Simulation and Results

October 9, 2018

This is attempt to replicate and simulate the data and the results from the papers. The functions used are kf3 (a 3 state kalman filter), stalta(to implement stalta method) and modified_stalta(a modified version).

The STA/LTA window lengths were decided by hit-trial method

As mentioned in the literature, the wave form of the P wave or S wave can be approximated to a random walk, the analysis is performed over a simulated P wave, Random walk approximated wave and S wave. The sampling frequency is $20kHz$ and the P wave frequency is $700Hz$ and S wave frequency is $20Hz$.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from statsmodels.tsa.arima_process import arma_generate_sample
        #to generate the gauss markov stationary process -ARMA(1,1)
```

```
In [2]: def kf3(d1,A,B,Q,R,fs,f0):
            t = np.linspace(0,1-1/fs,fs)
            C = np.array([[1,0,1]])
            xhat = np.zeros([3,len(d1)])
            xhat_prior = np.zeros([3,len(d1)])
            P_ = []
            Ppost = []
            xhat[:,0] = np.ones(3)
            P0 = np.eye(3)
            Ppost.append(P0)
            omega = 2*np.pi*f0
            for i in range(0,len(d1)-1):
                xhat_prior[:,i] = A @ xhat[:,i]
                Ptemp = A @ P0 @ (A.T) + Q
                Ktemp = C @ Ptemp @ (C.T) + R
                i_Ktemp = np.linalg.inv(Ktemp)
                KGain = Ptemp @ C.T @ i_Ktemp
                ptr = C @ xhat_prior[:,i]
                xhat[:,i+1] = xhat_prior[:,i] + KGain @ (d1[i]- ptr )
                P_.append(Ptemp)
                P0 = (np.eye(3) - KGain @ C) @ Ptemp
                Ppost.append(P0)
                bb = 1*omega*np.cos(omega*(t[i+1]))/fs
                A = np.array([[1,bb , 0],[0,1,0],[0,0,0.75]])
```

```
        return xhat

    def stalta(dataset):
        LTA_window = 2000
        STA_window = 150
        temp1 = np.convolve((dataset)**2,np.ones(STA_window)/STA_window,mode='valid')
        temp2 = np.convolve((dataset)**2,np.ones(LTA_window)/LTA_window,mode='valid')
        temp1 = temp1[0:np.size(temp2)]
        return np.divide(temp1,temp2)

    def modified_stalta(dataset):
        d = stalta(dataset)
        dtemp = dataset[0:len(d)]
        return abs(dtemp)*d**3
```

The seismic wavelets are modeled by

$$x_1(t) = x_2(t)sin(\omega(t - t_0))$$

Considering the $x_2(t)$ as a random walk, $\dot{x_2}(t) = w(t)$ where $w(t)$ is the white noise. In the $x_1(t)$ differentiation we consider the $x_2(t)$ to be constant.

$$\dot{x_1}(t) = x_2(t)\omega cos(\omega(t)) \text{ if,} t_0 = 0$$

$$\dot{x_2}(t) = w(t)$$

$$\dot{x_3}(t) = -\beta x_3(t) + \sqrt{2\sigma^2\beta w(t)}$$

The best way for event detection turns out to be the amplitude of the x_2

Continuous time

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & \omega cos(\omega t) & \omega \\ 0 & 0 & 0 \\ 0 & 0 & -\beta \end{bmatrix}}_{F(t)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 \\ q(t) & 0 \\ 0 & \sqrt{2\sigma^2\beta} \end{bmatrix}}_{G(t)} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$Z(t) = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + v(t)$$

Discrete Time

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \end{bmatrix} = \begin{bmatrix} 0 & \Delta\omega cos(\omega\Delta(k)) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & e^{-\beta\Delta} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ q[k] & 0 \\ 0 & \sqrt{2\sigma^2\beta} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$z[k] = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \end{bmatrix} + v[k]$$

- In the papers, there is a slight confusion over the notation. The matrix preceeding w_1,w_2 below is called Q.

- However, it is also stated to be used as error covariance matrix of the State Variables. This is an error.
- The probable reason for this confusion could be, since the $w_1[k]$,$w_2[k]$ are white noise but, these act as input for driving the system.
- So essentially there's no covariance matrix for the error in the state variables.
- In the simulation, the state variables noise covariance had to be given manually

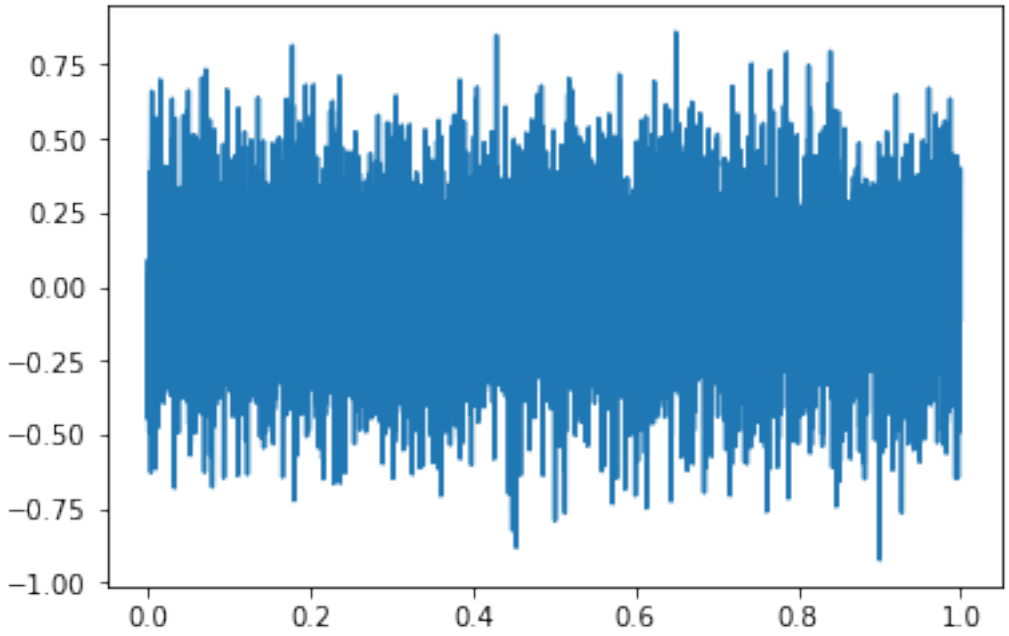
Now the P wave and S wave data is simulated

```
In [3]: fs = 20000
        f0 = 700
        f1 = 20

        omega = 2*np.pi*f0
        omega1 = 2* np.pi*f1

        t = np.linspace(0,1-0.00005,20000)
        t0 = t[2000]
        t1 = t[10000]
        B0 = 2
        A0 = 4
        h = 79
        h1 = 50

        A = A0 * np.exp(-h*(t-t0))*np.sin(omega*(t-t0))
        A1 = np.concatenate((np.zeros(2000),A[2000:]))
        B = B0 * np.exp(-h1*(t-t1))*np.sin(omega1*(t-t1))
        B1 = np.concatenate((np.zeros(10000),B[10000:]))
```

Now the random walk and the ambient noise is simulated. The ambient noise as mentioned is an ARMA(1,1) process. The standard deviation of the innovations is 0.1. The innovations of the random walk have the standard deviation 0.5. The mean for both are 0. And the measurement error noise standard deviation is considered to be 0.02. THe estimated time of arrival for the P wave and S wave are 0.1s and 0.5s respectively.

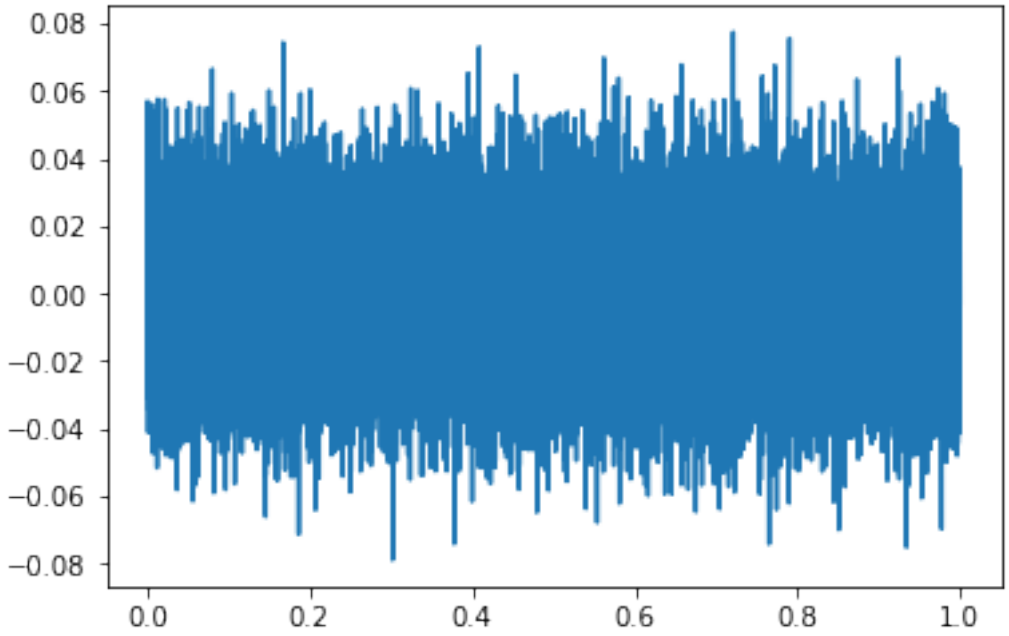
```
In [4]: np.random.seed(1234)
        mea1 = np.random.normal(0,0.5,1)
        dat = np.random.normal(0,0.5,999)
        dat1= np.concatenate((mea1,dat))
        dat1 = np.cumsum(dat1)
        dat = np.concatenate((dat1,np.mean(dat1)*np.ones(19000)))

        arparams = np.array([0.75])
        maparams = np.array([0.6])
        ar = np.r_[1, -arparams] # add zero-lag and negate
        ma = np.r_[1, maparams]
        gm_noise = arma_generate_sample(ar,ma,20000,sigma=0.1)
        plt.figure()
        plt.plot(t,gm_noise)
```

Out[4]: [<matplotlib.lines.Line2D at 0x7f9dde675438>]



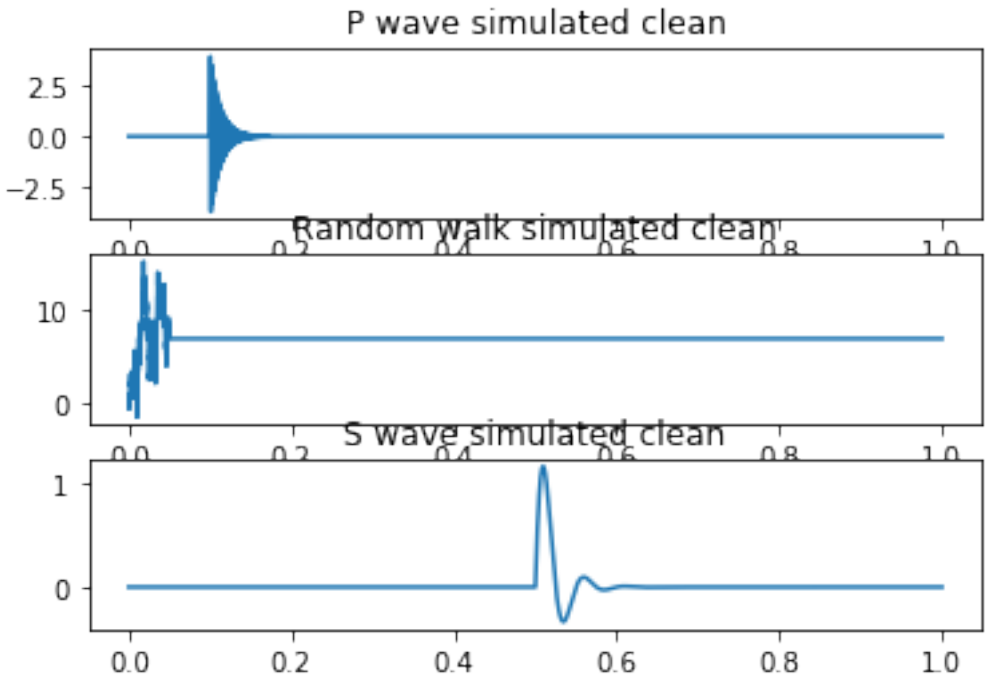
In [5]:
```
measurement_noise = np.random.normal(0,0.02,20000)
plt.figure()
plt.plot(t,measurement_noise)
```

Out[5]: [<matplotlib.lines.Line2D at 0x7f9dde608978>]



Making the data and the plot of the datasets considered for the analysis.

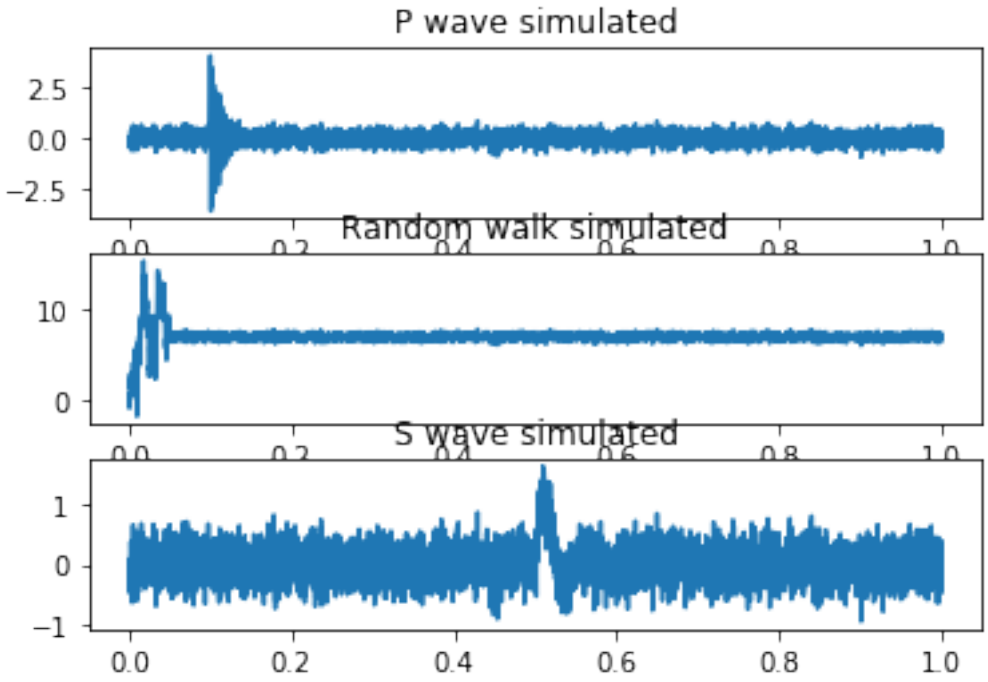
```
In [6]: l = ['P wave simulated clean','Random walk simulated clean','S wave simulated clean']
        plt.figure()
        plt.subplot(3,1,1)
        plt.plot(t,A1)
        plt.title(l[0])
        plt.subplot(3,1,2)
        plt.plot(t,dat)
        plt.title(l[1])
        plt.subplot(3,1,3)
        plt.plot(t,B1)
        plt.title(l[2])

Out[6]: Text(0.5,1,'S wave simulated clean')
```



The true arrival time for approximation of the P wave cannot be defined (Rather I don't know how to define it). So assuming the midpoint of the same which happens to be 0.025s.

```
In [7]: D = np.zeros([3,20000])
        D[0,:] = A1+gm_noise + measurement_noise
        D[1,:] = dat+gm_noise + measurement_noise
        D[2,:] = B1+gm_noise + measurement_noise
```

```
In [8]: l = ['P wave simulated','Random walk simulated','S wave simulated']
        plt.figure()
        for i in range(3):
            plt.subplot(3,1,i+1)
            plt.plot(t,D[i,:])
            plt.title(l[i])
```



As the 3-state space model for this problem is specified, we can use the state space model representation directly. The constraints have been given in the papers and accordingly the state space models are used for Kalman filtering. The results are plotted as well. Also the error variance of the states is not provided. **There is an abuse of the notation in the papers.** So the error covariance matrix has been manually given. Since a lot of the parameters are hardcoded and manually given, the results are not the best.

Making the constraint matrices. In the literature, Q is used as the error covariance matrix of the state variables. However while stating the state space model, the noise variables (which are white and were used to simulate the random walk and ambient noise) are considered to be the inputs. So there is no consideration for the noise in the state variables.

```
In [9]: A_1 = np.array([[1, 1*omega/fs, 0],[0,1,0],[0,0,0.75]])

        B_1 = np.array([[0,0],[0.5,0],[0,0.6]])

        Q = np.array([np.random.normal(0,0.01,20000),
                      np.random.normal(0,0.02,20000),np.random.normal(0,0.03,20000)])
        Q = np.matmul(Q,np.transpose(Q))/20000
```

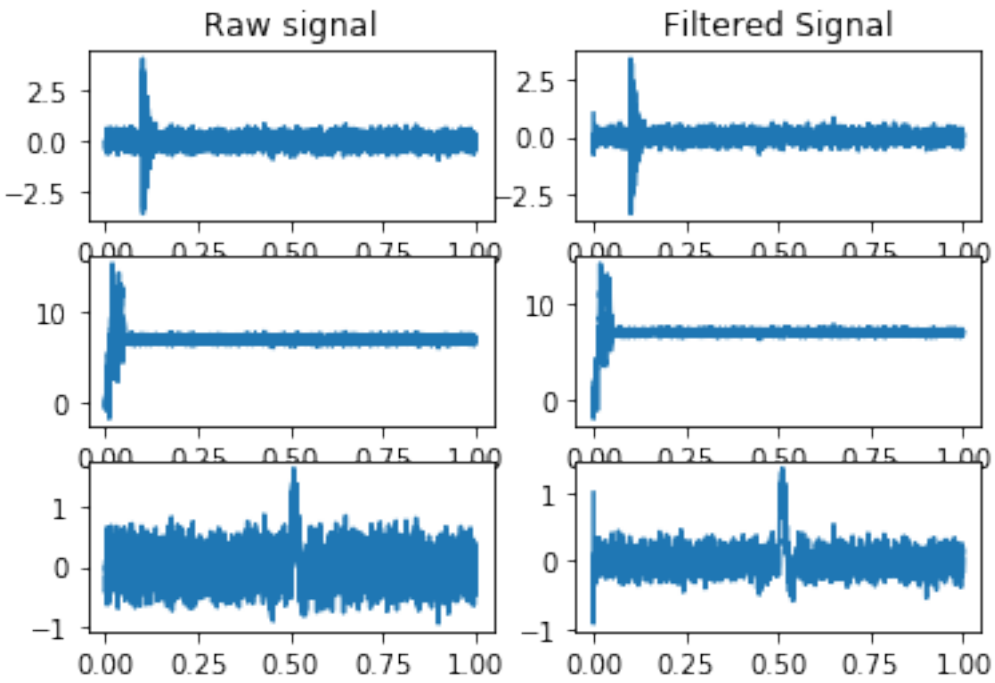
The B_1, B_2 and B_3 variables in the code are the said Q matrices in the paper, which is not a great way to state. The Q matrix considered is the error covariance matrix hardcoded.

```
In [10]: OP = np.zeros([3,3,20000]) #OP holds the estimated state variables for the three cases
         OP[0,:,:] = kf3(D[0,:],A_1,B_1,Q,np.var(measurement_noise),fs,f0)
         OP[1,:,:] = kf3(D[1,:],A_1,B_1,Q,np.var(measurement_noise),fs,f0)
         OP[2,:,:] = kf3(D[2,:],A_1,B_1,Q,np.var(measurement_noise),fs,f1)
```

There is no specific reason for considering the random walk with high frequency, but there is not much information given about how to estimate the random walk properties for P wave and S wave separately, apart from the variance of the generating innovations.

Plotting the filtered estimates followed by STA/LTA and modified STA/LTA.

```
In [11]: plt.figure()
         for i in range(3):
             plt.subplot(3,2,2*i+1)
             plt.plot(t,D[i,:])
             if(i==0):
                 plt.title('Raw signal')
             plt.subplot(3,2,2*i+2)
             plt.plot(t,OP[i,0,:])
             if(i==0):
                 plt.title('Filtered Signal')
```



STA/LTA method

$$stalta[i] = \frac{\sum_{k=i-l_1}^{i+l_2} \frac{(signal[k])^2}{(l_2+l_1)}}{\sum_{j=i-L_1}^{i+L_2} \frac{(signal[j])^2}{(L_2+L_1)}}$$

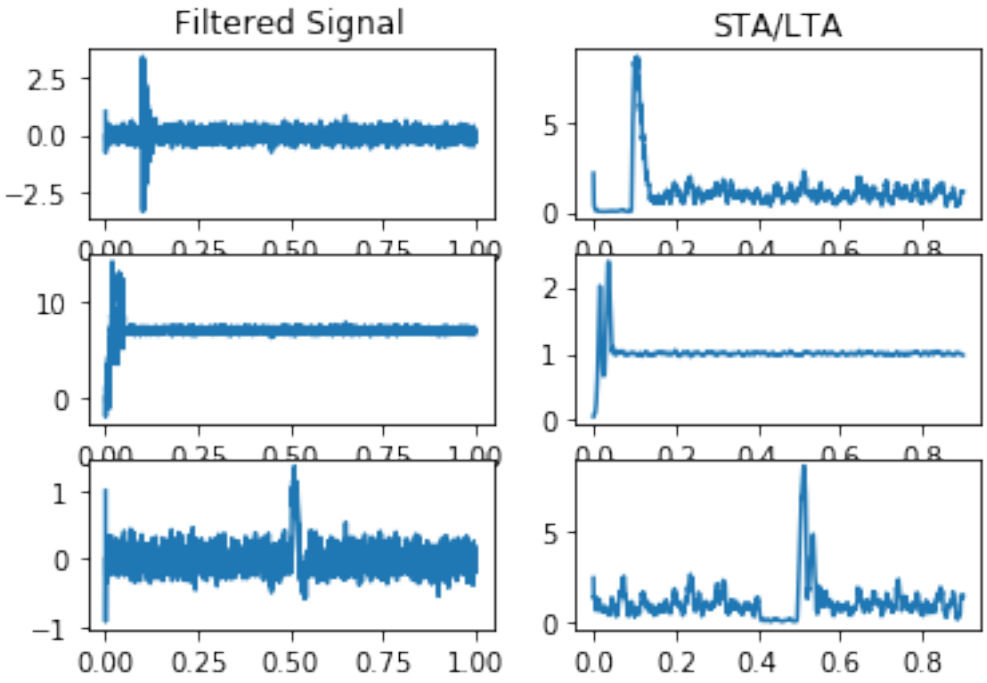
where l_1 and l_2 are the STA window starting and stopping indices; L_1 and L_2 are the STA window starting and stopping indices

Modified STA/LTA

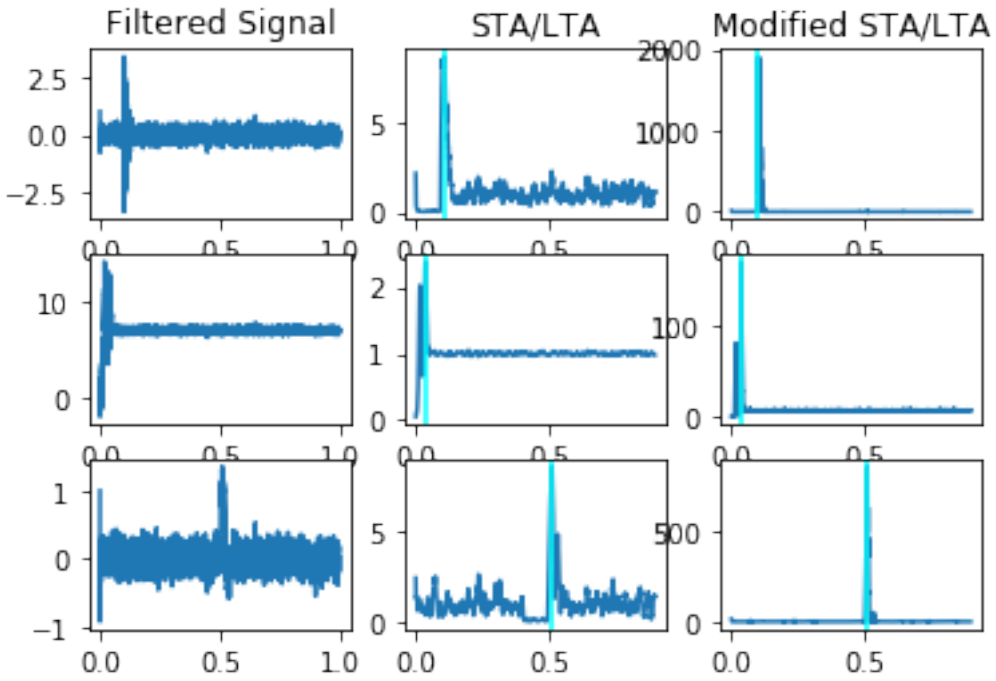
$$mstalta[i] = |signal[i]|^m (stalta[i])^n$$

$m = 1$ and $n = 3$ m,n,l_1,l_2,L_1 and L_2 are the hyperparameters.

```
In [12]: r = []
         plt.figure()
         for i in range(3):
             plt.subplot(3,2,2*i+1)
             plt.plot(t,OP[i,0,:])
             if(i==0):
                 plt.title('Filtered Signal')
             plt.subplot(3,2,2*i+2)
             r1 = stalta(OP[i,0,:])
             plt.plot(t[0:len(r1)],r1)
             if(i==0):
                 plt.title('STA/LTA')
             r.append(r1)
```



In [13]:
```
mr=[]
t1 =[]
t2 =[]
plt.figure()
for i in range(3):
    plt.subplot(3,3,3*i+1)
    plt.plot(t,OP[i,0,:])
    if(i==0):
        plt.title('Filtered Signal')
    plt.subplot(3,3,3*i+2)
    r1 = r[i]
    if(i==0):
        plt.title('STA/LTA')
    plt.plot(t[0:len(r1)],r1)
    tr1 = np.argmax(r1)
    plt.axvline(t[tr1],color='cyan')
    plt.subplot(3,3,3*i+3)
    mr1 = modified_stalta(OP[i,0,:])
    mr.append(mr1)
    tmr1 = np.argmax(mr1)
    plt.plot(t[0:len(mr1)],mr1)
    plt.axvline(t[tmr1],color='cyan')
    if(i==0):
        plt.title('Modified STA/LTA')
    t1.append(tr1)
    t2.append(tmr1)
```



```
In [14]: print('P wave Random Walk and S wave time estimates in seconds using STA/LTA method')
         print(t[t1])
```

```
P wave Random Walk and S wave time estimates in seconds using STA/LTA method
[0.10525 0.03665 0.51205]
```

```
In [15]: print('P wave Random Walk and S wave time estimates in seconds using modified STA/LTA m
         print(t[t2])
```

```
P wave Random Walk and S wave time estimates in seconds using modified STA/LTA method
[0.1004  0.0361  0.51135]
```

1 Comments

The Kalman filter working is verifyed. On application of the STA/LTA method and modified STA/LTA method we can see the detection can be done by setting appropriate thresholds. However, here the P wave and S wave were detected individually(each represented by 3 state variables). This can be done together by using a 5 state variable model (has been done in the next report). So the time estimates are pretty close to the truth (0.1s, 0.025s,0.5s)