filter(input_signal,vf[k],dt,R) # use
the IIR filter exponetial wavelet instead of Morlet
    else:
```

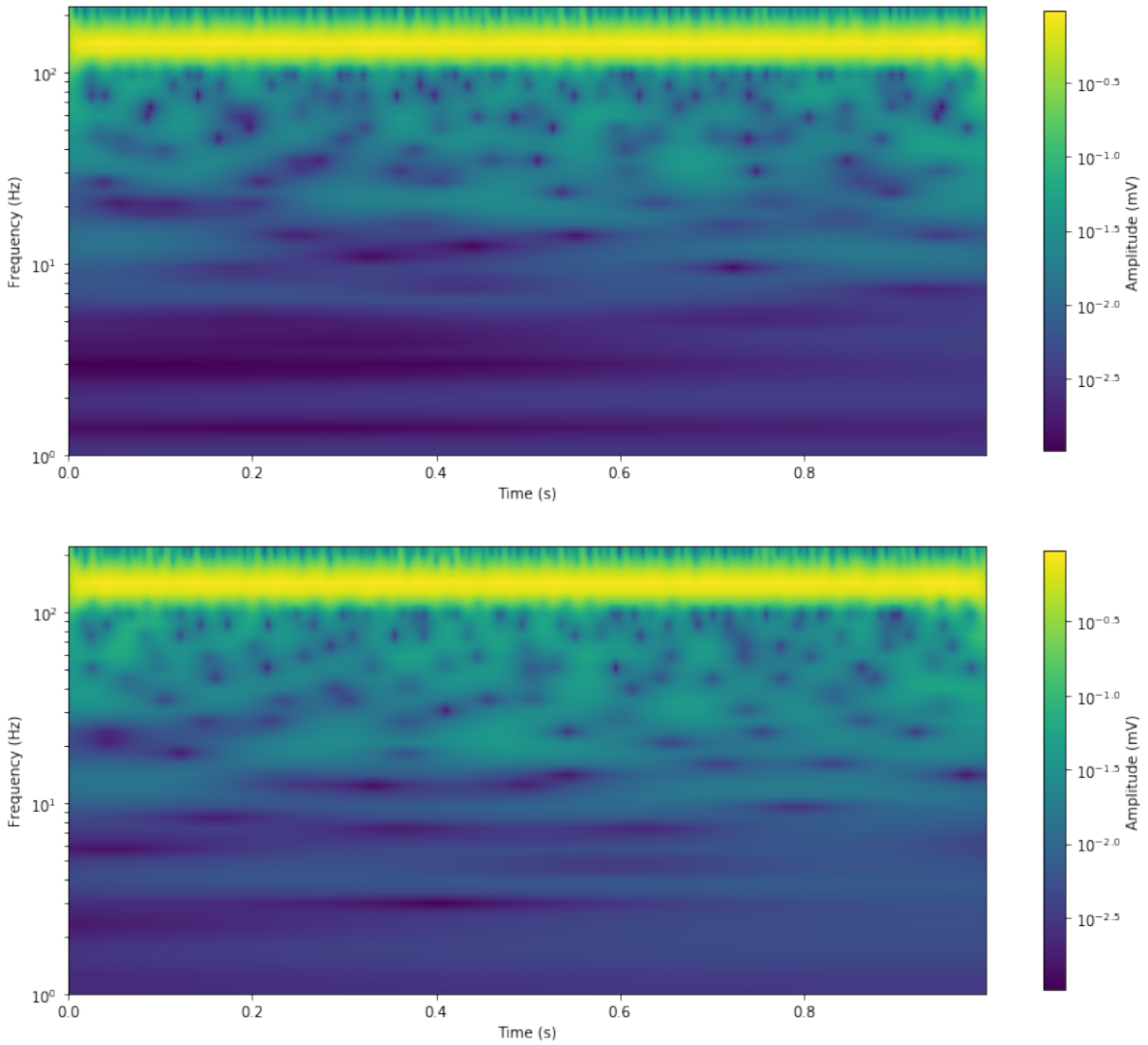
```
        for k in range(Nf):
            N =  int(2*R/vf[k]/dt) # Compute size of the wavelet
            wave = sg.morlet(N,w=R,s=1,complete=0)/N*np.pi*2 # Norm
alize de amplitude returned by sg.morlet
            result[k,:] = sg.fftconvolve(input_signal,wave,mode='sa
me')

    if drawplot: # beautiful plots for matplotlib... to bad pyecog
uses pyqtgraph! X'(

        exp_range_base10 = range(int(np.floor(np.log10(minf))),int(
np.floor(np.log10(maxf)+1)))
        base10_ticks = [[10**i+j*10**i for j in range(9)] for i in
exp_range_base10]
        base10_ticks = np.reshape(base10_ticks,-1)
        vf_base10 = ( np.log(base10_ticks) - np.log(vf[0]))/np.log(
1+alfa)
        labels  = ['$10^{'+str(i)+'}$' for i in exp_range_base10]
        labels10 = ['']*len(base10_ticks)
        for i in range(len(labels)):
                    labels10[i*9] = labels[i]

        plt.imshow(np.log10(2*np.abs(result)+eps),aspect=.5*dt*(Ns-
1)/Nf, interpolation='bilinear',extent = (0,dt*(Ns-1),Nf-1,0))
        plt.yticks(vf_base10,labels10)
        plt.ylim((0,Nf-1))
        plt.ylabel('Frequency (Hz)')
        plt.xlabel('Time (s)')

        cbar = plt.colorbar(shrink = .55)
        cbar_labels = cbar.ax.get_yticklabels()
        cbar_labelsText = [cbar_labels[i].get_text() for i in range
(len(cbar_labels))]
        cbar_labelsText10 = ['$10^{'+cbar_labelsText[i]+'}$' for i
in range(len(cbar_labels))]
        cbar.ax.set_yticklabels(cbar_labelsText10)
        cbar.set_label('Amplitude (mV)')

    return result
```

In [173]:
```
fig = plt.figure(figsize=(14,10))
morlet_wavelet(rsin, dt=1/512, R=7, freq_interval=(), drawplot=1, e
ps=.001, quick=False);
fig = plt.figure(figsize=(14,10))
morlet_wavelet(isin, dt=1/512, R=7, freq_interval=(), drawplot=1, e
ps=.001, quick=False);
```





In [174]:
```
fig = plt.figure(figsize=(14,10))
morlet_wavelet(rsin+noise, dt=1/512, R=7, freq_interval=(), drawplo
t=1, eps=.001, quick=False);
fig = plt.figure(figsize=(14,10))
morlet_wavelet(isin+noise, dt=1/512, R=7, freq_interval=(), drawplo
t=1, eps=.001, quick=False);
```



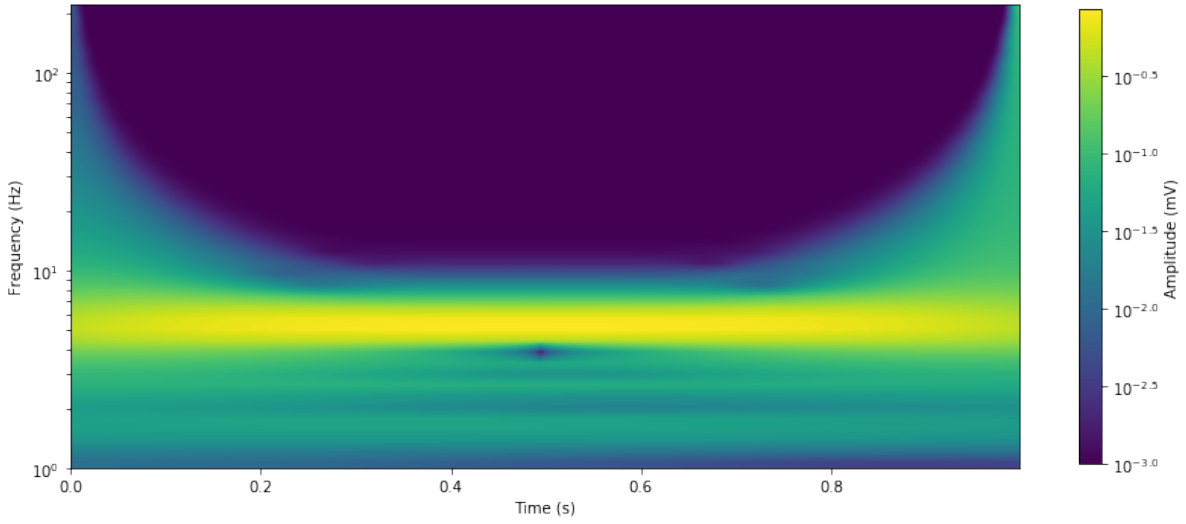
I'm fine with all this! - at first glance it doesn't look too different from a broadband white spectrum.

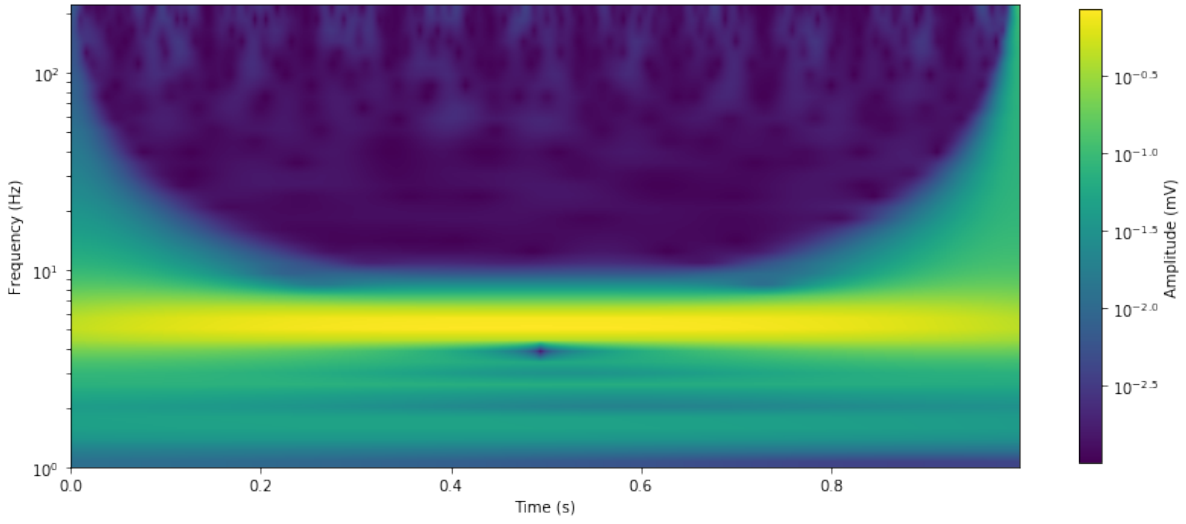
The effect I was a bit more woried about comes at lower frequencies, however, it's absolutely tiny! Here is an illustration:

In [175]:
```
f = 5.125 # signal frequency (more realistic if not exactly multiple of fs/N)
amp = 1 # signal amplitude

rsin = amp*np.sin(2*np.pi*f*time) # sinusoid regularly sampled
isin = amp*np.sin(2*np.pi*f*(itime)) # sinusoid irregularly sampled
SNR = np.sqrt(sum(rsin**2)/sum((rsin-isin)**2))
print('SNR: ' + str(SNR))
```

```
SNR: 448.367997159
```

In [176]:
```
fig = plt.figure(figsize=(14,10))
morlet_wavelet(rsin, dt=1/512, R=7, freq_interval=(), drawplot=1, eps=.001, quick=False);
fig = plt.figure(figsize=(14,10))
morlet_wavelet(isin, dt=1/512, R=7, freq_interval=(), drawplot=1, eps=.001, quick=False);
```





One can see the 10Hz modulation of the broadband noise at the zero crossings of the slower oscillation.

In [ ]: