Simulations and Results-P wave and S wave together

October 9, 2018

Unlike the previous case, here both S and P wave are considered. The implementation is scaled up accordingly.

```
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 kf5(d1,A,B,Q,R,fs,f0,f1):
            t = np.linspace(0,1-1/fs,fs)
            C = np.array([[1,0,1,0,1]])
            xhat = np.zeros([5,len(d1)])
            xhat_prior = np.zeros([5,len(d1)])
            P_ = []
            Ppost = []
            xhat[:,0] = np.ones(5)
            P0 = np.eye(5)
            Ppost.append(P0)
            omega = 2*np.pi*f0
            omega1 = 2*np.pi*f1
            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(5) - KGain @ C) @ Ptemp
                Ppost.append(P0)
                bb = 1*omega*np.cos(omega*(t[i+1]))/fs
                cc = 1*omega1*np.cos(omega1*(t[i+1]))/fs
                A = np.array([[1,bb,0,0,0],[0,1,0,0,0],[0,0,1,cc,0]
                ,[0,0,0,1,0],[0,0,0,0,0.75]])
            return xhat
```

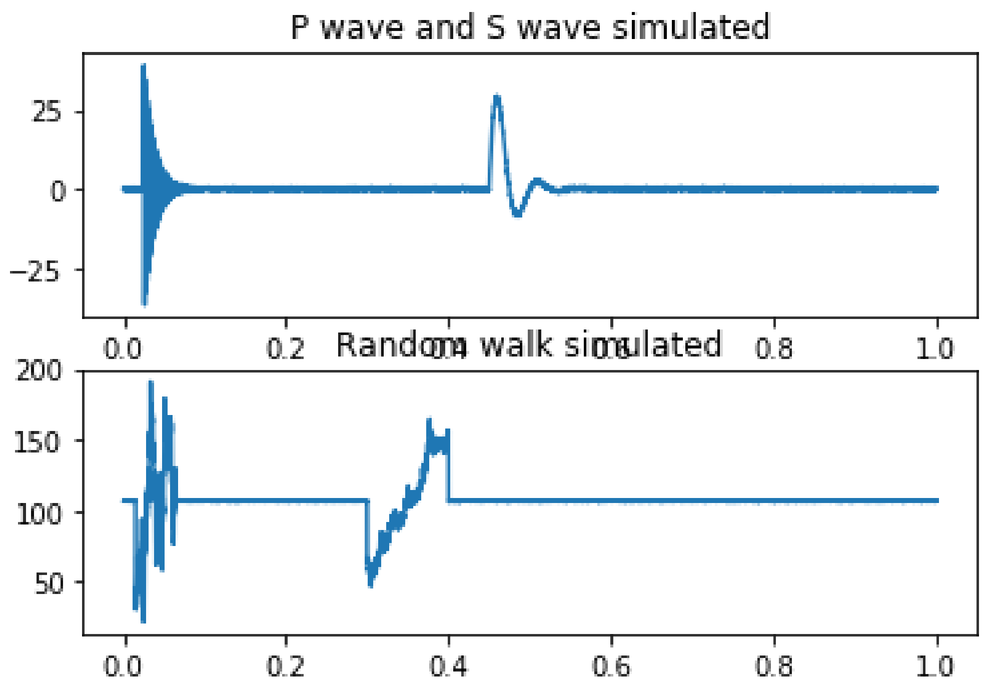
```
def stalta(dataset):
    LTA_window = 5000
    STA_window = 300
    temp1 = np.convolve((dataset)**2,np.ones(STA_window)/STA_window,mode='valid')
    temp2 = np.convolve((dataset)**2,np.ones(LTA_window)/LTA_window,mode='valid')
    temp1 = temp1[0:np.size(temp2)]
    return np.divide(temp1,temp2)

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

Making the data

```
In [3]: 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)

        mea2 = np.random.normal(0,1,1)
        dat = np.random.normal(0,1,1999)
        dat2= np.concatenate((mea2,dat))
        dat2 = np.cumsum(dat2)
        data = np.concatenate((dat1,np.mean(dat1)*np.ones(19000)))

        datb = np.concatenate((dat2,np.mean(dat2)*np.ones(18000)))

        data = 10*np.roll(data,300)
        datb = np.roll(datb,6000)

        datr= data+datb

        fs = 20000
        f0 = 700 #P wave
        f1 = 20 #S wave

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

        t = np.linspace(0,1-0.00005,20000)
        t0 = t[500]
        t1 = t[9000]
        B0 = 50
        A0 =40
        h = 79
        h1 = 50
```

```
A = A0 * np.exp(-h*(t-t0))*np.sin(omega*(t-t0))
A1 = np.concatenate((np.zeros(500),A[500:]))
B = B0 * np.exp(-h1*(t-t1))*np.sin(omega1*(t-t1))
B1 = np.concatenate((np.zeros(9000),B[9000:]))

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)

measurement_noise = np.random.normal(0,0.02,20000)
D = np.zeros([2,20000])
D[0,:] = A1+B1+gm_noise + measurement_noise
D[1,:] = datr+gm_noise + measurement_noise
```

```
In [4]: l = ['P wave and S wave simulated','Random walk simulated']
        plt.figure()
        for i in range(2):
            plt.subplot(2,1,i+1)
            plt.plot(t,D[i,:])
            plt.title(l[i])
```

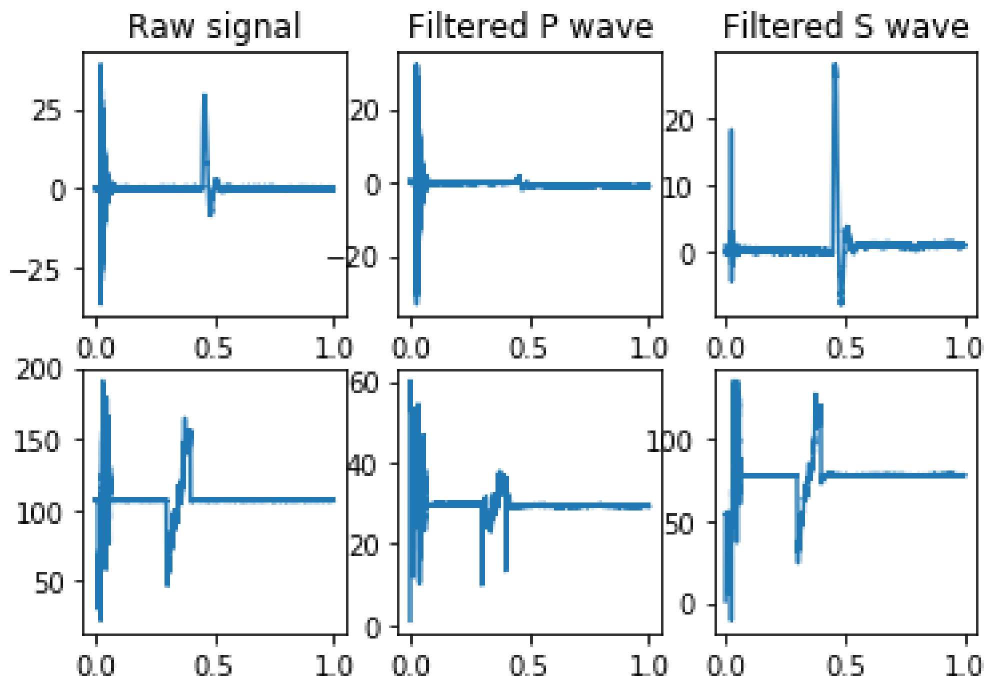


```
In [5]: A_1 = np.array([[1,1*omega/fs,0,0,0],[0,1,0,0,0],[0,0,1,1*omega1/fs,0]
                ,[0,0,0,1,0],[0,0,0,0,0.75]])

        B_1 = np.array([[0,0,0],[0.5,0,0],[0,0,0],[1,0,0],[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),np.random.normal(0,0.04,20000),
                      np.random.normal(0,0.05,20000)])
        Q = np.matmul(Q,np.transpose(Q))/20000
```

Filtered Signals

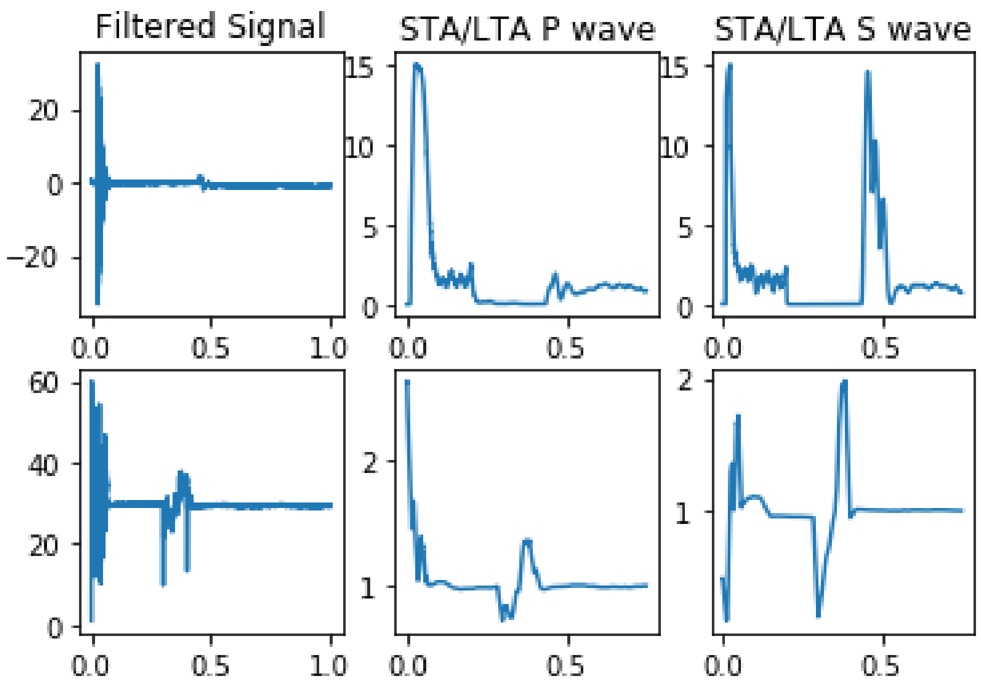
```
In [6]: OP = np.zeros([2,5,20000])
        OP[0,:,:] = kf5(D[0,:],A_1,B_1,Q,np.var(measurement_noise),fs,f0,f1)
        OP[1,:,:] = kf5(D[1,:],A_1,B_1,Q,np.var(measurement_noise),fs,f0,f1)
```

```
In [7]: plt.figure()
        for i in range(2):
            plt.subplot(2,3,3*i+1)
            plt.plot(t,D[i,:])
            if(i==0):
                plt.title('Raw signal')
            plt.subplot(2,3,3*i+2)
            plt.plot(t,OP[i,0,:])
            if(i==0):
                plt.title('Filtered P wave')
            plt.subplot(2,3,3*i+3)
            plt.plot(t,OP[i,2,:])
            if(i==0):
                plt.title('Filtered S wave')
```



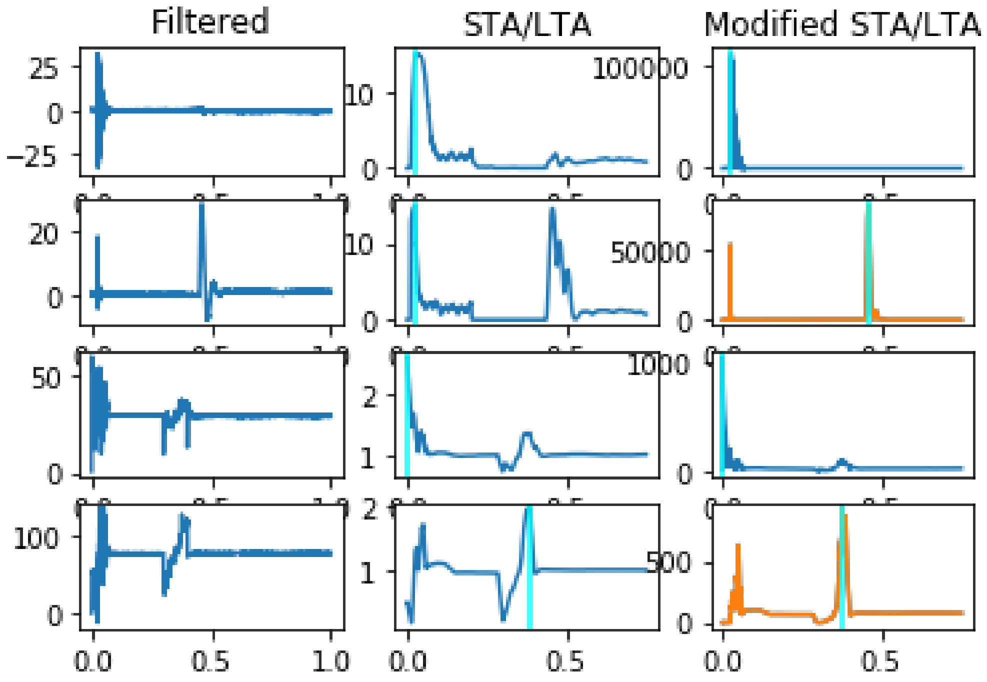
STA/LTA implementation

```
In [8]: r = []
        rr = []
        plt.figure()
        for i in range(2):
            plt.subplot(2,3,3*i+1)
            plt.plot(t,OP[i,0,:])
            if(i==0):
                plt.title('Filtered Signal')
            plt.subplot(2,3,3*i+2)
            r1 = stalta(OP[i,0,:])
            r2 = stalta(OP[i,2,:])
            plt.plot(t[0:len(r1)],r1)
            if(i==0):
                plt.title('STA/LTA P wave')
            plt.subplot(2,3,3*i+3)
            plt.plot(t[0:len(r2)],r2)
            if(i==0):
                plt.title('STA/LTA S wave')
            r.append(r1)
            rr.append(r2)
```



Modified STA/LTA implementation

```
In [9]: mr=[]
        nr=[]
        tp= []
        tn=[]
        plt.figure()
        for i in range(2):
            plt.subplot(4,3,6*i+1)
            plt.plot(t,OP[i,0,:])
            if(i==0):
                plt.title('Filtered')
            plt.subplot(4,3,6*i+2)
            r1 = r[i]
            r2 = rr[i]
            plt.plot(t[0:len(r1)],r1)
            tr1 = np.argmax(r1)
            plt.axvline(t[tr1],color='cyan')
            if(i==0):
                plt.title('STA/LTA')
            plt.subplot(4,3,6*i+3)
            mr1 = modified_stalta(OP[i,0,:])
            nr1 = modified_stalta(OP[i,2,:])
            mr.append(mr1)
            nr.append(nr1)
            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')
            plt.subplot(4,3,6*i+4)
            plt.plot(t,OP[i,2,:])

            plt.subplot(4,3,6*i+5)
            plt.plot(t[0:len(r2)],r2)
            tr2 = np.argmax(r2)
            plt.axvline(t[tr2],color='cyan')
            plt.subplot(4,3,6*i+6)
            plt.plot(t[0:len(nr1)],nr1)
            tnr1 = np.argmax(nr1)
            plt.plot(t[0:len(nr1)],nr1)
            plt.axvline(t[tnr1],color='cyan')
            tn.append(tr1)
            tn.append(tr2)
            tp.append(tmr1)
            tp.append(tnr1)
```



```
In [10]: print('P wave Random Walk and S wave time estimates in seconds using STA/LTA method')
         print(t[tp])
```

```
P wave Random Walk and S wave time estimates in seconds using STA/LTA method
```

```
[2.7550e-02 4.5845e-01 3.5000e-04 3.7700e-01]
```

```
In [11]: print('P wave Random Walk and S wave time estimates in seconds using STA/LTA method')
         print(t[tn])
```

```
P wave Random Walk and S wave time estimates in seconds using STA/LTA method
[2.8650e-02 2.5100e-02 5.0000e-05 3.8505e-01]
```

0.1 Comments

This was just an implementation for detecting both the P wave and S wave simultaneously. Only in the case where in STA/LTA, the maximum amplitude signal is P wave and not S wave. Elsewhere it is performing a good job. The true time of arrivals for the simulated P wave and S wave are 0.025s and 0.45s. For the random walk P wave and S wave the time of arrivals are 0.04s and 0.35s. As mentioned in the previous report, the time of arrival for these is taken as the mean of the width of the wavelet(random walk) plus the delay.