Core Temperature From Heart Rate Algorithm Matrix Math

December 21, 2016

1 Core Temperature Algorithm

The following simulation is based on the 2010 paper by Buller et al. The algorithm takes a time sequence of measured heart rate and variance of heart rate and outputs an estimate of core temperature. Following Buller's method we employ a Kalman filter to combine the core temperature estimate based on heart rate and the core temperature estimate based on a model of core temperature time evolution. The Kalman filter combines these estimates based on the variance of the noise associated with each process.

1.1 Kalman Filter Overview

The Kalman Filter is a specific type of g-h filter. The Kalman filter employs a linear state space model and Gaussian noise models. In fact it is optimal in the mean square sense for these conditions. The gains (g-h) are dynamically updated based on the time evolution of the noise parameters.

The generic g-h algorithm can be visualized as follows:

Initialization

1. Initialize the state of the filter
2. Initialize our belief in the state

Predict

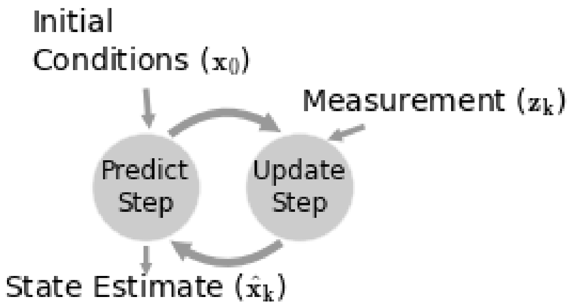
1. Use system behavior to predict state at the next time step
2. Adjust belief to account for the uncertainty in prediction

Update

1. Get a measurement and associated belief about its accuracy
2. Compute residual between estimated state and measurement
3. Compute scaling factor based on whether the measurement or prediction is more accurate
4. set state between the prediction and measurement based on scaling factor
5. update belief in the state based on how certain we are in the measurement

1.1.1 Equations for Kalman Filter

The Kalman filter takes the paradigm described above and formalizes it for linear state space models with Gaussian Noise.


Initial
Conditions ($\mathbf{x}_0$)
Measurement ($\mathbf{z}_\mathbf{k}$)
Predict Step
Update Step
State Estimate ($\hat{\mathbf{x}}_\mathbf{k}$)

Generic g-h filter

	Univariate	Univariate (Kalman form)	Multivariate
Predict	$\bar{\mu} = \mu + \mu_{f_x}$ $\bar{\sigma}^2 = \sigma_x^2 + \sigma_{f_x}^2$	$\bar{x} = x + dx$ $\bar{P} = P + Q$	$\bar{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u}$ $\bar{\mathbf{P}} = \mathbf{F}\mathbf{P}\mathbf{F}^\mathsf{T} + \mathbf{Q}$

Without worrying about the specifics of the linear algebra, intuition tell us:

$\mathbf{x}$, $\mathbf{P}$ are the state mean and covariance. They correspond to x and σ^2.

$\mathbf{F}$ is the *state transition function*. This defines how the state evolves over time. When multiplied by $\mathbf{x}$ it computes the prior. In the case of the current implementation $\mathbf{F} = I$, meaning the state does not change over time. Specifically for core temperature this assumption implies that core temperature remains constant over the measurement interval.

$\mathbf{Q}$ is the process covariance. It corresponds to $\sigma_{f_x}^2$. This is the variance of the core temperature.

$\mathbf{B}$ and $\mathbf{u}$ model control inputs to the system. In the current implementation they are ignored.

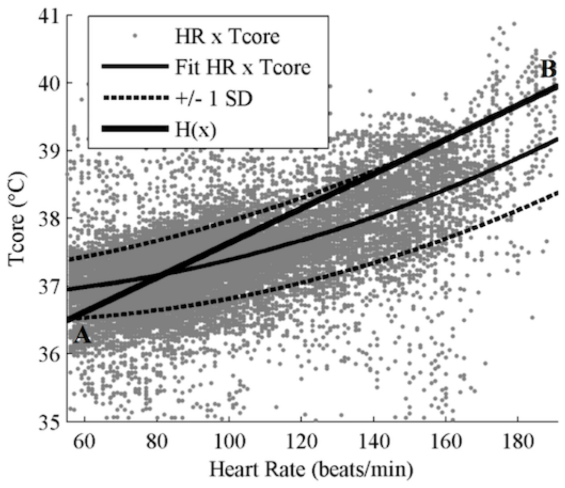
	Univariate	Univariate (Kalman form)	Multivariate
Update	 $\mu = \frac{\bar{\sigma}^2\mu_z + \sigma_z^2\bar{\mu}}{\bar{\sigma}^2 + \sigma_z^2}$ $\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$	$y = z - \bar{x}$ $K = \frac{\bar{P}}{\bar{P} + R}$ $x = \bar{x} + Ky$ $P = (1 - K)\bar{P}$	$\mathbf{y} = \mathbf{z} - \mathbf{H}\bar{\mathbf{x}}$ $\mathbf{K} = \bar{\mathbf{P}}\mathbf{H}^\mathsf{T}(\mathbf{H}\bar{\mathbf{P}}\mathbf{H}^\mathsf{T} + \mathbf{R})^{-1}$ $\mathbf{x} = \bar{\mathbf{x}} + \mathbf{K}\mathbf{y}$ $\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\bar{\mathbf{P}}$

$\mathbf{H}$ is the measurement function. This is found empirically in the current implementation. H is meant to be linear however in Buller's paper it is not. Thus we implement an extended Kalman Filter.

$\mathbf{z}$, $\mathbf{R}$ are the measurement mean and noise covariance. They correspond to z and σ_z^2 in the univariate filter. Our measurement is heart rate and thus R corresponds to the variance of the heart rate.

$\mathbf{y}$ and $\mathbf{K}$ are the residual and Kalman gain. The Kalman gain weights the update of the state with a combination of the measurements and state evolution.

The details will be different than the univariate filter because these are vectors and matrices, but the concepts are exactly the same:



Core Temperature as a function of Heart Rate

- Use a Gaussian to represent our estimate of the state and error
- Use a Gaussian to represent the measurement and its error
- Use a Gaussian to represent the process model
- Use the process model to predict the next state (the prior)
- Form an estimate part way between the measurement and the prior

1.2 Comments on Buller's Paper

Buller's paper is akin to a classic western movie - there is good, bad and ugly.

1.2.1 GOOD

The paper summarizes the relationship between core temperature and heart rate from three field studies comprising of 38 test subjects and 20,000 data points. The results are summarized in the following graph and really is the heart of the paper.

Note in this chart the linear fit is not great. Buller notes that many of the core temperature heart rate data points where not taken at steady state thus the large variance. His workaround is to fit a line between the lowest point in the fit minus one standard deviation to the highest point in the fit plus one standard deviation curves. This results in H(x) which maps heart rate to core temperature. He defines the linear transform H as the map between core temperature and heart rate (confusing) as

$$H(x) = 39.701x - 1381.6890$$

Using the same data sets he find the variance of core temperture (over all heart rates) as $Q_o = 0.000576$

1.2.2 BAD

Buller severely confuses Kalman state representation with his implementation. The crux of the matter is that while $H(x)$ is a function of a line it is **not** a linear transform. Specifically define transform T such that:

$$T[x] = mx + b$$

then the following does not hold

$$T[G_1 x_1] + T[G_2 x_2] \neq G_1 T[x_1] + G_2 T[x_2]$$

However Kalman Filters as described in Bullers paper only hold on linear transforms. Thus we need to linearize $H(x)$ at each x or core temperature before applying the Kalman Filter. This is really an extended Kalman Filter.

1.2.3 UGLY

The paper has a mistake in equeation 3. The residual $\tilde{y}_t$ **needs** to be defined as the the difference between the measured heart rate and estimated heart rate not the other way around. Specifically:

$\mathbf{y} = \mathbf{z} - \mathbf{H}\bar{\mathbf{x}}$

NOT

$\mathbf{y} = \mathbf{H}\bar{\mathbf{x}} - \mathbf{z}$

The way it is described in equation 3 cause the filter to be unstable and rail $+/-\infty$.

1.3 Current Implementaion Notes

We modify (actually fix and make work) Buller′s model. This is accomplished by extending the state space to accommodate a linear function as the measruing funciton. In other words we linearize $H(X)$ at the expense of increasing the state space by one dimension.

$$\bar{x}_t = \begin{bmatrix} \bar{x}_t \\ 1 \end{bmatrix}$$

$$\bar{P}_t = \begin{bmatrix} \bar{P}_t & 0 \\ 0 & 1 \end{bmatrix}$$

$$Q_t = \begin{bmatrix} Q_o & 0 \\ 0 & 0 \end{bmatrix}$$

$$F = I$$

$$H = \begin{bmatrix} m_o & b_o \end{bmatrix}, where \quad m_o = 39.701 \quad and \quad b_o = -1381.6890$$

We also hold the Kalman gain for our constant bias term at unity prior to state space updates.

$$K_t = \begin{bmatrix} K_{1t} \\ 1 \end{bmatrix}$$

```
In [121]: %run '/Users/tank/Valencell/notebooks/core/core_routines.ipynb'
          %matplotlib inline
```

```
Using Numpy    version 1.11.1
Using Pandas   version 0.18.1
Using python   version 3.5.2
Using Anaconda version 4.2.0 (x86_64)
Using GCC      version 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)]
```

```
In [122]: def initKalmanFilter(To=37, R=500):
              # define initialize state variables
              #To = 35         # initial core temp guess
              Qo = 0.00056    # empircal based on paper
              Po = Qo         # initial estimate
              Hslope = 39.3701
              Hintercept = -1381.6890

              # define state
              Tpred = np.array([[To],[1]])
              H = np.array([[Hslope, Hintercept]])
              Ppred = np.array([[Po, 0],[0,1]])
              Q = np.array([[Qo, 0],[0,0]])
              #R = 500 # variance of heart rate measurement, update adaptively

              Pold = Ppred
              Told = Tpred

              return [Told, Pold, H, Q, R]
```

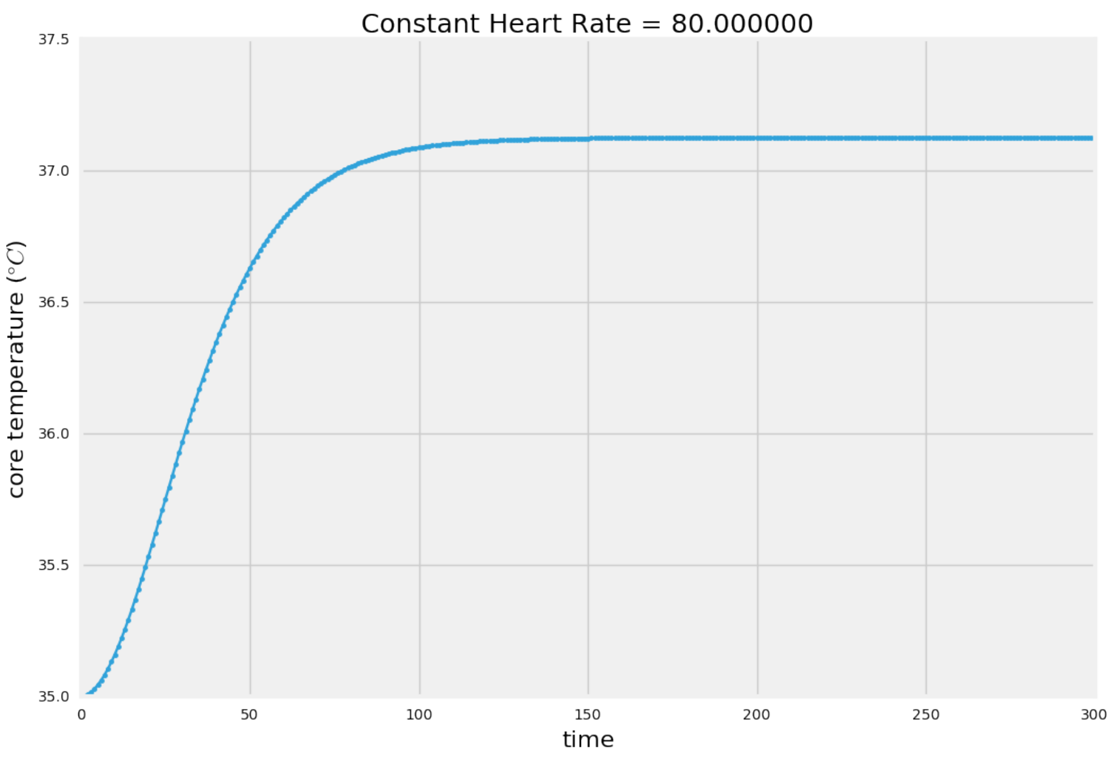
```
In [123]: def kalmanFilter(HR, Told, Pold, Q, H, R):
              #print(Pold[0][0])
              Tpred = Told
              Ppred = Pold + Q

              K = Ppred.dot(H.T) / (H.dot(Ppred).dot(H.T) + R)
              P = (np.eye(2) - K.dot(H)).dot(Ppred)
              K[1] = 0 #force a linear model (rough EKF)
              T = Tpred + K.dot(HR - H.dot(Tpred))
              #T[1] = 1 #force a linear model (rough EKF)
              return [T,P,K]
```

1.3.1 Constant Heart Rate

Note that increasing he Heart Rate variance R will decrease the track rate of core temperature as expected.

```
In [124]: # constant heart rate
          [Told, Pold, H, Q, R] = initKalmanFilter(35, R=300)
          HR = 80
          count = 0
          maxCount = 60*5 # 5 min with nominal 1 Hz Sampling
```

```
KK = []
TT = []
PP = []
while count<maxCount:
    [t,p,k] = kalmanFilter(HR, Told, Pold, Q, H, R)
    KK.append(k)
    TT.append(t)
    PP.append(p)
    Told = t
    Pold = p
    count = count + 1

KK = np.squeeze(np.array(KK))
TT = np.squeeze(np.array(TT))
PP = np.squeeze(np.array(PP))

plot(TT[:,0], xlabel = 'time', ylabel = 'core temperature ' + '('+r'$\deg
                 title='Constant Heart Rate = %f'%(HR));
```



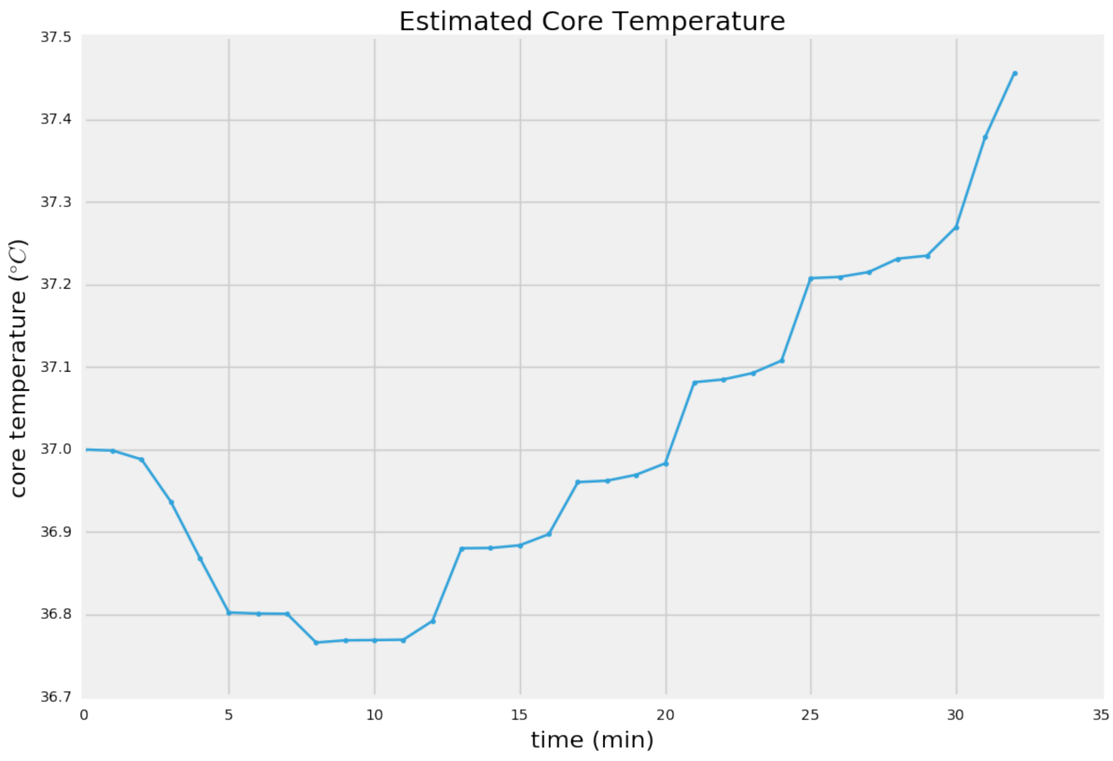
1.3.2 Variable Heart Rate

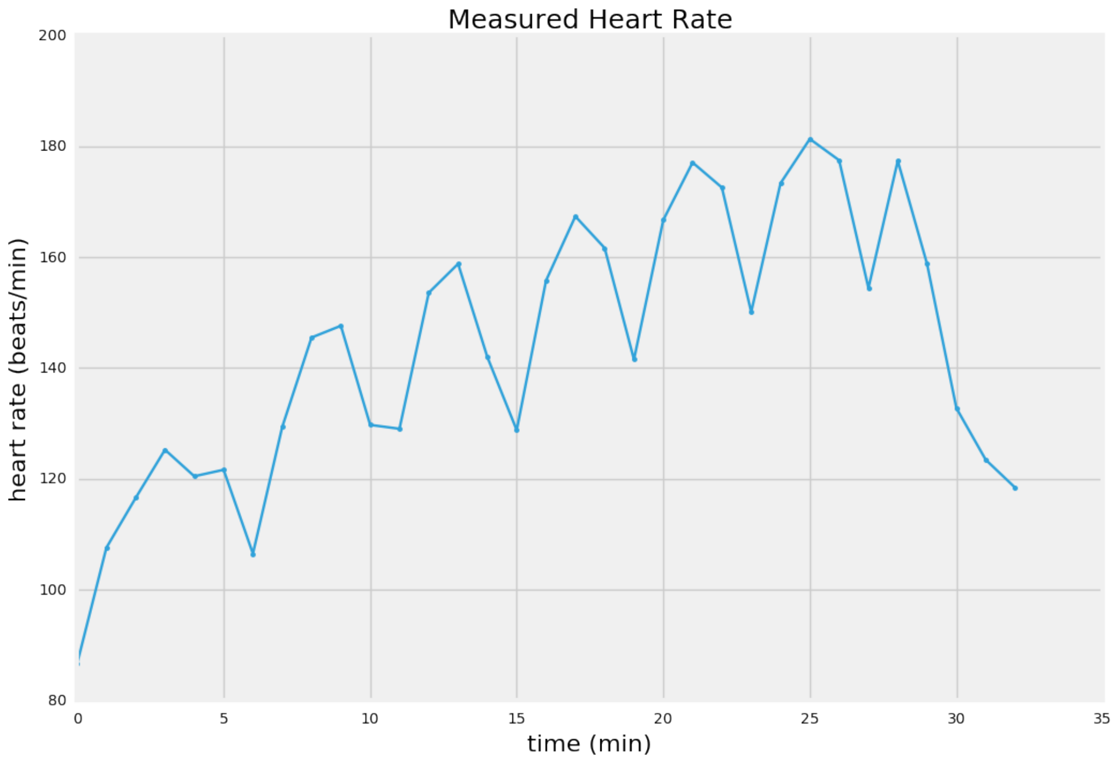
Heart rate is taken from a sample test file. The variance is found by taking a rolling average over a window.

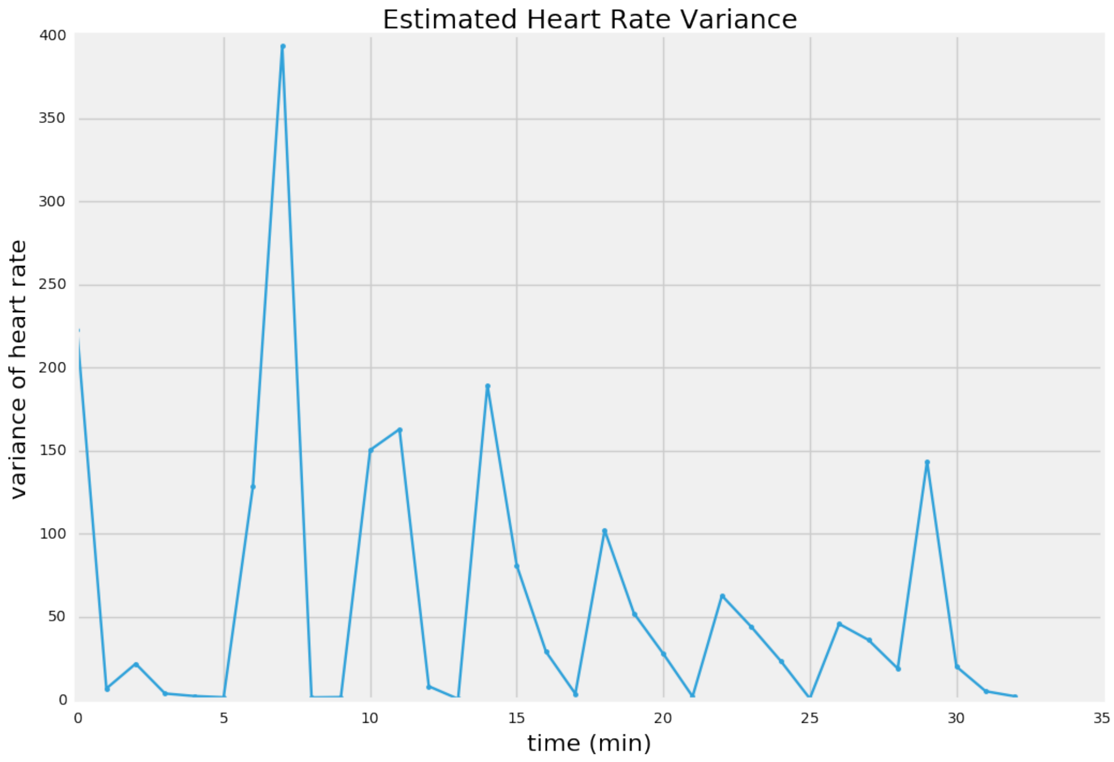
```
In [125]: # heart rate from file
          filename = '~/JoWa.xlsx'
          winLen = 60
          import pandas as pd
          if filename == '~/hr.csv':
              dfHeartRate = pd.read_csv(filename)
              dfHeartRate.drop(dfHeartRate.columns[[0,2,3,4]],axis=1,inplace=True)
          elif (filename == '~/meje.csv') or (filename == '~/AlKe.csv'):
              dfHeartRate = pd.read_csv(filename)
              dfHeartRate.drop(dfHeartRate.columns[[0,1,2,4]],axis=1,inplace=True)
          elif (filename == '~/JoWa.xlsx'):
              dfHeartRate = pd.read_excel(filename)
              dfHeartRate.drop(dfHeartRate.columns[[0,1]],axis=1,inplace=True)

          HR = dfHeartRate.values
          R = dfHeartRate.rolling(window=winLen).var().values

          # prefill due to lag of filter
          R[0:winLen-1] = R[winLen]

In [126]: numBlock = int(len(HR) / winLen)
          HR60 = HR[0:winLen*numBlock].reshape(numBlock,winLen).mean(axis=1)
          R60  = HR[0:winLen*numBlock].reshape(numBlock,winLen).var(axis=1)
```

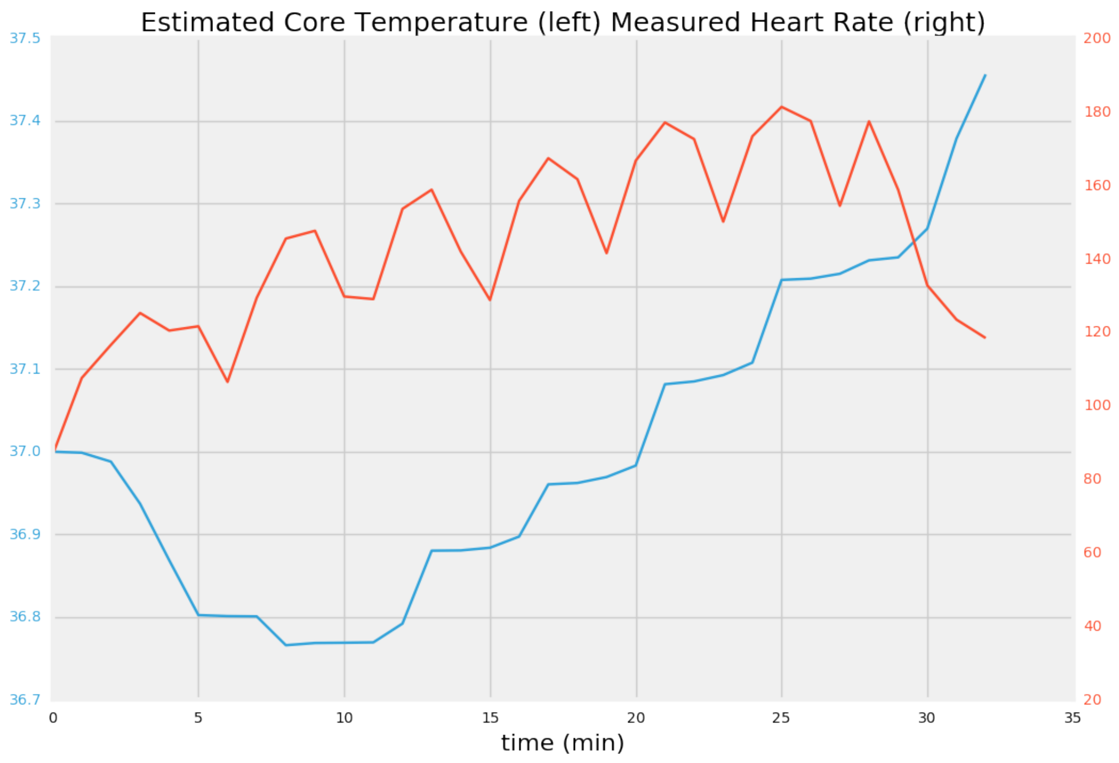
1.3.3 Update every minute with a nomative fs = 1 Hz

```
In [127]: # file heart rate
          [Told, Pold, H, Q, _] = initKalmanFilter(37, R=1000)
          count = 0
          maxCount = 60*60
          KK = []
          TT = []
          PP = []
          while count < HR60.size:
              [t,p,k] = kalmanFilter(HR[count], Told, Pold, Q, H, R60[count])#+150

              KK.append(k)
              TT.append(t)
              PP.append(p)
              Told = t
              Pold = p
              count = count + 1

          KK = np.squeeze(np.array(KK))
          TT = np.squeeze(np.array(TT))
          PP = np.squeeze(np.array(PP))

          plot(TT[:,0], xlabel = 'time (min)',
                        ylabel = 'core temperature ' + '('+r'$\degree C$)',
                        title  = 'Estimated Core Temperature')

          plot(HR60, xlabel = 'time (min)',
                     ylabel = 'heart rate (beats/min)',
                     title  = 'Measured Heart Rate')

          plot(R60, xlabel = 'time (min)',
                    ylabel = 'variance of heart rate',
                    title  = 'Estimated Heart Rate Variance')

          plotyy(TT[:,0], HR60, xlabel='time (min)',
                 title='Estimated Core Temperature (left) Measured Heart Rate (righ
```

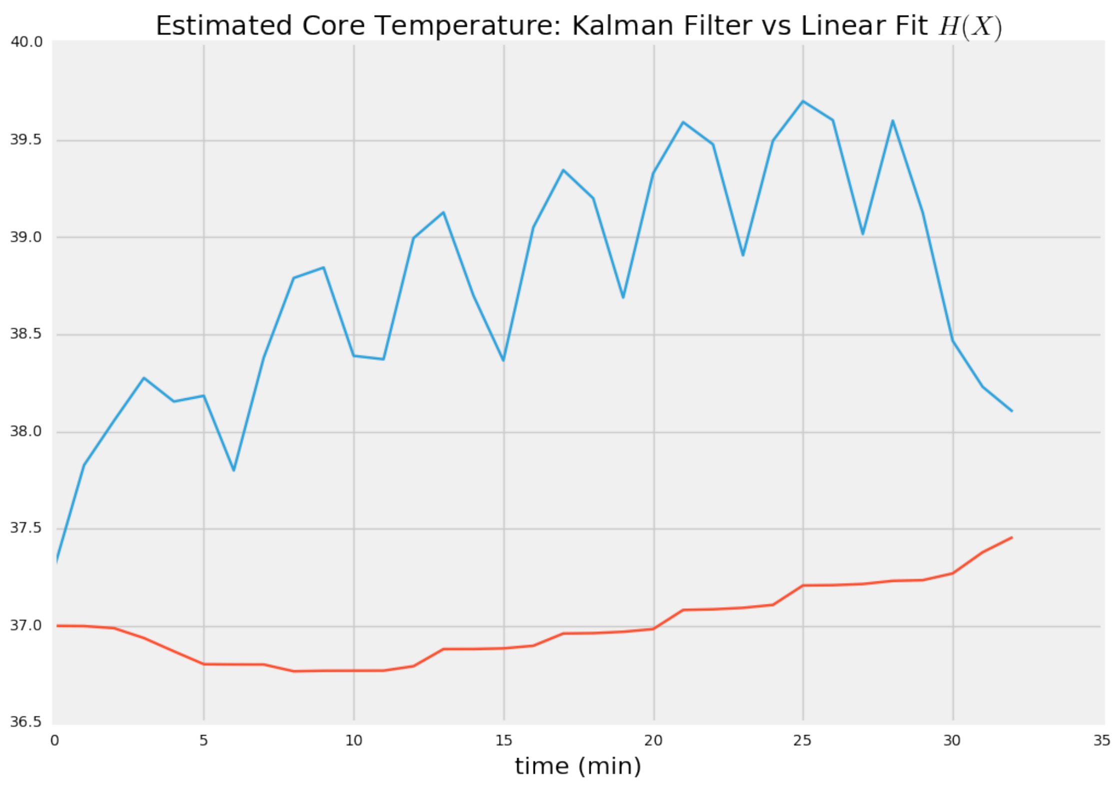



Measured Heart Rate
heart rate (beats/min)
time (min)
200
180
160
140
120
100
80
0
5
10
15
20
25
30
35


Estimated Heart Rate Variance
variance of heart rate
time (min)
400
350
300
250
200
150
100
50
0
0
5
10
15
20
25
30
35



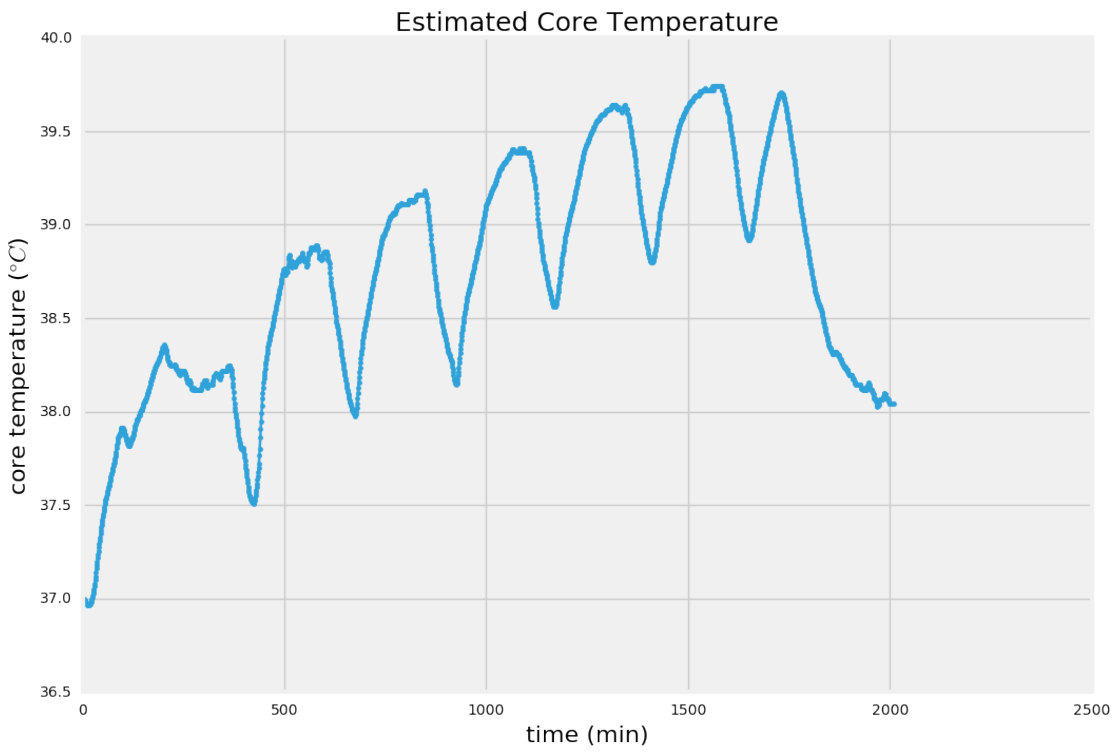
```
In [128]: Hslope = 39.3701
          Hintercept = -1381.6890

          x = HR60
          y = (x - Hintercept)/Hslope
          p = setPlot(xlabel='time (min)',
                      title='Estimated Core Temperature: Kalman Filter vs Linear Fit $
          p.plot(y);
          p.plot(TT[:,0]);
```

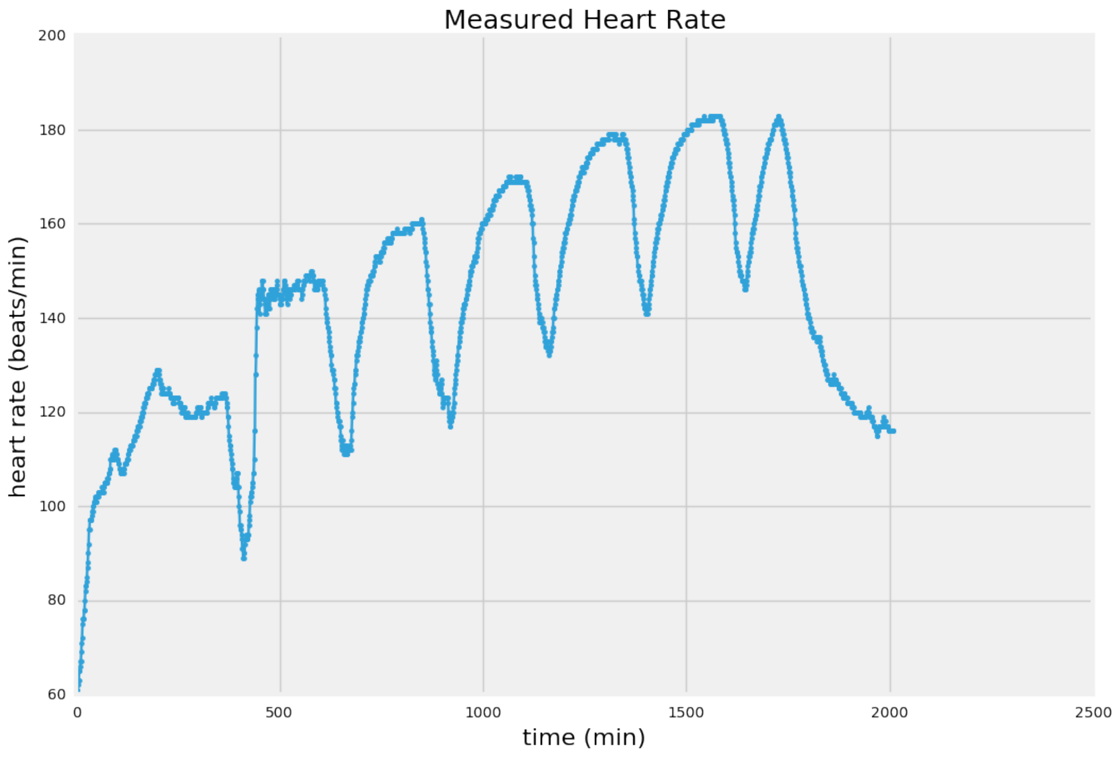


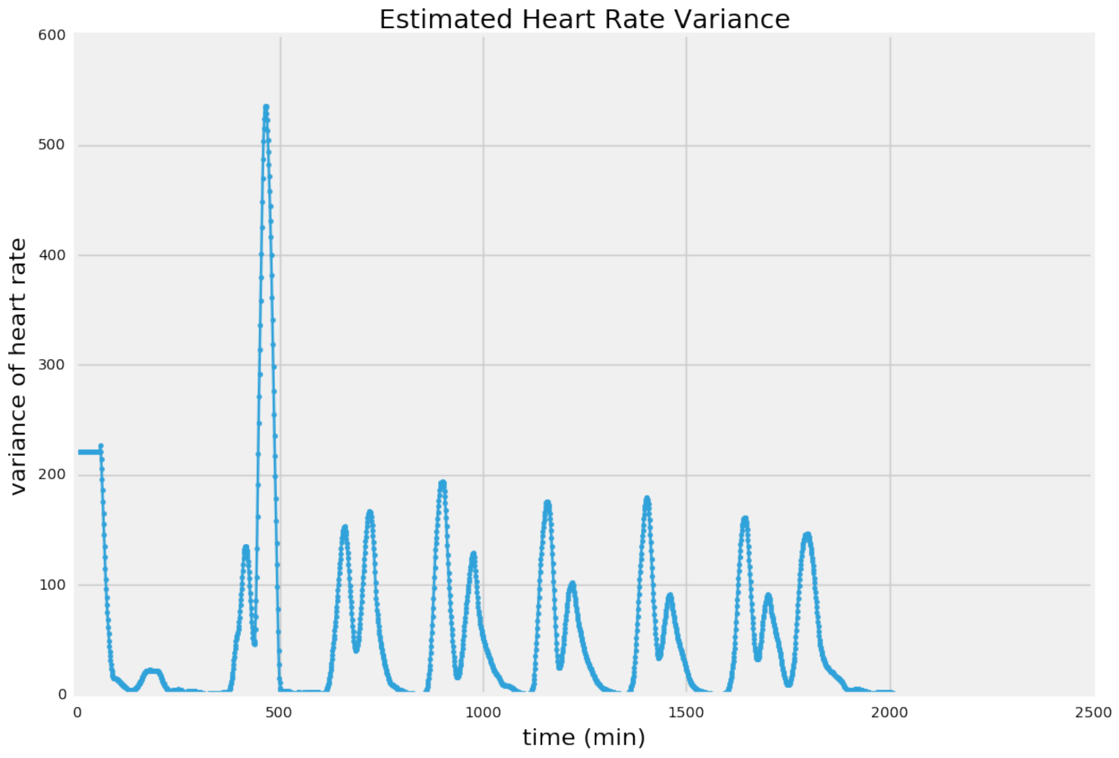
1.3.4 Update at fs (1 Hz)

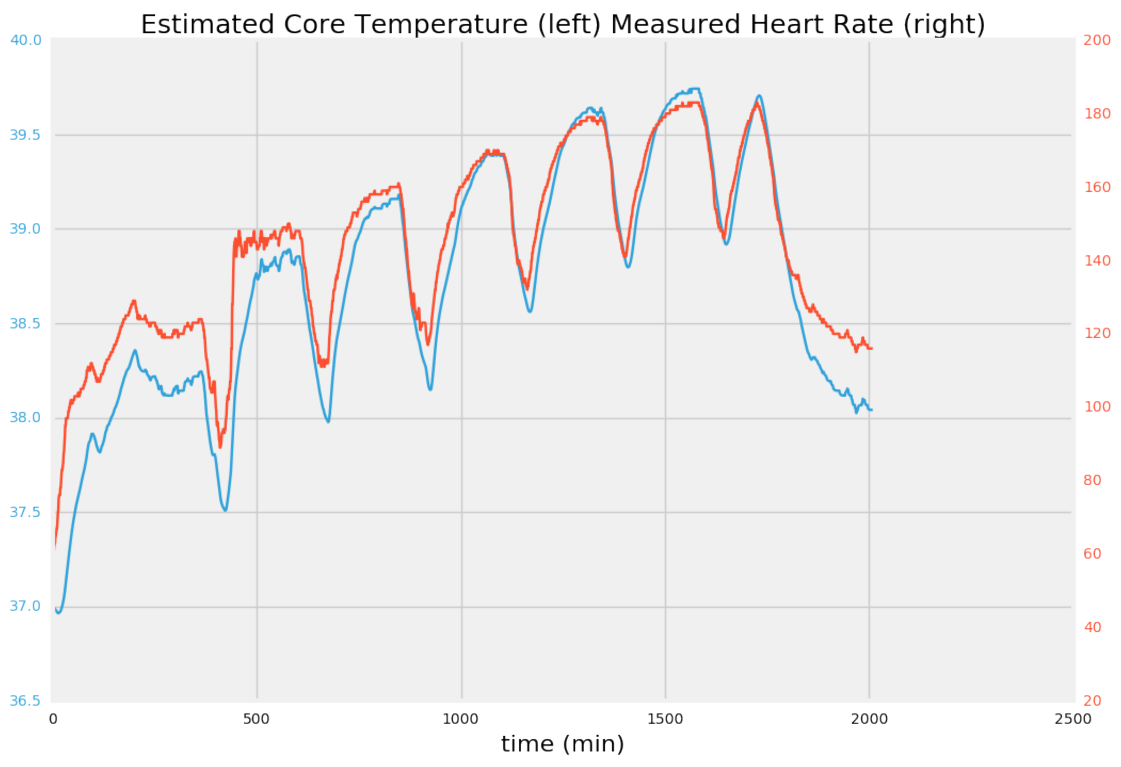
In [129]:
```
# file heart rate
[Told, Pold, H, Q, _] = initKalmanFilter(37, R=1000)
count = 0

KK = []
TT = []
PP = []
while count < HR.size:
    [t,p,k] = kalmanFilter(HR[count], Told, Pold, Q, H, R[count])

    KK.append(k)
    TT.append(t)
    PP.append(p)
    Told = t
    Pold = p
    count = count + 1

KK = np.squeeze(np.array(KK))
TT = np.squeeze(np.array(TT))
PP = np.squeeze(np.array(PP))
```

```
plot(TT[:,0], xlabel = 'time (min)',
              ylabel = 'core temperature ' + '('+r'$\degree C$)',
              title  = 'Estimated Core Temperature')

plot(HR, xlabel = 'time (min)',
           ylabel = 'heart rate (beats/min)',
           title  = 'Measured Heart Rate')

plot(R, xlabel = 'time (min)',
          ylabel = 'variance of heart rate',
          title  = 'Estimated Heart Rate Variance')

plotyy(TT[:,0], HR, xlabel='time (min)',
       title='Estimated Core Temperature (left) Measured Heart Rate (righ
```

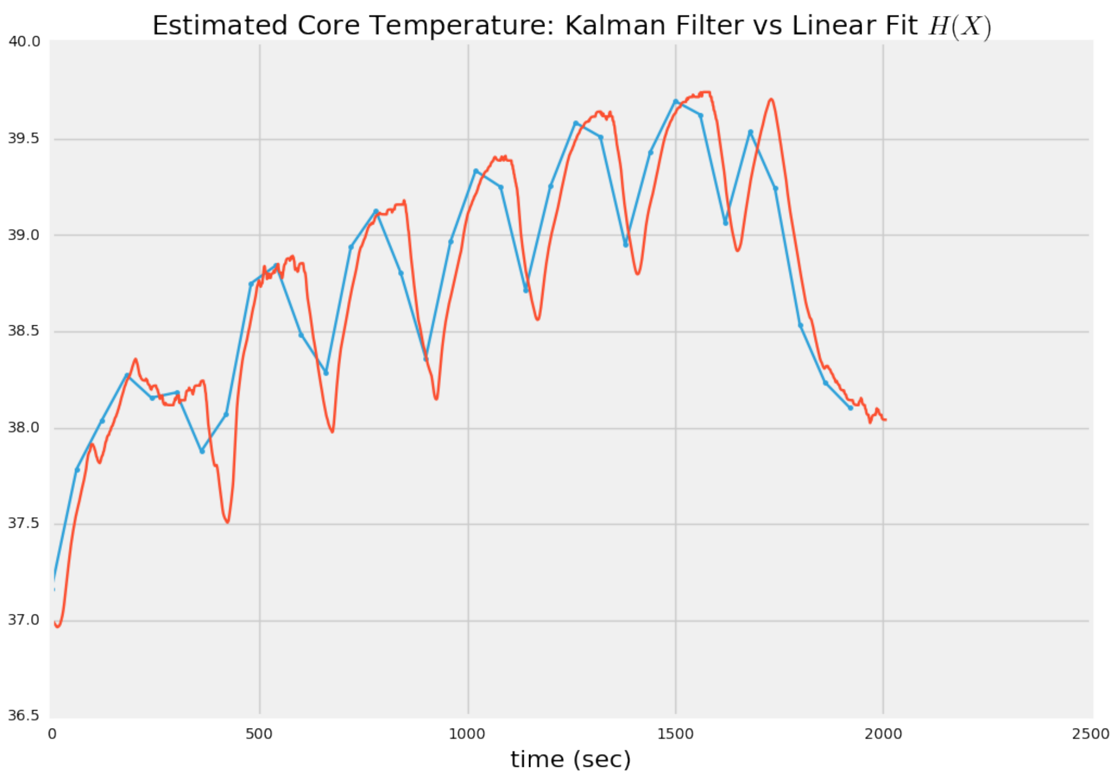



Measured Heart Rate
heart rate (beats/min)
time (min)
200
180
160
140
120
100
80
60
0
500
1000
1500
2000
2500


Estimated Heart Rate Variance
variance of heart rate
time (min)
600
500
400
300
200
100
0
0
500
1000
1500
2000
2500



```
In [130]: Hslope = 39.3701
          Hintercept = -1381.6890

          x = HR
          y = (x - Hintercept)/Hslope

          numBlock = int(len(HR) / winLen)
          t = TT[0:winLen*numBlock,0].reshape(numBlock,winLen).mean(axis=1)
          time = np.arange(len(HR))
          time60 = np.arange(len(t))*60

          p = setPlot(xlabel='time (sec)',
                   title='Estimated Core Temperature: Kalman Filter vs Linear Fit $
          p.plot(time60, t, '.-');
          p.plot(time, TT[:,0], );
```

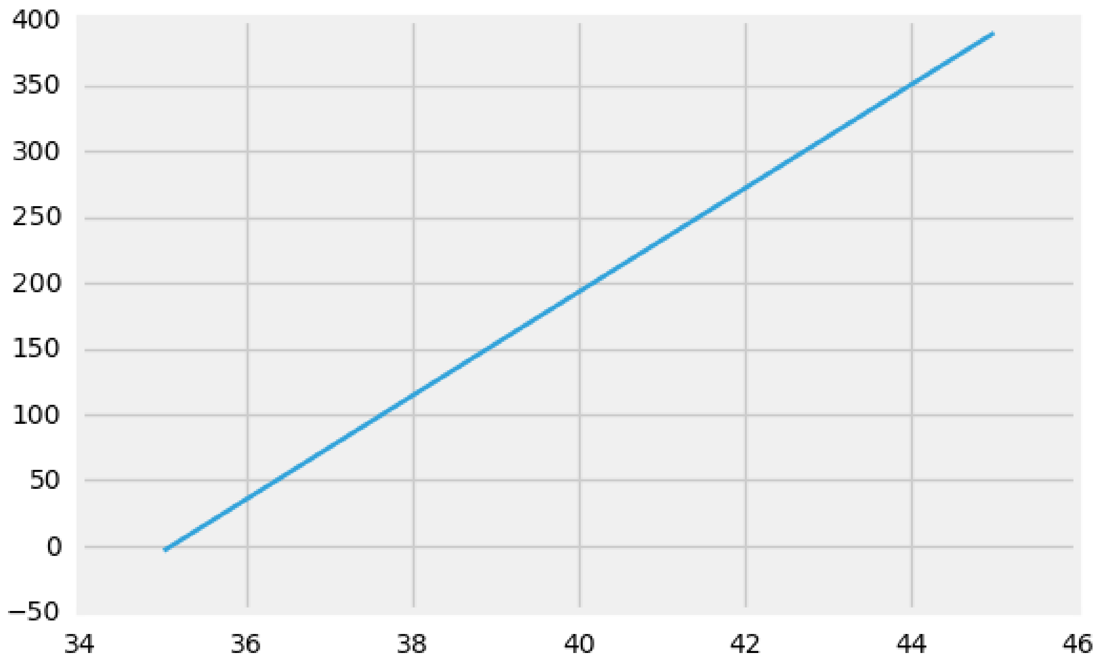


```
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```

1.4 Debug Code

```
In [120]: Hslope = 39.3701
          Hintercept = -1381.6890
          x = np.arange(35,45,.001)
          y = Hslope*x + Hintercept
          plt.plot(x,y)
```

```
Out[120]: [<matplotlib.lines.Line2D at 0x118321c50>]
```



```
In [ ]:
```