Programmable Embedded Systems
Android Class Work

Submitted by
Pratyush Jaiswal
18EE35014

1-Write an Android App to get the time vs accelerometer data as shown in the matfiles.

2- Write the KF code to clean the acceleration

3- Plot (in matlab or python) t vs ax and ax_estimated ay and ay_estimated then az and az_estimated.

4- Plot for different values of Q and R (at least 7 variants)

In order to use the Kalman filter to estimate the internal state of a process given only a sequence of noisy observations, one must model the process in accordance with the following framework. This means specifying the matrices following:

- Fk , the state-transition model;
- Hk , the observation model;
- Qk , the covariance of the process noise;
- Rk , the covariance of the observation noise;
- and sometimes Bk, the control-input model

Kalman Filter Code

```java
package com.example.app_accelerometer;
import android.content.Context;
import android.widget.Toast;
import java.lang.Math;
//////////////////////Kalman Flter////////////////////////////
public class KalmanFilter1 {


    private static  final MatFunc MATLAB = new MatFunc();
    public static  final double[][] H = new double[][]{  {0, 0, 0, 0, 0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 1, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 1}};

    public static double[][] P = new double[9][9];
    public static double[][] R = new double[3][3];
    public static double[][] Q = new double[9][9];
    public static double[][] X = new double[9][1];
    public static double[][] K = new double[9][3];
    public double prevTime = 0.0;
    public static double h;


    public void init() {
        prevTime = System.currentTimeMillis() / 1e3;

        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 3; j++) {
                K[i][j] = 0;
            }
        }
        for (int i = 0; i < 9; i++) {
            X[i][0] = 0.1;

            for (int j = 0; j < 9; j++) {
                if (i == j) {
//                    P[i][j] = 5;
//                    Q[i][j] = 1;
                    P[i][j] = 1;
                    Q[i][j] = 0.0001;
                } else {
                    P[i][j] = 0;
                    Q[i][j] = 0;
                }
            }
        }
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (i == j) {
//                    R[i][j] = 0.1;
                    R[i][j] = 0.01;
                } else

                    R[i][j] = 0;
            }
        }
    }
```

```
    public double[][] track(double[][] Z, double time) {

        // Implements the Kalman Filter
        // dX = Ph*X + q
        // z = H*X + r
        // Q = Cov(q), R = Cov(r)
        h = time - prevTime;
        prevTime = time;
        double h2 = (h*h)/2;
        double[][] Ph = new double[][]{ {1, 0, 0, h, 0, 0, h2, 0, 0},
                {0, 1, 0, 0, h, 0, 0, h2, 0},
                {0, 0, 1, 0, 0, h, 0, 0, h2},
                {0, 0, 0, 1, 0, 0, h, 0, 0},
                {0, 0, 0, 0, 1, 0, 0, h, 0},
                {0, 0, 0, 0, 0, 1, 0, 0, h},
                {0, 0, 0, 0, 0, 0, 1, 0, 0},
                {0, 0, 0, 0, 0, 0, 0, 1, 0},
                {0, 0, 0, 0, 0, 0, 0, 0, 1}};
        // Step 1 : Compute Kalman Gain
        // K = P*H*(H*P*H' + R)^-1
        double[][] Ht = MATLAB.matTranspose(H, 3, 9);
        double[][] Ktemp = MATLAB.matMul(H, P, 3, 9, 9);
        Ktemp = MATLAB.matMul(Ktemp, Ht, 3, 9, 3);
        Ktemp = MATLAB.matAdd(Ktemp, R, 3, 3, 1);
        double[][] inv = new double[3][3];
         boolean invertible = MATLAB.inverse(Ktemp, inv, 3);
        if (invertible) {
            Ktemp = MATLAB.matMul(P, Ht, 9, 9, 3);
            Ktemp = MATLAB.matMul(Ktemp, inv, 9, 3, 3);
            for (int i=0; i<9; i++) {
                for (int j=0; j<3; j++) {
                    K[i][j] = Ktemp[i][j];
                }
            }
        }
        // Step 2 : Update the estimates
        // zcap = H*x
        // x = x + K*(z - zcap)
        double[][] zcap = MATLAB.matMul(H, X, 3, 9, 1);
        X = MATLAB.matAdd(X, MATLAB.matMul(K, MATLAB.matAdd(Z, zcap, 3, 1, -1), 9, 3, 1), 9, 1, 1);

        // Step 3 : Compute posterior error covariance
        // P = (I - K*H)*P
        P = MATLAB.matMul(MATLAB.matAdd(MATLAB.genID(9), MATLAB.matMul(K, H, 9, 3, 9), 9, 9, -1), P, 9,
9, 9);
        // Save the value
        double[][] zret=new double [3][1];
        zret[0][0]=X[6][0];
        zret[1][0]=X[7][0];
        zret[2][0]=X[8][0];
        // Step 4 : Project ahead
        // x = Ph*x
        // P = Ph*P*Ph' + Q
        X = MATLAB.matMul(Ph, X, 9, 9, 1);
        P = MATLAB.matMul(MATLAB.matMul(Ph, P, 9, 9, 9), MATLAB.matTranspose(Ph, 9, 9), 9, 9, 9);
        P = MATLAB.matAdd(P, Q, 9, 9, 1);
        return zcap;

//        return zret;
    }


}
```

MainActivity Code

```java
package com.example.app_accelerometer;

import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
import android.os.Environment;
import java.io.*;
import java.lang.Math;

///////////////  Main-Activity//////////////////////////

public class MainActivity extends Activity implements SensorEventListener {
    private TextView xText,yText,zText,text;
    private Sensor mySensor;
    private SensorManager SM;
    private Boolean flag=false;
    private static final KalmanFilter1 KF=new KalmanFilter1();
    private static final MatFunc MATLAB =new MatFunc();
    String fileName="sensordata.txt";
    String baseDir = Environment.getExternalStorageDirectory().getAbsolutePath();
    String pathDir = baseDir + "/Android/data/com.mypackage.app_accelerometer/";
    File file;

    File gpxfile;
    FileWriter writer;
    private OutputStreamWriter outputWriter;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
//        initializeViews();
        SM=(SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mySensor=SM.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        SM.registerListener( this,mySensor,SensorManager.SENSOR_DELAY_NORMAL);
        xText=(TextView) findViewById(R.id.xText);
        yText=(TextView) findViewById(R.id.yText);
        zText=(TextView) findViewById(R.id.zText);
        text=(TextView) findViewById(R.id.text);
        file=new File(MainActivity.this.getFilesDir(),"Kalman");
        if (!file.exists()) {
            file.mkdir();
        }
    }
```

```
//    @Override
    public void onStartClick(View view) {
            flag=true;

        try {
            text.setText("starting writing data");
            gpxfile=new File(file,"sensordata.txt");
            writer=new FileWriter(gpxfile);
//            FileOutputStream fileout=openFileOutput(,MODE_PRIVATE);
//            outputWriter=new OutputStreamWriter(fileout);
            KF.init();

        } catch(Exception e){

            text.setText(e.getMessage());
            e.printStackTrace();
        }
    }
//    @Override
    public void onStopClick(View view) {
        flag=false;

        try {

//            outputWriter.close();
            writer.close();
            text.setText("Stopped writting data!");
            Toast.makeText(MainActivity.this,"File saved successfully!",
            Toast.LENGTH_SHORT).show();
        } catch (Exception e) {
            text.setText(e.getMessage());
            e.printStackTrace();
        }
    }
//    protected void onResume() {
//        super.onResume();
//
//    }
    @Override
    public void onSensorChanged(SensorEvent event) {



        double[][] Z=new double[3][1];
        for (int i=0; i<3; i++) {
            Z[i][0] = event.values[i];
        }
        double[][] zcap = KF.track(Z, System.currentTimeMillis()/1e3);

        xText.setText("X: " + Z[0][0]+"->X_new: "+zcap[0][0]);
        yText.setText("Y: " + Z[1][0]+"->Y_new: "+zcap[1][0]);
        zText.setText("Z: " + Z[2][0]+"->Z_new: "+zcap[2][0]);
        if (flag) {
            String str = (Z[0][0] + "," + Z[1][0] + "," + Z[2][0] + "\t\t\t" + zcap[0][0]+','+zcap[1]
[0]+','+zcap[2][0]+'\n');

        try {
//            outputWriter.write(str);
            writer.append(str);
            writer.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    }

    @Override
    public void onAccuracyChanged(Sensor sensor,int accuracy){

    }

}
```

MatrixCalculation Code

```java
package com.example.app_accelerometer;

import java.lang.Math;

public class MatFunc {

    public double[][] matMul(double[][] A, double[][] B, int n, int m, int s) {
        // A - n x m
        // B - m x s
        // C - n x s
        double[][] C = new double[n][s];
        for (int i=0; i<n; i++) {
            for (int j=0; j<s; j++) {
                C[i][j] = 0.0;
            }
        }
        for (int i=0; i<n; i++) {
            for (int j=0; j<s; j++) {
                for (int k=0; k<m; k++) {
                    C[i][j] += A[i][k]*B[k][j];
                }
            }
        }
        return C;
    }

    public double twoNorm(double[][] A, int n, int m) {
        double sum = 0;
        for (int i=0; i<n; i++) {
            for (int j=0; j<m; j++) {
                sum += A[i][j]*A[i][j];
            }
        }
        return Math.sqrt(sum);
    }

    public double twoNormVect(double[] A, int n) {
        double sum = 0;
        for (int i=0; i<n; i++) {
            sum += A[i]*A[i];
        }
        return Math.sqrt(sum);
    }
```

```java
public double oneNorm(double[] A, int n) {
    double sum = 0;
    for (int i=0; i<n; i++) {
        sum += Math.abs(A[i]);
    }
    return sum;
}

public double[][] matTranspose(double[][] A, int n, int m) {
    // A - n x m
    // B - m x n
    double[][] B = new double[m][n];
    for (int i=0; i<n; i++) {
        for (int j=0; j<m; j++) {
            B[j][i] = A[i][j];
        }
    }
    return B;
}

public double[][] matAdd(double[][] A, double[][] B, int n, int m, int sign) {
    double[][] C = new double[n][m];
    for (int i=0; i<n; i++) {
        for (int j=0; j<m; j++) {
            C[i][j] = A[i][j] + B[i][j]*sign;
        }
    }
    return C;
}

static void getCofactor(double[][] A, double[][] temp, int p, int q, int n)
{
    int i = 0, j = 0;
    for (int row = 0; row < n; row++)
    {
        for (int col = 0; col < n; col++)
        {
            if (row != p && col != q)
            {
                temp[i][j++] = A[row][col];
                if (j == n - 1)
                {
                    j = 0;
                    i++;
                }
            }
        }
    }

}
```

```
    static double determinant(double A[][], int n, int N)
    {
        int D = 0;
        if (n == 1)
            return A[0][0];
        double [][]temp = new double[N][N];
        int sign = 1;
        for (int f = 0; f < n; f++)
        {
            getCofactor(A, temp, 0, f, n);
            D += sign * A[0][f] * determinant(temp, n - 1, N);
            sign = -sign;
        }

        return D;
    }

    static void adjoint(double[][] A, double[][] adj, int  N)
    {
        if (N == 1)
        {
            adj[0][0] = 1;
            return ;
        }
        int sign = 1;
        double [][]temp = new double[N][N];

        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
            {
                getCofactor(A, temp, i, j, N);
                sign = ((i + j) % 2 == 0)? 1: -1;
                adj[j][i] = (sign)*(determinant(temp, N-1, N));
            }
        }

    }

   static boolean inverse(double A[][], double [][]inverse, int N)
    {
        double det = determinant(A, N, N);
        double [][]C=inverse;
        if (det == 0)
        {
            System.out.print("Singular matrix, can't find its inverse");
//            Toast.makeText(MatFunc.this, "Singular matrix, can't find its inverse",
Toast.LENGTH_SHORT ).show();
            return false;
        }
        double [][]adj = new double[N][N];
        adjoint(A, adj, N);
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                inverse[i][j] = adj[i][j]/(float)det;
//        System.out.print(inverse);
        return true;
    }

    public double[][] genID(int n) {
        double[][] ID = new double[n][n];
        for (int i=0; i<n; i++) {
            for (int j=0; j<n; j++) {
                if (i==j)
                    ID[i][j] = 1;
                else
                    ID[i][j] = 0;
            }
        }
        return ID;
    }

}
```

Matlab Code to generate plots

```
% Matlab Code to generate plots:


load testmat_pratyush;


% Initial Guess

% state


x=randn(9,1);


% Covariance


P=eye(9);




% Process Noise covariance Q


Q=0.0007*eye(9);




% Measurement Noise covariance R
```

```
R=0.07*eye(3);




X=[]; Z=[]; Gain=[]; Err=[];

N=length(t);

% Construct H matrix

        H=[zeros(3,6) eye(3)];




for n=1:N-1

        h=t(n+1)-t(n);

        h2=h^2/2;

        % Construct Phi matrix

        phi=[eye(3)     h*eye(3) h2*eye(3)
```

```
zeros(3)          eye(3)  h*eye(3)

zeros(3)          zeros(3)    eye(3)];



%  Compute the Kalman Gain K

K=P*H'*inv(H*P*H'+R);



% Update the states

z=a(n,:)';

err=(z-H*x);

x=x+K*err;



% Update the P
```

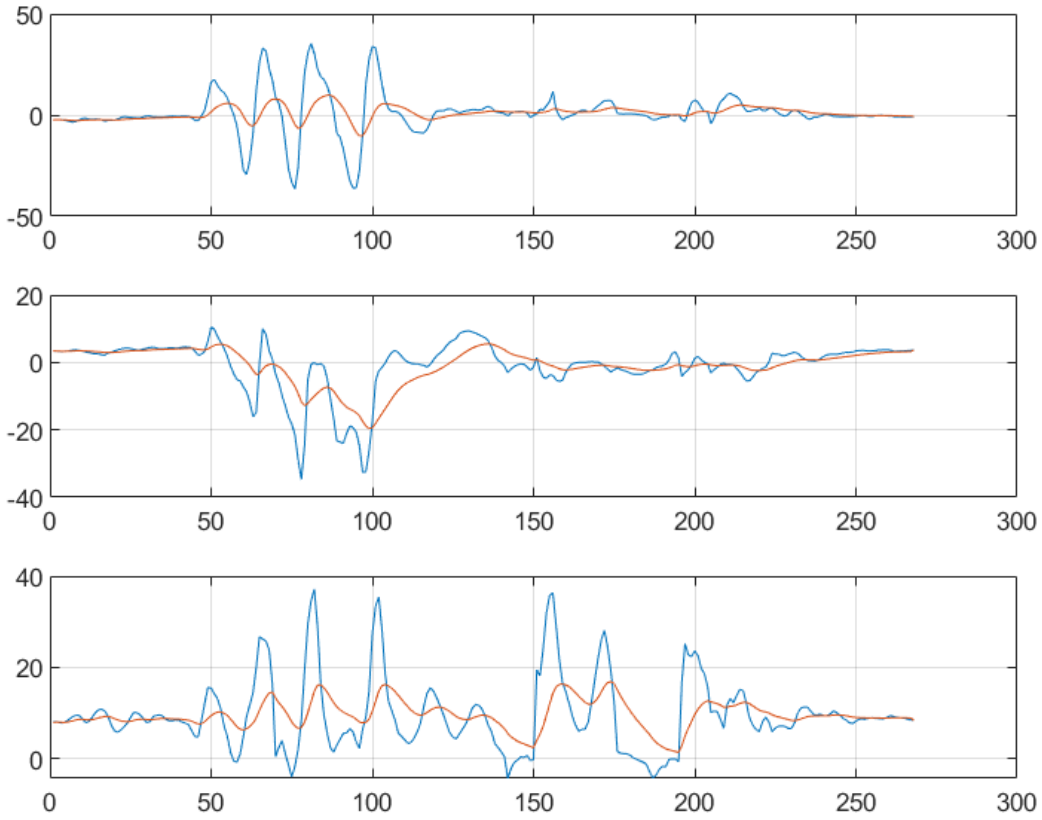
```
        P=(eye(9)-K*H)*P;




        %Save the values

        X=[X;x(:)'];

        Z=[Z;z(:)'];

        Gain=[Gain;K(:)'];

        Err=[Err;err(:)'];



        % Project Ahead

        x=phi*x;

        P=phi*P*phi'+Q;

end
```

```
ae=[X(:,7:9);a(end,:)];




subplot(311)

plot([a(:,1) ae(:,1)])

grid on




subplot(312)

plot([a(:,2) ae(:,2)])

grid on
```

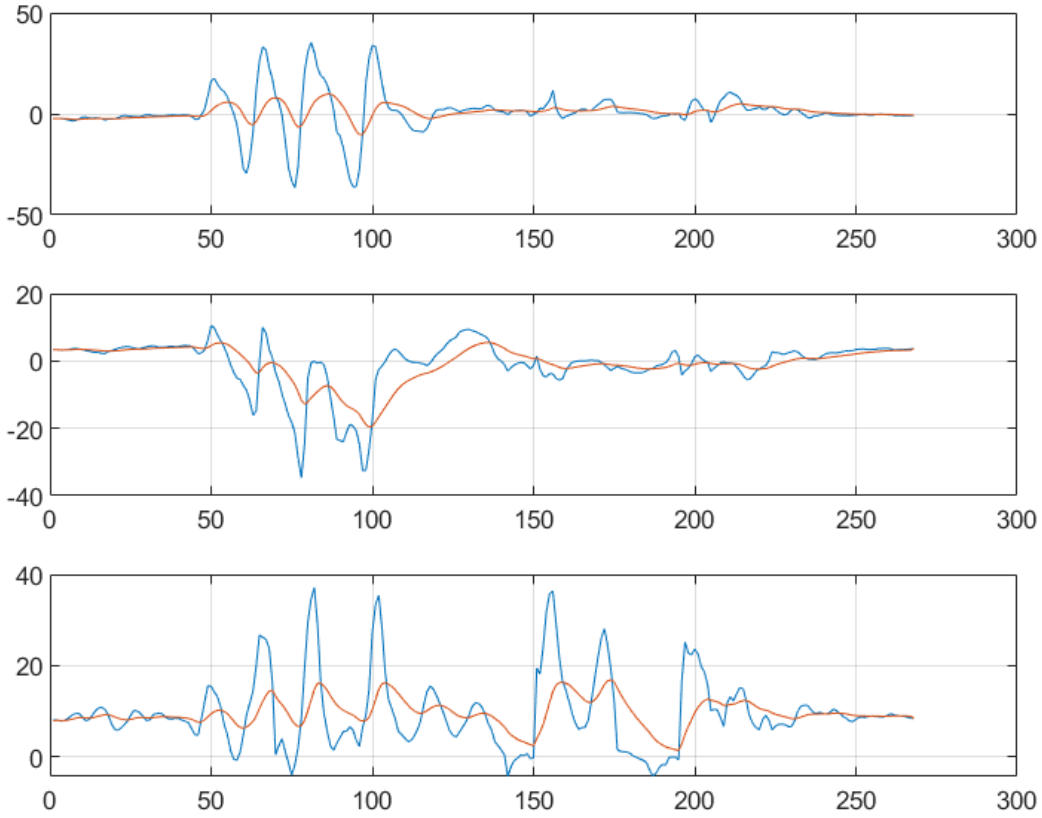
```
subplot(313)


plot([a(:,3) ae(:,3)])


grid on




shg
```

Plots

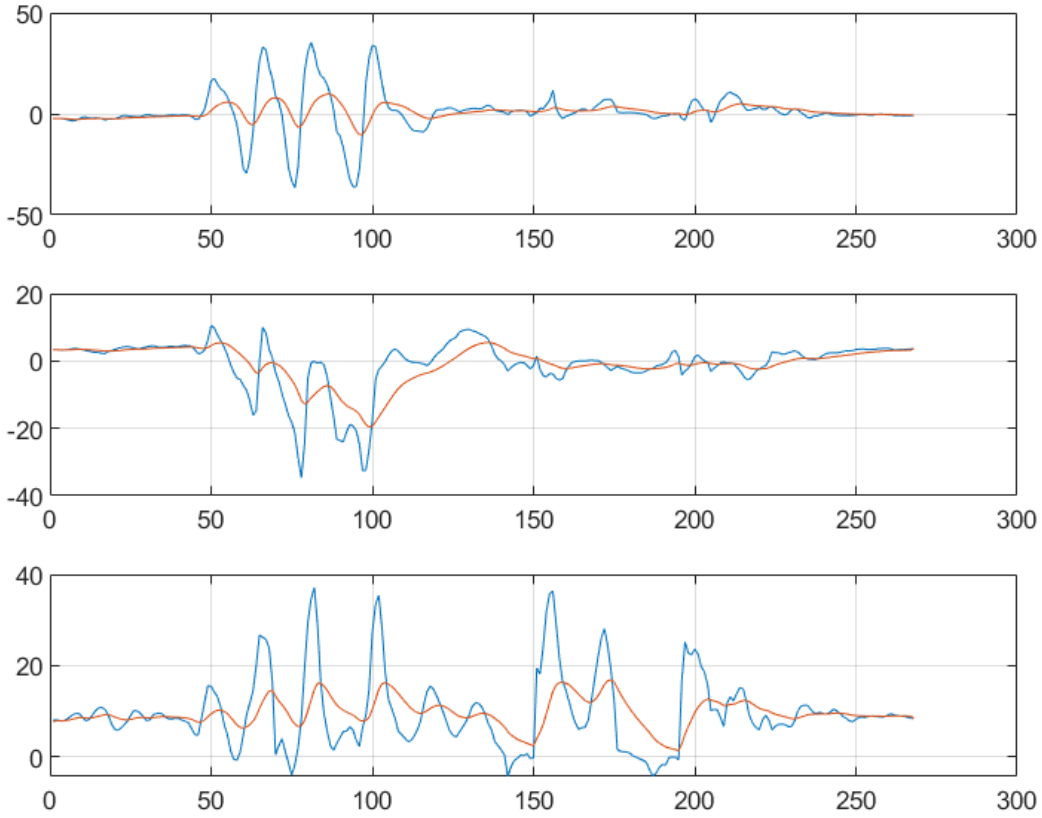
Q=0.001, R=0.01



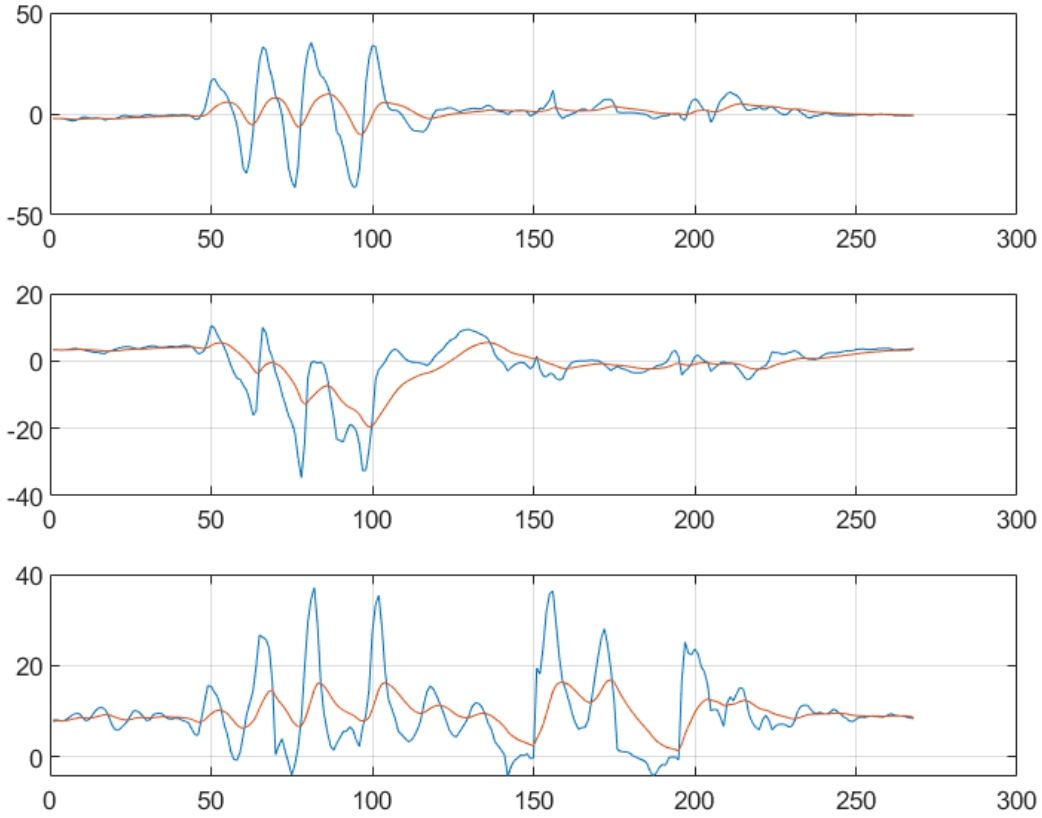
Q=0.002, R=0.02


50
0
-50
0
50
100
150
200
250
300
20
0
-20
-40
0
50
100
150
200
250
300
40
20
0
0
50
100
150
200
250
300

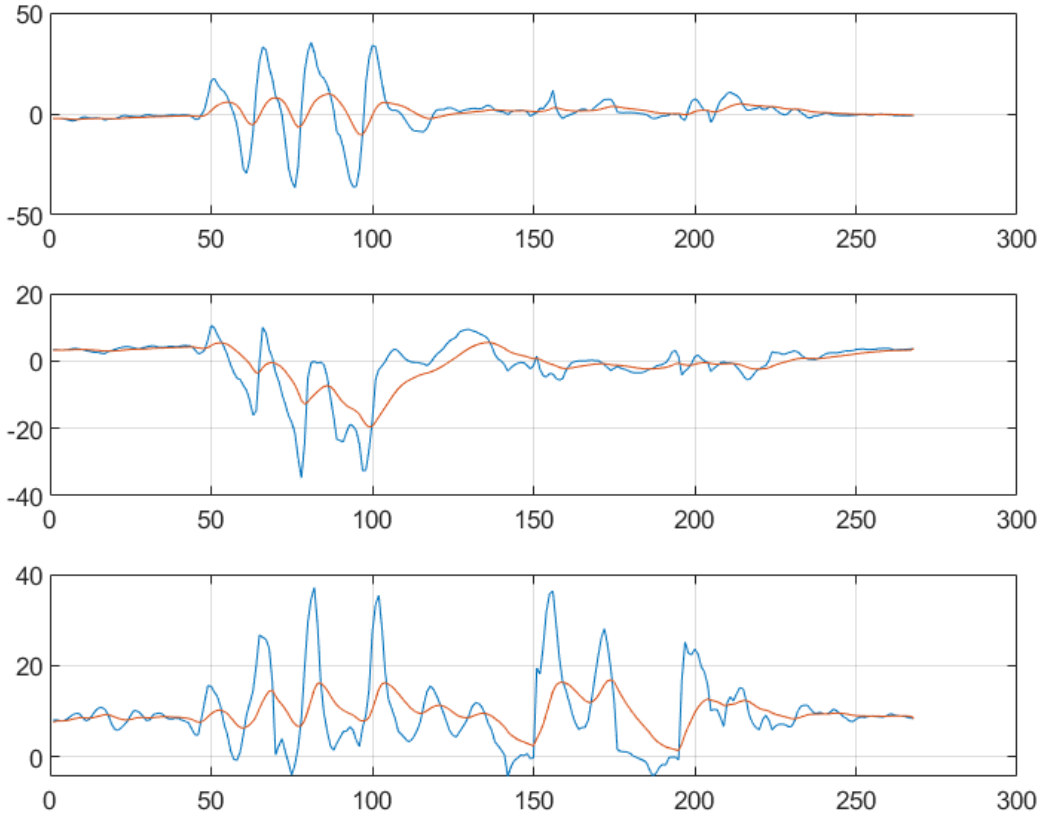
Q=0.003, R=0.03


50
0
-50
0
50
100
150
200
250
300
20
0
-20
-40
0
50
100
150
200
250
300
40
20
0
0
50
100
150
200
250
300

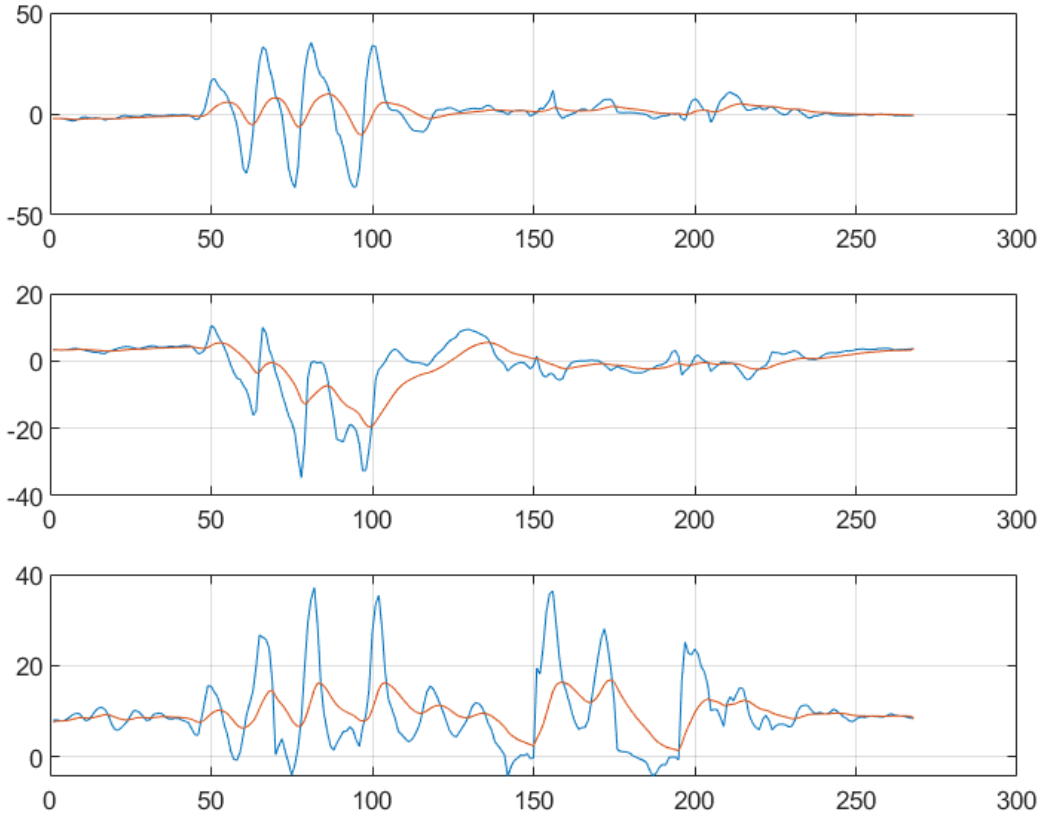
Q=0.004, R=0.04


50
0
-50
0
50
100
150
200
250
300
20
0
-20
-40
0
50
100
150
200
250
300
40
20
0
0
50
100
150
200
250
300

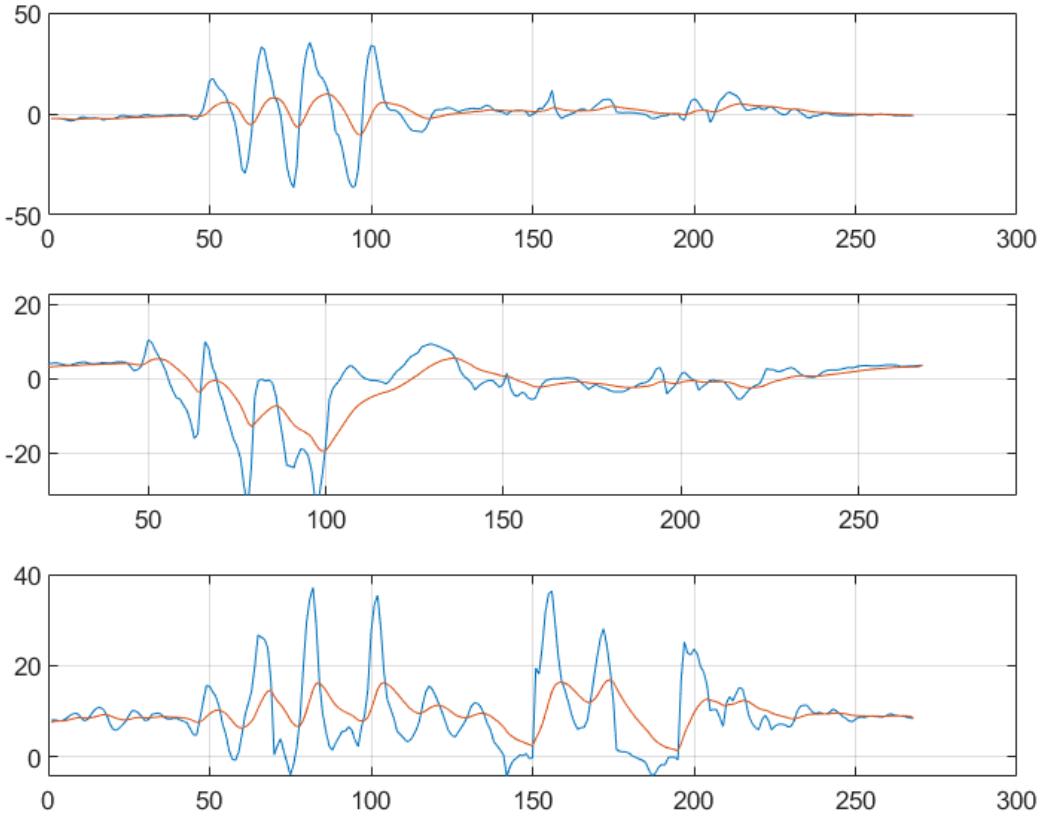
Q=0.005, R=0.05


50
0
-50
0
50
100
150
200
250
300
20
0
-20
-40
0
50
100
150
200
250
300
40
20
0
0
50
100
150
200
250
300

Q=0.006, R=0.06


50
0
-50
0
50
100
150
200
250
300
20
0
-20
-40
0
50
100
150
200
250
300
40
20
0
0
50
100
150
200
250
300

Q=0.007, R=0.07


50
0
-50
0
50
100
150
200
250
300
20
0
-20
50
100
150
200
250
40
20
0
0
50
100
150
200
250
300