# Vegetation Detection Challenge – Approach & Algorithm Summary Sheet – Masum Rab

**Powerline Bushfire Safety Program (PBSP) Vegetation Detection Challenge Background:**

In May 2017, the PBSP launched a Vegetation Detection Challenge as there is currently no mechanism that can identify vegetation faults in a timely manner and more specifically what type of species of vegetation is on/touching (causing a fault) a powerline. For example, if a tree or branch touches the line a device could detect the type of tree and send a message back to the distribution business as to whether there is potential for a fire start e.g. urgent action required or that the tree is a peppercorn (for example) and poses no risk, however in time it should be removed from the line.

**The Vegetation Detection Challenge aim was for the development of an algorithm that can identify what particular plant species is causing a fault if a tree branch were to fall onto a powerline.**

A large amount of Fault Signature detection data collected throughout the PBSP assisted teams with the development of the algorithm.

The Challenge focused on three particular plant species:

- Salix species (Willow) high fire probability
- Franxinus Angustifolia (Desert Ash) medium fire probability
- Schinus Molle (Peppercorn) low fire probability

A consolidation of the fault signature data for the above three species is available on the DataVic website - Vegetation Detection Challenge data. This consolidation removes the need to download the entire fault signature data set.

The full authorised data set including 300GB of photos, videos, test logs and report, are available on the DataVic website.

For a summary of Masum Rab and his teams' algorithm please see details in the below table:

| Summary of Approach & Algorithm |
| --- |
| • This team had a greater focus on the approach rather than the algorithm.<br>• The team had a deep understanding of the problem and the issues in relation to the real-world environment, where obtaining a relatively low-level vegetation signature from a very noisy environment will be a significant issue and therefore the team's main focus was on this aspect.<br>• The team put considerable thought into the options available to discover meaningful information in a very noisy environment. The team examined the following:<br>  ○ Phase relationship between voltage and current.<br>  ○ High frequency component resolved into active and reactive power.<br>  ○ Power spectral density.<br>  ○ Removal of background noise (whitened signal).<br>  ○ Detrended fluctuation analysis. |

    - Adaptive filtering of incoming signals and match filtering to determine if there is a potential branch present on the powerline.

- The team developed a multifaceted concept which incorporates adaptive filtering of the incoming signals and matched filtering. The team's analysis of the options and vision is very far reaching and has deep insight, however the implementation of the thought process would need to be considered further.

- The algorithm developed by this team only looked at one species (Willow). The team did not investigate accuracy measures for the species or related to the danger. However, the approach used by this team could be used in combination with the algorithms developed by teams one and two to apply to a real-world environment.

In [1]:
```
import matplotlib.mlab as mlab
```

In [243]:
```
import pandas as pd
import numpy as np
# from __future__ import print_function
import numpy as np
import matplotlib.pylab as plt
import padasip as pa
import glob
plt.style.use('ggplot') # nicer plots
np.random.seed(52102) # always use the same random seed to make results compara
ble
from scipy import signal
from scipy.io import loadmat
import warnings
warnings.filterwarnings('ignore')
%config InlineBackend.print_figure_kwargs = {}
%matplotlib inline
runsheet = pd.read_csv('./Data+import.csv')
import holoviews as hv
import datashader as ds

from holoviews.operation.datashader import aggregate, datashade, dynspread, sha
de
from holoviews.operation import decimate

print(__doc__)
```

```
Automatically created module for IPython interactive environment
```

In [244]:
```
import matplotlib.mlab as mlab
from scipy import interpolate
import scipy.integrate as integrate
from scipy.optimize import curve_fit
from scipy import interpolate
```

In [245]:
```
# Loading the dataset for the presentation
# number of samples for the fourier transform:
fs=1e5
NFFT = 4*fs
# data = loadmat('E:/DATA/presentationdata.mat')
data = loadmat('E:/DATA/data.mat')
cdata = loadmat('E:/DATA/complexPower.mat')

#What I need for the figures in this notebook?
# import./save of mat file
#

# data = loadmat('C:/Users/mrab/Dropbox/Analysis/presentationdata.mat')
```

In [247]:
```
## DATA for presentation

tLF=data['tLF'].flatten()
tHF=data['tHF'].flatten()

V =data['LFV'].flatten()
I =data['LFI'].flatten()
HFV =data['HFV'].flatten()
HFI =data['HFI'].flatten()
buf=int(0.65*len(tLF))

cV=cdata['cV']
cI=cdata['cI']
cHFI=cdata['cHFI']
cHFV=cdata['cHFV']
# cV=signal.hilbert(Ldf.LFV)
# cI=signal.hilbert(Ldf.LFI)

# cHFV=signal.hilbert(HFV)
# cHFI=signal.hilbert(HFI)

# complexPower=np.conj(cI)*(cV)
complexPower=cdata['LcP']
HcP=cdata['HcP']

Ldf=pd.DataFrame({'time': tLF.flat,'V':V.flat,'I':I.flat,'LcP':np.abs(LcP).flat
,'real':np.real(LcP).flat\
                  ,'imag':np.imag(LcP).flat})
Ldf.tLF=pd.to_timedelta(Ldf.time, unit='s')
Ldf =Ldf.set_index('time')

# Hdf=pd.DataFrame({'tHF': tHF,'HFV':HFV,'HFI':HFI,'HcP':np.abs(HcP),'real':np.
real(HcP),'real':np.imag(HcP)})
phi = np.angle(LcP)
dphi = np.diff(phi,axis=0)
Ldf.head()
```

Out[247]:

| | I | LcP | V | imag | real |
|---|---|---|---|---|---|
| **time** | | | | | |
| **0.000000** | 0.000000 | 16.136934 | 0.00000 | -1.674047e-12 | 16.136934 |
| **0.000007** | -0.002681 | 22.368140 | 14.20899 | -9.719821e+00 | 20.145937 |
| **0.000017** | -0.002681 | 17.036381 | 14.20899 | -9.721900e+00 | 13.990102 |
| **0.000027** | -0.002681 | 17.998256 | 14.20899 | -9.409945e+00 | 15.342430 |
| **0.000037** | -0.002681 | 16.318817 | 14.20899 | -9.411400e+00 | 13.331517 |

In [148]:
```
hv.extension('bokeh')
```

# Vegetation Detection Challenge

### High Quality Data set generated from test scenario

Video and photos of tests in Springvale. Raw Signal from Gen3i diagram and sound file.

### Implementation challenges on High Voltage Powerlines

Prohibitive cost of detection devices that need to achieve a detection in the presence of very low signal amplitude. Background of 22kV and >300A. Transmission lines have attenuation as lines behave as antennas radiating at different frequencies due to their geometry and real world constraints. This is changing all the time.

### Implementation challenges on SWER lines

- Similar issues regarding transmission attenuation. Signals from events will be more prominent however the load balancing issues make it more difficult to stabilise a reading.

---

# Strategy to overcome low SNR: Matched filter

- Used in Gravitational Astronomy for gravity wave detection, detecting a oscillatoin the size of an atom with respect to the distance between us and the sun.
- Just won the Nobel Prize for physics!
- What is Matched Filtering and why is it Optimal?
- Getting a frequency template form the Test Data set. Surprising challenge as the data set is so clean. Using Calibration runs for statistical properties of detectors. Does not necessarily tell us about real world.

# Characterizing noise in the detector

You need to understand the noise in your detector when there are no signals, to know for sure if you've actually measured a signal or just noise that happens to look like a signal.

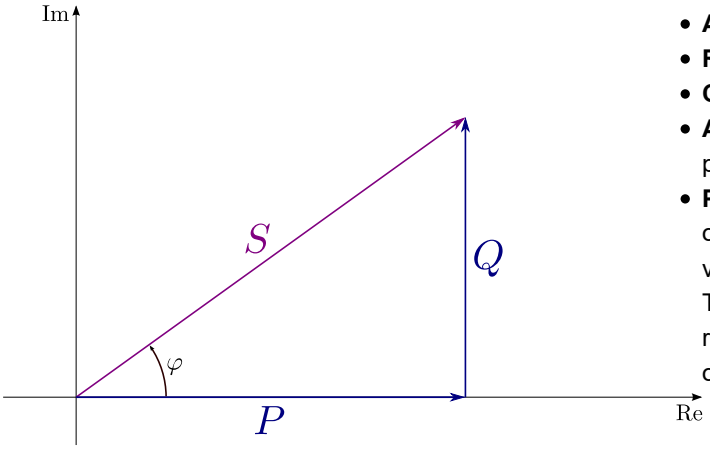
## Measure the background noise level.

- Willow (Salix Sp.)
- Test 830 (phase to phase fault)
- Test 540 (phase to earth)

# Test 830: Willow Phase to Phase fault

## Power plotted as complex numbers

Here we are using the convention $P = V \times conj(I)$.



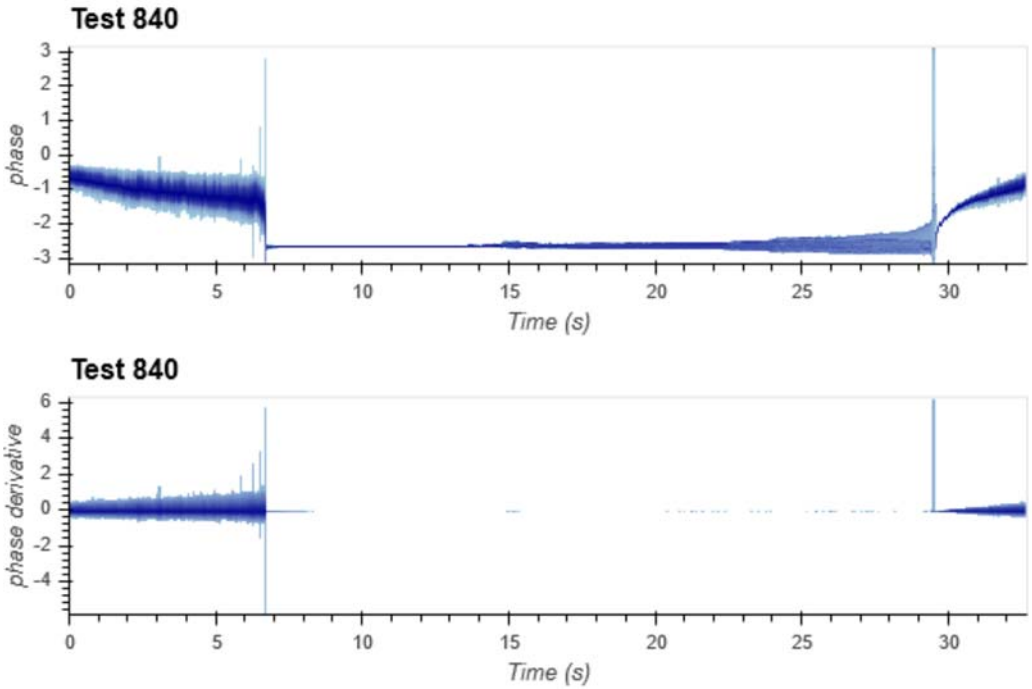

- **Active power**, P, or real power: *watt* (W)
- **Reactive power**, Q: *volt-ampere reactive* (var)
- **Complex power,S**: *volt-ampere (VA)*
- **Apparent power**, |S|: the magnitude of complex power S: *volt-ampere (VA)*
- **Phase of voltage relative to current**, $\phi$: the angle of difference (in degrees) between current and voltage;
  The complex power is the vector sum of active and reactive power. The apparent power is the magnitude of the complex power.

## Phase of voltage to current $\angle\phi = \angle V - \angle I$ and $\frac{d\phi}{dt}$

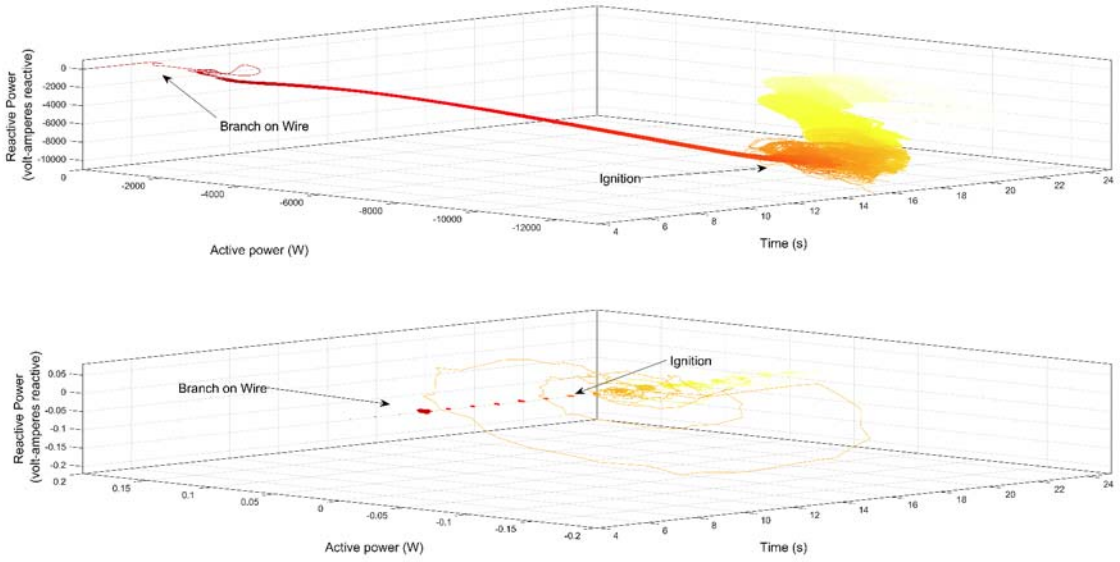
In [211]:
```
%%opts RGB  [height=200 width=600 tools=['hover']]
(datashade(p)+datashade(dp)).cols(1)
```

Out[211]:

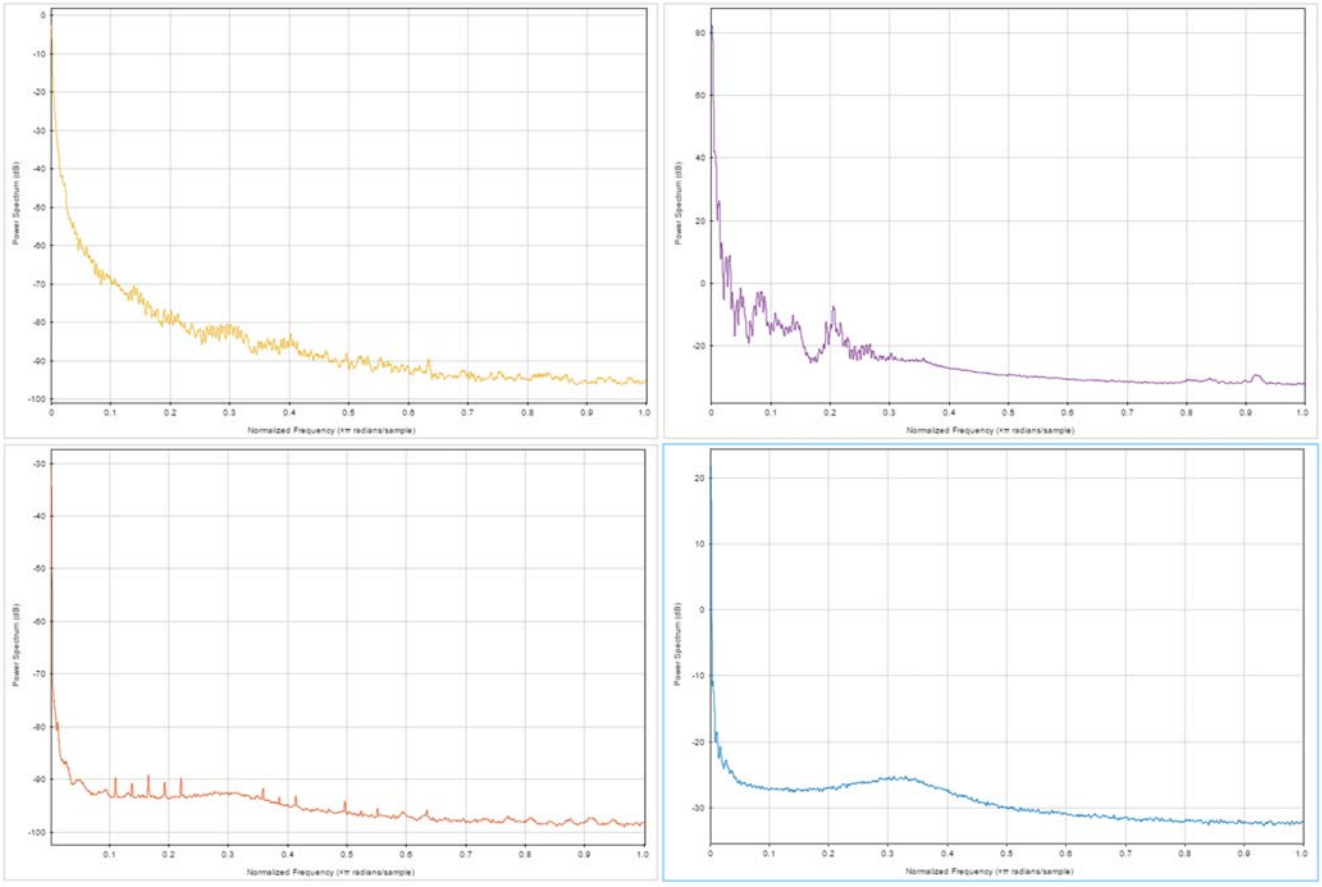
(https://b



### Low and High frequency components plotted as complex numbers




## Subsection corresponding to fault



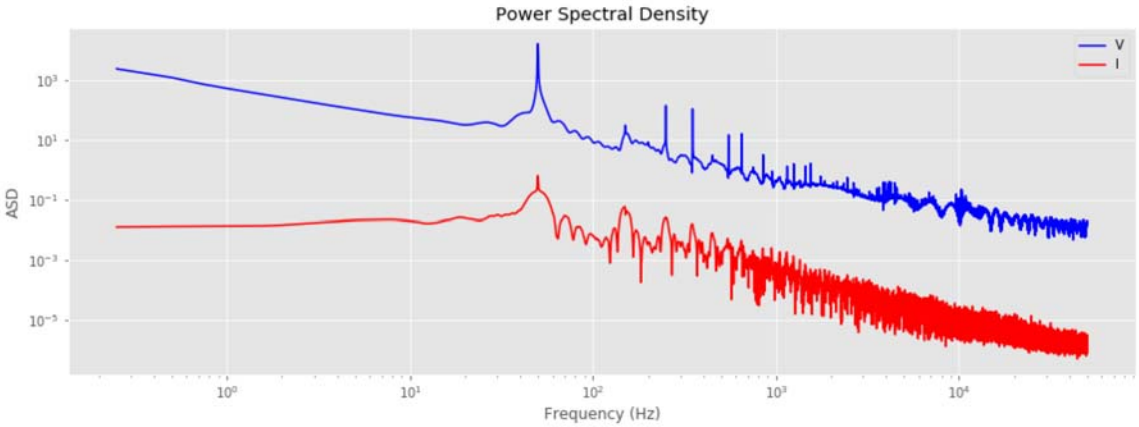
## Looking at the noise: the frequency domain

You may be familiar with the Fourier transform, which lets us write our noise function $n(t)$ as a sum of sines with different sizes, with all the frequencies that could be in the data:

In [229]:
```
# number of sample for the fast fourier transform:
fs=int(1e5)
NFFT = 4*fs

Pxx_V, freqs = mlab.psd(V, Fs = fs, NFFT = NFFT)
Pxx_I, freqs = mlab.psd(I, Fs = fs, NFFT = NFFT)

# We will use interpolations of the ASDs computed above for whitening:
psd_V = interpolate.interp1d(freqs, Pxx_V)
psd_I = interpolate.interp1d(freqs, Pxx_I)
```

In [228]:
```
plt.figure(figsize=(15,5))
plt.loglog(freqs, np.sqrt(Pxx_V),'b',label='V')
plt.loglog(freqs, np.sqrt(Pxx_I),'r',label='I')
plt.grid('on')
plt.ylabel('ASD')
plt.xlabel('Frequency (Hz)')
plt.legend(loc='upper right')
plt.title('Power Spectral Density')
# plt.savefig(eventname+'_ASDs.'+plottype)
```

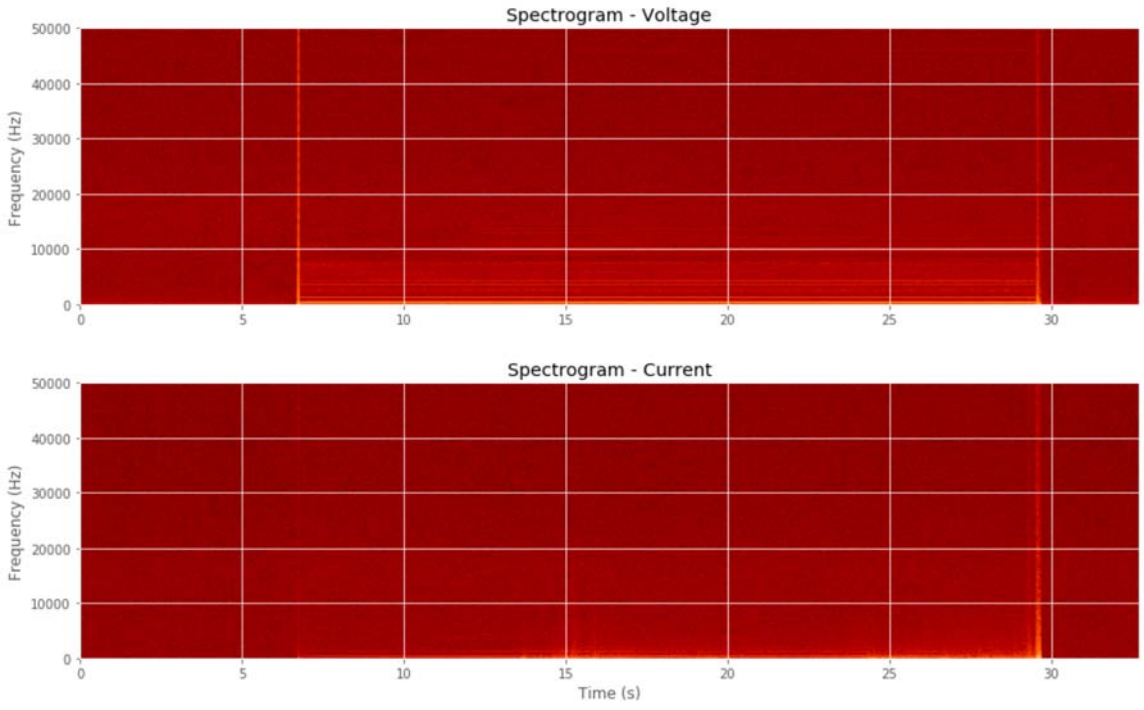
Out[228]: <matplotlib.text.Text at 0x2260ce8eb00>




## Spectrogram - frequency changes over time

In [231]:
```
# find signal and non signal segments
NFFT = int(fs/8)
# and with a lot of overlap, to resolve short-time features:
NOVL = int(NFFT*15./16)
# and choose a window that minimizes "spectral leakage"
window = np.blackman(NFFT)

spec_cmap='gist_heat'

# Plot the V spectrogram:
plt.figure(figsize=(15,4))
spec_V, freqs, bins, im = plt.specgram(V, NFFT=NFFT, Fs=fs, \
                                       window=window,noverlap=NOVL, cmap=spec_c
map, xextent=[0,tLF[-1]])
plt.title('Spectrogram - Voltage')
plt.axis([0, tLF[-1], 0, 50000])
# plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)')
spec_cmap='gist_heat'
# Plot the V spectrogram:
plt.figure(figsize=(15,4))
spec_I, freqs, bins, im = plt.specgram(I, NFFT=NFFT, Fs=fs, \
                                       window=window,noverlap=NOVL, cmap=spec_c
map, xextent=[0,tLF[-1]])
#     plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('Frequency (Hz)')
# plt.colorbar()
plt.axis([0, tLF[-1], 0, 50000])
plt.xlabel('Time (s)')
plt.title('Spectrogram - Current')
```

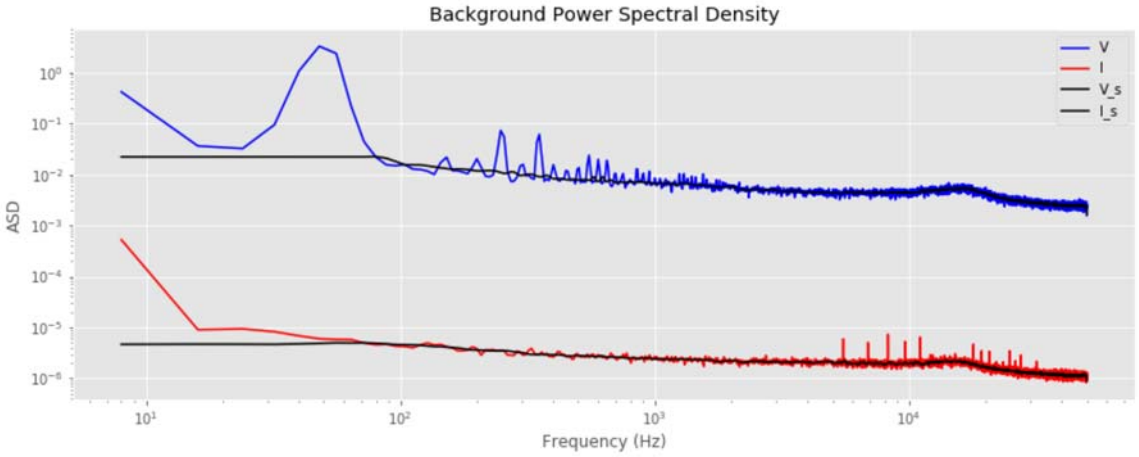
Out[231]: <matplotlib.text.Text at 0x225cf281be0>




## Background Signal

In [232]:
```
I_b = I[1:429550]
V_b = V[1:429550]

Pxx_V_b, freqs_V_b = mlab.psd(V_b, Fs = fs, NFFT = NFFT)
Pxx_I_b, freqs_I_b = mlab.psd(I_b, Fs = fs, NFFT = NFFT)

# We will use interpolations of the ASDs computed above for whitening:
psd_V_b = interpolate.interp1d(freqs_V_b, Pxx_V_b)
psd_I_b = interpolate.interp1d(freqs_I_b, Pxx_I_b)

PxxV_s =signal.medfilt(np.abs(Pxx_V_b),21)
PxxV_b =signal.savgol_filter(PxxV_s,5,2)

PxxI_s =signal.medfilt(np.abs(Pxx_I_b),21)
PxxI_b =signal.savgol_filter(PxxI_s,5,2)

psd_smooth_I_b =  interpolate.interp1d(freqs_I_b, PxxI_b)
psd_smooth_V_b =  interpolate.interp1d(freqs_V_b, PxxV_b)

plt.figure(figsize=(14,5))
plt.loglog(freqs_V_b, np.sqrt(Pxx_V_b),'b',label='V')
plt.loglog(freqs_I_b, np.sqrt(Pxx_I_b),'r',label='I')
plt.loglog(freqs_V_b, np.sqrt(PxxV_b),'k',label='V_s')
plt.loglog(freqs_I_b, np.sqrt(PxxI_b),'k',label='I_s')

plt.grid('on')
plt.ylabel('ASD')
plt.xlabel('Frequency (Hz)')
plt.legend(loc='upper right')
plt.title('Background Power Spectral Density')
```

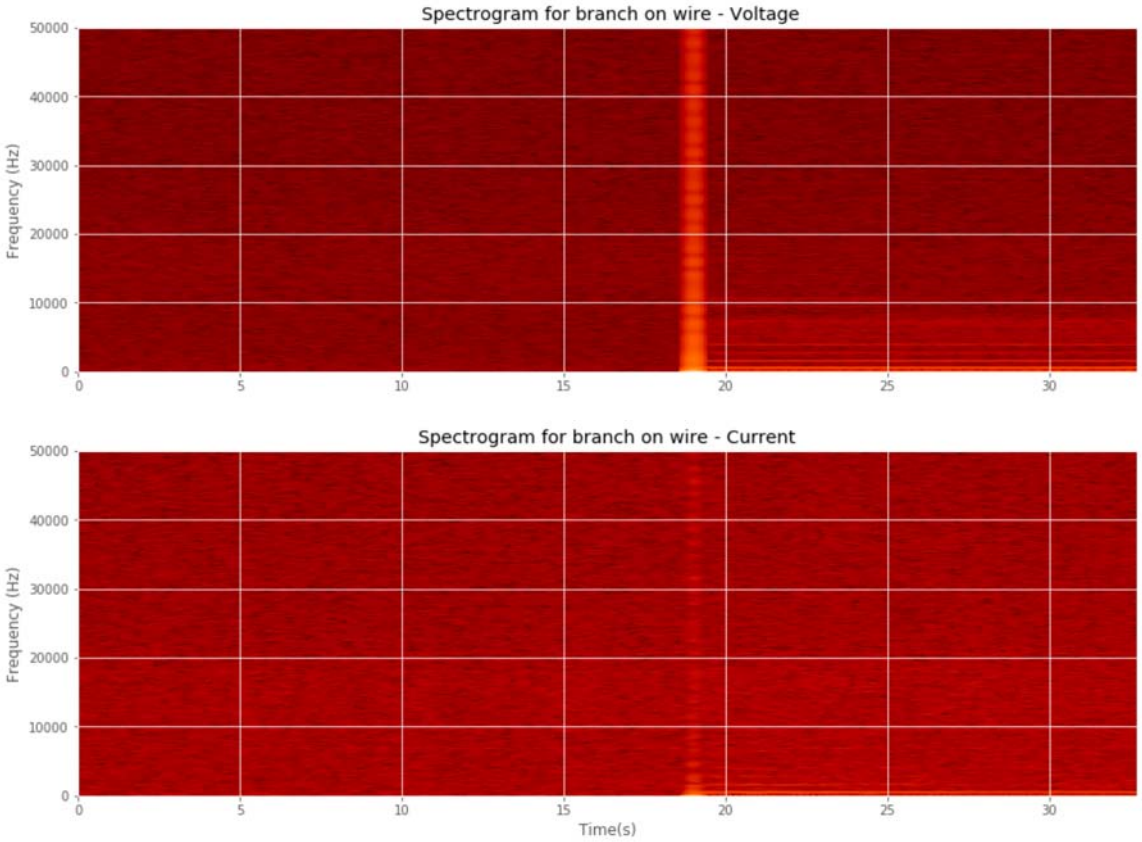
Out[232]: <matplotlib.text.Text at 0x2257069b080>




## Power specturm for first 4.2s of event

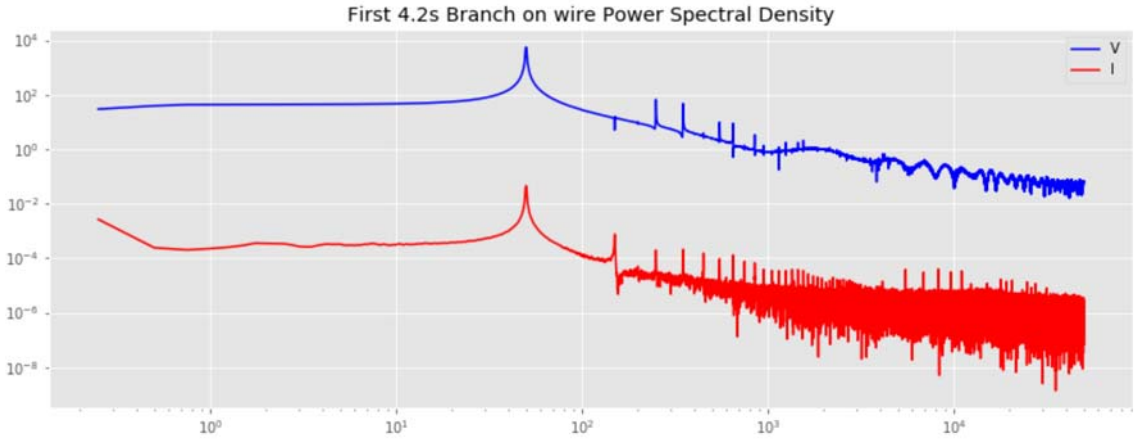
In [205]:
```
#4.2s slice
I_e = I[429550:859099]
V_e = V[429550:859099]

# I_e = I[429550:2579000]
# V_e = V[429550:2579000]

# find signal and non signal segments
NFFT = int(fs/8)
# and with a lot of overlap, to resolve short-time features:
NOVL = int(NFFT*15./16)
# and choose a window that minimizes "spectral leakage"
window = np.blackman(NFFT)

spec_cmap='gist_heat'
# Plot the V spectrogram:
plt.figure(figsize=(15,5))
spec_V_e, freqs, bins, im = plt.specgram(V_e, NFFT=NFFT, Fs=fs,\
                                         window=window,noverlap=NOVL, cmap=spec
_cmap, xextent=[0,tLF[-1]])
#     plt.xlabel('time (s) since '+str(tevent))
plt.title(' Spectrogram for branch on wire - Voltage')
p
lt.ylabel('Frequency (Hz)')
plt.axis([0, tLF[-1], 0, 50000])
# Plot the I spectrogram:
spec_cmap='gist_heat'
plt.figure(figsize=(15,5))
spec_I, freqs, bins, im = plt.specgram(I_e, NFFT=NFFT, Fs=fs,\
                                       window=window,noverlap=NOVL, cmap=spec_c
map, xextent=[-0,tLF[-1]])
plt.title('Spectrogram for branch on wire - Current')
plt.ylabel('Frequency (Hz)')
plt.xlabel('Time(s)')

## background plt.plot(I[1:400000])
```

Out[205]: <matplotlib.text.Text at 0x225b18ac860>



In [193]:
```
# number of sample for the fast fourier transform:
fs=int(1e5)
NFFT = 4*fs

Pxx_V_e, freqs_V_e = mlab.psd(V_e, Fs = fs, NFFT = NFFT)
Pxx_I_e, freqs_I_e = mlab.psd(I_e, Fs = fs, NFFT = NFFT)

# We will use interpolations of the ASDs computed above for whitening:
psd_V_e = interpolate.interp1d(freqs_V_e, Pxx_V_e)
psd_I_e = interpolate.interp1d(freqs_I_e, Pxx_I_e)

plt.figure(figsize=(14,5))
plt.loglog(freqs_V_e, np.sqrt(Pxx_V_e),'b',label='V')
plt.loglog(freqs_I_e, np.sqrt(Pxx_I_e),'r',label='I')
plt.title('First 4.2s Branch on wire Power Spectral Density')
# plt.loglog(freqs, np.sqrt(Pxx_s),'r',label='I')
# plt.grid('on')
# plt.ylabel('ASD')
# plt.xlabel('Freq (Hz)')
# plt.legend(loc='upper right')
# plt.savefig(eventname+'_ASDs.'+plottype)
plt.legend(loc='upper right')
```

Out[193]: <matplotlib.legend.Legend at 0x2255818c128>

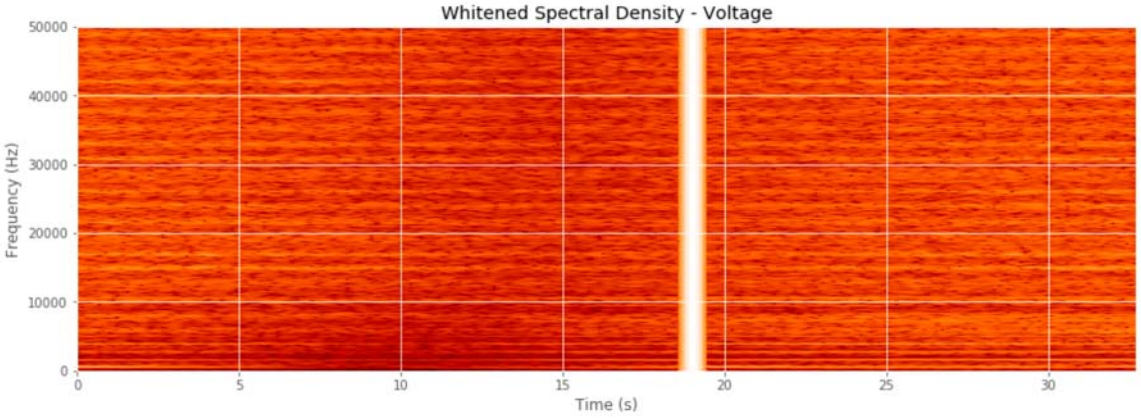



## Whiten Spectra

In [237]:
```
dt = 1e-5
# function to whiten data
def whiten(Measurement, interp_psd, dt):
    Nt = len(Measurement)
    freqs = np.fft.rfftfreq(Nt, dt)
#     freqs1 = np.linspace(0,2048.,Nt/2+1)
    # whitening: transform to freq domain, divide by asd, then transform back,
    # taking care to get normalization right.
    Mf = np.fft.rfft(Measurement)
    norm = 1./np.sqrt(1./(dt*2))
    white_Mf = Mf / np.sqrt(interp_psd(freqs)) * norm
    white_Mt = np.fft.irfft(white_Mf, n=Nt)
    return white_Mt

# now whiten the data from V and I, and the template (use signal PSD):
V_whiten = whiten(V_e,psd_V_e,dt)
I_whiten = whiten(I_e,psd_I_e,dt)
```

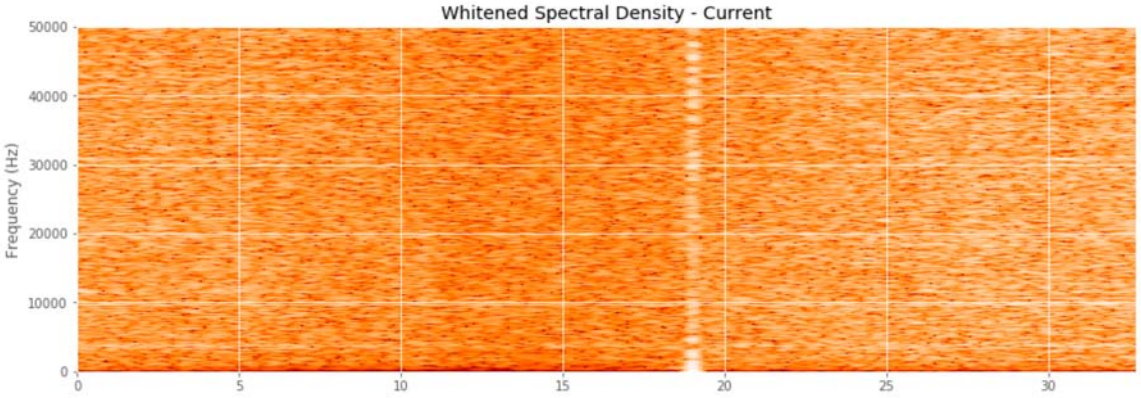
In [235]:
```
# deltat = 1e-5
NFFT = int(fs/8)
# and with a lot of overlap, to resolve short-time features:
NOVL = int(NFFT*15./16)
# and choose a window that minimizes "spectral leakage"
# (https://en.wikipedia.org/wiki/Spectral_leakage)
window = np.blackman(NFFT)

# the right colormap is all-important! See:
spec_cmap='gist_heat'
# Plot the V spectrogram:
plt.figure(figsize=(15,5))
spec_V_w, freqs_V_w, bins, im = plt.specgram(V_whiten, NFFT=NFFT, Fs=fs,\
                                             window=window,noverlap=NOVL, cmap=
spec_cmap, xextent=[0,tLF[-1]])
#     plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('Frequency (Hz)')
plt.xlabel('Time (s)')
# plt.colorbar()
plt.axis([0, tLF[-1], 0, 50000])
plt.title('Whitened Spectral Density - Voltage')
```

Out[235]: <matplotlib.text.Text at 0x2260b7a0c18>



In [236]:
```
# find signal and non signal segments

deltat = 1e-5
NFFT = int(fs/8)
# and with a lot of overlap, to resolve short-time features:
NOVL = int(NFFT*15./16)
# and choose a window that minimizes "spectral leakage"
# (https://en.wikipedia.org/wiki/Spectral_leakage)
window = np.blackman(NFFT)

# the right colormap is all-important! See:
spec_cmap='gist_heat'
# Plot the V spectrogram:
plt.figure(figsize=(15,5))
spec_I, freqs_I, bins, im = plt.specgram(I_whiten , NFFT=NFFT, Fs=fs, \
                                         window=window,noverlap=NOVL, cmap=spec
_cmap, xextent=[0,tLF[-1]])
#     plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('Frequency (Hz)')
# plt.colorbar()
plt.axis([0, tLF[-1], 0, 50000])
plt.title('Whitened Spectral Density - Current')
```

Out[236]:
```
<matplotlib.text.Text at 0x2260b83d080>
```




In [238]:
```
##
```

In [239]:
```
# V_whiten = whiten(V_whiten,psd_smooth_V_b,dt)
# I_whiten = whiten(V_whiten,psd_smooth_I_b,dt)

# Pxx_V_whiten, freqs = mlab.psd(V_whiten, Fs = fs, NFFT = NFFT)
# Pxx_I_whiten, freqs = mlab.psd(I_whiten, Fs = fs, NFFT = NFFT)
# plt.figure(figsize=(14,5))
# plt.loglog(freqs, np.sqrt(Pxx_V_whiten),'b',label='V')
# plt.loglog(freqs, np.sqrt(Pxx_I_whiten),'r',label='I')
# plt.legend(loc='upper right')
```

# Detrended Fluctuation Analysis(DFA)

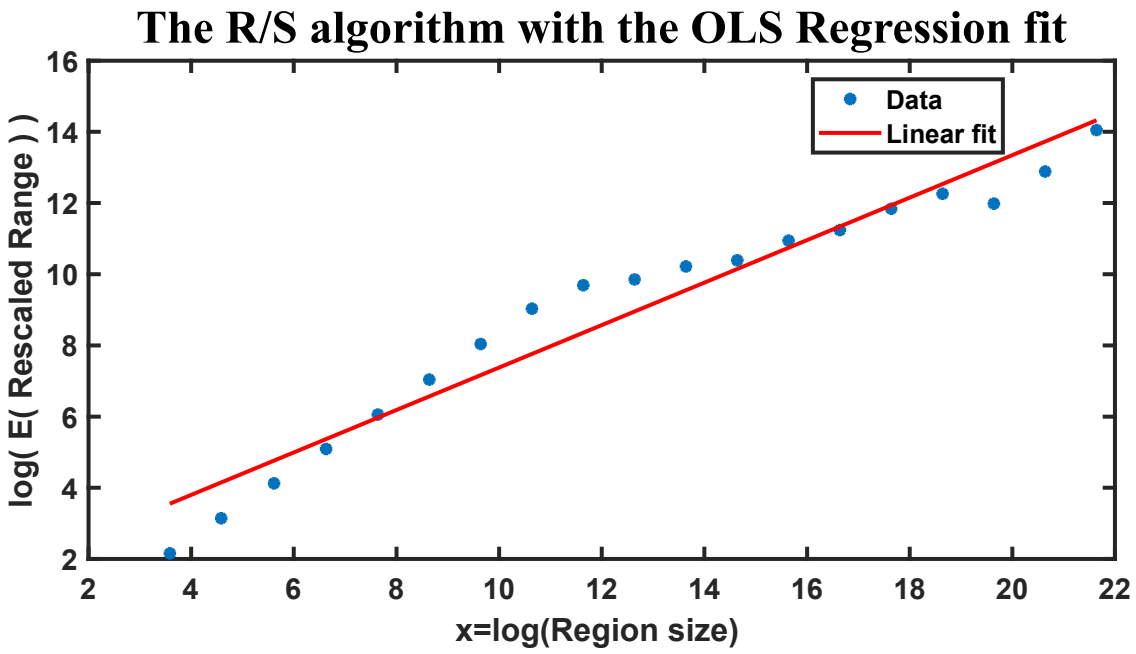
Presence of Long Range Order and fractal nature of signal characteristics. An equation? Very difficult to write down a model that can handle long term characteristics at the same time as accurately predicting sort term behaviour. Failure mode physics has fractal dynamics.

## Hurst Exponent.

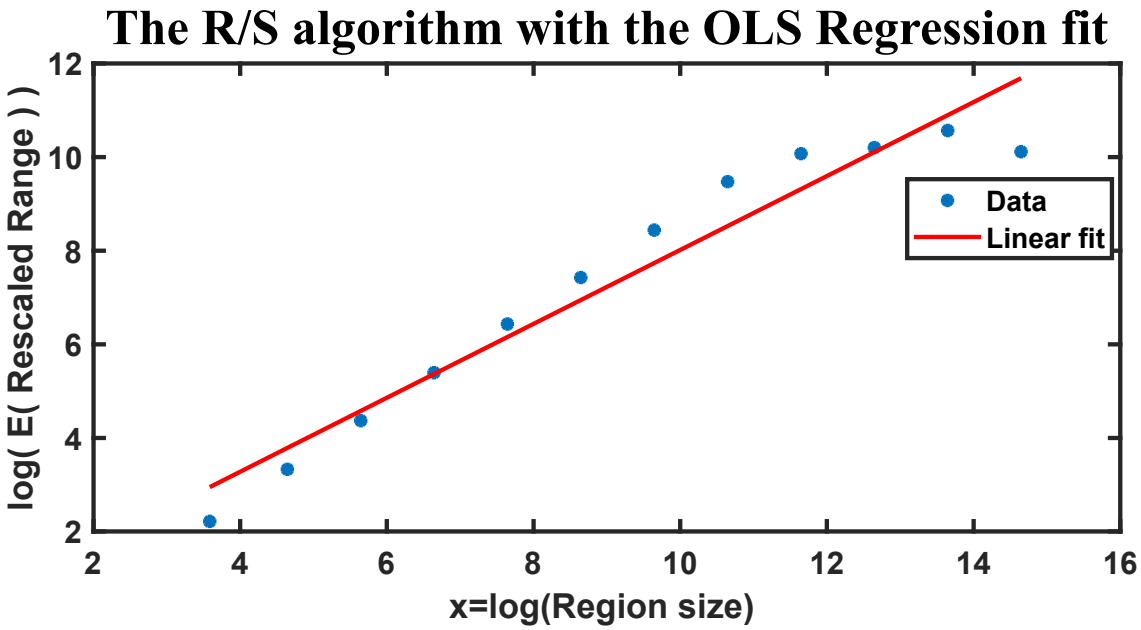
$$E\Big[\frac{R(n)}{S(n)}\Big] = Cn^H$$

$R(n)$ is the range of the first *n* values, and $S(n)$ is their standard deviation, $\mathrm{E}[x]$, is the expected value of *x*. *n* is the time span of the observation (number of data points in a time series) and *C* is a constant.

**Hurst exponent complete run h = 0.60**




**Hurst exponent during ramp up period run h = 0.80**



# Realtime Detection Algorithm

## Block Diagram of Algorithm

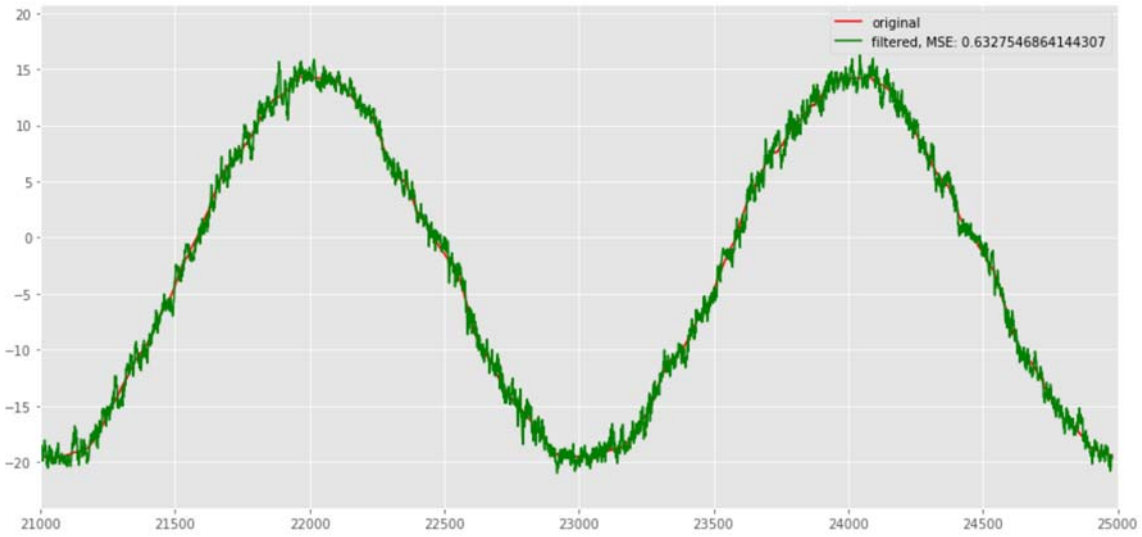
## Onset detection:

#### Phase locking for onset detection?

Maybe too noisy once the signal has to travel significant distances. Using thresholds with fixed values for Derivative and/or Cumulative sum as onset detection mechanisms. Simple strategies to check for events. Can tell us to check if there is a signal. Real world scenario is constantly changing.

---

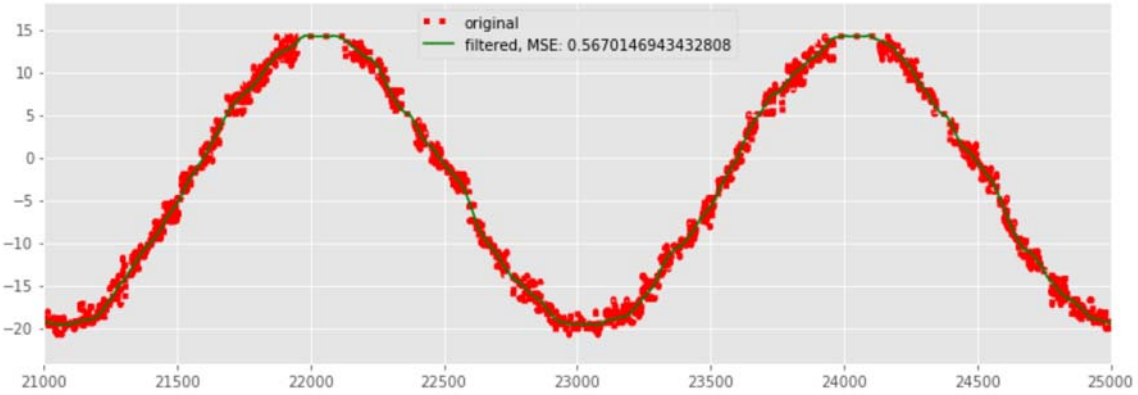
In [240]:
```
# filtering
x = pa.input_from_history(d, n)[:-1]
d = d[n:]
u = u[n:]
f = pa.filters.FilterRLS(mu=0.9, n=n)
y, e, w = f.run(d, x)

# error estimation
MSE_d = np.dot(u-d, u-d) / float(len(u))
MSE_y = np.dot(u-y, u-y) / float(len(u))

# results
plt.figure(figsize=(12.5,6))
plt.plot(u, "r", label="original")
plt.plot(y, "g", label="filtered, MSE: {}".format(MSE_y))
plt.xlim(N-4000,N)
plt.legend()
plt.tight_layout()
plt.show()
```



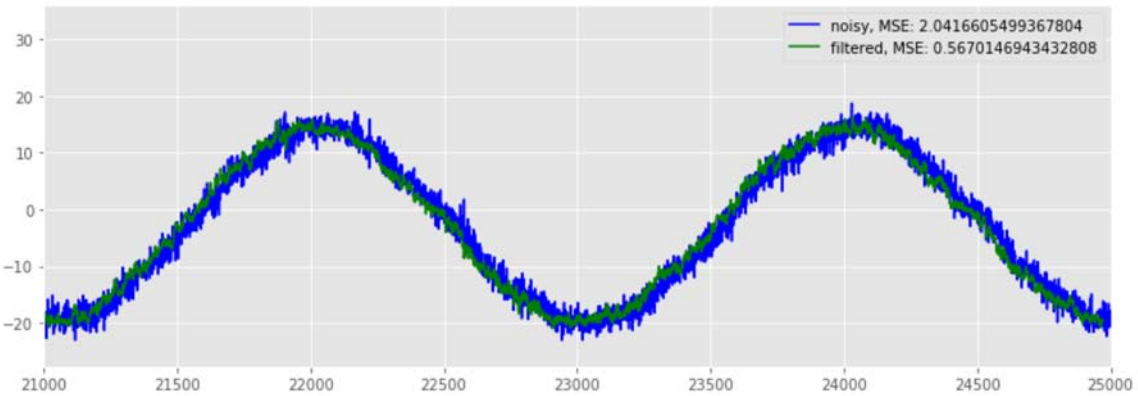

# Filtering Solution: Adaptive filters.

An example of an adaptive filter rejecting gaussian noise to return background from test measurements.

In [241]:
```
# filtering
x = pa.input_from_history(d, n)[:-1]
d = d[n:]
u = u[n:]
f = pa.filters.FilterRLS(mu=0.9, n=n)
y, e, w = f.run(d, x)

# error estimation
MSE_d = np.dot(u-d, u-d) / float(len(u))
MSE_y = np.dot(u-y, u-y) / float(len(u))

# signals creation: u, v, d
# Here I have selected a section of the signal without a branch touching
u1 = pd.Series.from_csv("background.csv").values
u = signal.savgol_filter(u1, 101, 2)
N = len(u)
n = 10
v = np.random.normal(0,1.2, N)
d = u1 + v

plt.figure(figsize=(11,4))
plt.plot(u1, "r:", linewidth=4, label="original")
plt.plot(u, "g", label="filtered, MSE: {}".format(MSE_y))
plt.xlim(N-4000,N)
plt.legend()
plt.tight_layout()
```




In [242]:
```
# results
plt.figure(figsize=(11,4))
plt.plot(d, "b", label="noisy, MSE: {}".format(MSE_d))
plt.plot(y, "g", label="filtered, MSE: {}".format(MSE_y))
plt.xlim(N-4000,N)
plt.legend()
plt.tight_layout()
plt.show()
```



# Conclusions and looking forward:

## Limited time so far. What are next steps of analysis:

- Finishing the template generation process and testing against artificial signals. What is the level of detail required to detect branch events with a given confidence level in a very noisy environment?
- Getting through the dataset. Big data and good data but how messy can I make the signal before it becomes unrecognisable?
- Work done already in EDA will allow for massive parallelisation as necessary.

- What are the difficulties around implementation of this approach and will it work on SWER lines?
- What are the useful aspects of my work for other teams? The aim is for them to be able to directly run this software for themselves.

## Thank you.