comparative-study

December 2, 2018

```
In [37]: import seaborn as sns
         import pandas as pd
         import numpy as np
         from matplotlib import pyplot as plt, patches
         import matplotlib
         from scipy.signal import argrelextrema, resample
         from scipy.stats import describe
         from sklearn.preprocessing import PolynomialFeatures
         from sklearn.linear_model import LinearRegression, Ridge

         # Ignore some warnings.
         import warnings
         warnings.simplefilter(action='ignore', category=FutureWarning)

         # Set graphing params.
         sns.set()
         matplotlib.rcParams['figure.figsize'] = (25.0, 10.0)
```

0.0.1 Munging functions

These are helper functions. Columns are renamed to a standard format:

Header	Value
f	Raw force (N)
mdia	Muscle diameter (in)
msgtype	N/A
pwm	Digital PWM value
rc	Constant
rw	Constant
t	Temperature
t0	Starting temperature
timestamp	Absolute time (s)

```
In [2]: # Renames columns to standard format listed above,
        # Normalizes timestamps relative to 0 (instead of experiment start time),
        # Fills NaNs, inverts force.
```

```
        def clean(df):
            df.columns = ["f", "mdia", "msgtype", "pwm", "rc", "rw", "t", "t0", "timestamp"]
            if(df.iloc[0].timestamp > 0):
                df.timestamp -= df.iloc[0].timestamp
            df.pwm.replace(to_replace=0, value=np.NaN, inplace=True)
            df.f = -df.f
            return df.fillna(0)

In [3]: # Grabs rolling average with default lag of 300-points.
        # You can set #points in rolling window with `window` param, and
        # select whether to add the rolling average column to the given
        # dataframe with `add_col` param.
        def ma(df, window=500, add_col=True):
            m = df.f.rolling(window=window).mean()
            if add_col:
                df['f_ra'] = m
                return df.fillna(0)
            return m

In [4]: # Grabs minima and maxima.
        # This is its own function only because this is an expensive operation.
        # Note this function operates on the rolling average, not the raw force, to prevent nois
        def get_extrema(df, order=2000, ra=False):
            col = 'f_ra' if ra else 'f'
            maxima = argrelextrema(df[col].values, np.greater, order=order)
            minima = argrelextrema(df[col].values, np.less, order=order)
            return minima, maxima

In [5]: # Labels cycles based on local (force) minima and maxima.
        # Points not belonging to a valid cycle, e.g. at start and end of experiment,
        # are labeled with <cycle# = -1>.
        def label_cycles(df, minima, maxima, abs_min=None, abs_max=None, plot_boxes=True):
            # Plot line connecting all minima, and another line connecting all maxima.
            fig, ax = plt.subplots(figsize=(25, 6), nrows=1, ncols=1)
            if abs_min and abs_max:
                ax.plot(df.f, alpha=0.3)
                ax.plot(df.f.iloc[abs_max[0].tolist()], '--')
                ax.plot(df.f.iloc[abs_min[0].tolist()], '--')

            # Label cycles.
            df['cycle'] = -1
            for i in range(minima[0].size-1):
                rng = list(range(minima[0][i], minima[0][i+1]))
                df.iloc[rng, df.columns.get_loc('cycle')] = i # I don't know why `iloc` succeeds

            # Draw bounding boxes around each cycle.
            if plot_boxes:
                for i in minima[0].tolist():
```

```
                ax.add_patch(
                    patches.Rectangle((i, df.f.min()), df[df.cycle==i].shape[0], df.f.max()-
                )
        df.cycle = df.cycle.astype(np.int64) # Just to be safe
        return df
```

In [128]:
```
# Squashes periods down to a constant for all rows in a dataset.
# You can set the desired period with param `period`.
# If you don't, the function just chooses the shortest period.
def squash_periods(df, period=-1, plot_cycles=True):
    c_to_p = {} # maps cycles to their periods
    columns=["f", "mdia", "msgtype", "pwm", "rc", "rw", "t", "t0", "timestamp", "f_ra"
    fig, ax = plt.subplots(figsize=(25, 12), nrows=1, ncols=1)
    df.cycle, X = df.cycle.astype(np.int64), pd.DataFrame(columns=columns)

    for c in range(df.cycle.max()):
        c_to_p[c] = df[df.cycle==c].shape[0]
    if period<0:
        period = int(min(c_to_p.values()))

    for c in range(df.cycle.max()):
        C = pd.DataFrame(resample(df[df.cycle==c], period), columns=columns)
        C.cycle = c
        X = pd.concat([X, C], ignore_index=True)
        if plot_cycles:
            ax.plot(C.f_ra, color=plt.cm.RdYlBu(c/df.cycle.max()))
    X.timestamp = df.timestamp.iloc[:X.timestamp.size,]

    if plot_cycles:
        cmap = plt.cm.ScalarMappable(
            norm=matplotlib.colors.Normalize(vmin=0, vmax=df.cycle.max()),
            cmap=plt.get_cmap('RdYlBu', df.cycle.max())
        )
        cmap.set_array([])
        plt.colorbar(cmap, orientation="horizontal", pad=0.1)

    return X, c_to_p
```

In [78]:
```
# Prints descriptive stats about a 1-D dataset.
# (Mean, stdev, first quartiles, and index of dispersion if `dispersion=True`).
def stats(series, dispersion=True):
    print("SUMMARY\n")
    if dispersion:
        mean, var = np.mean(series.values), np.var(series.values)
        print("index of dispersion\t%f\n%s" % (var/mean, '-'*50))
    print(series.describe(include='all'))
```

In [8]:
```
# Takes difference of each point by given lag (default=1 cycle, not 1 pt).
def difference(df, periods=None, cols_to_diff=['f', 't', 'f_ra']):
```

```
            if not periods:
                periods = df[df.cycle==1].size
            return df[cols_to_diff].diff(periods=periods).fillna(0)
```

```
In [344]: def reg_detrend(df, x='timestamp', y='f', order=2):
              reg, poly = LinearRegression(), PolynomialFeatures(order)
              X_t = poly.fit_transform(df[x].values.reshape(-1,1))
              reg.fit(X_t, df[y].values.reshape(-1,1))
              return reg.predict(X_t)
```

```
In [173]: def average(X):
              cycles, n = X.cycle.max(), X[X.cycle==1].shape[0]
              means = [sum(X.f_det.iloc[j::n])/cycles for j in range(n)]
              return means
```

```
In [319]: def power(pwm, R, r, periods=None):
              i = (pwm/255.0)/(R+r)
              P = i**2 * R
              fig, ax1 = plt.subplots()
              ax1.set_xlabel('Cycle #')
              ax1.set_ylabel('Power')
              ax1.plot(P)
              ax1.tick_params(axis='y')

              if periods.any():
                  ax2 = ax1.twinx()
                  ax2.set_ylabel('Period', color='r')
                  ax2.plot(periods, color='r')
                  ax2.tick_params(axis='y') #, labelcolor='r')
                  return (ax1, ax2, fig)
              return P
```

1 Datasets

We inspect trials from `11/06` and `11/16`. For `11/06`:

$$d = 1 in \text{Duty cycle} = 150 ms f_{min} = 5 N f_{max} = 40 NR = 27.5 \Omega T = 4.5 hrsc = 40$$

The `11/16` trial is similar, but lasts for

$$c = 100$$

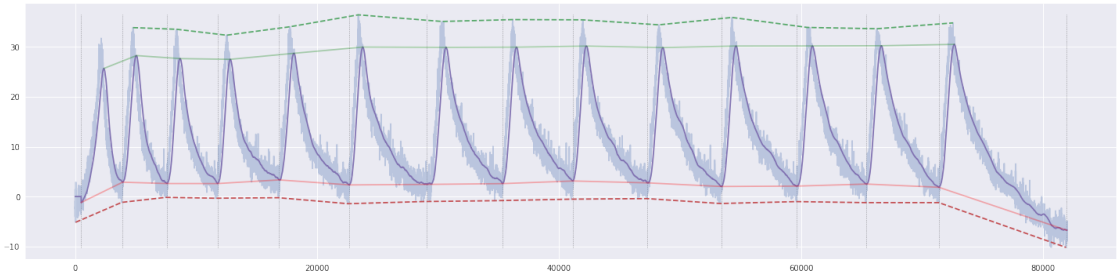
possibly due to conductive paste in the silicon.

```
In [10]: # X_25 is from a 3/4"(?) diameter muscle back in September.
         X_25 = pd.DataFrame.from_dict(pd.read_msgpack("data_2018-09-25-14-24-26.msgpack"))
         X_25.insert(1, "mdia", 0.75)
         X_25 = ma(clean(X_25))
```

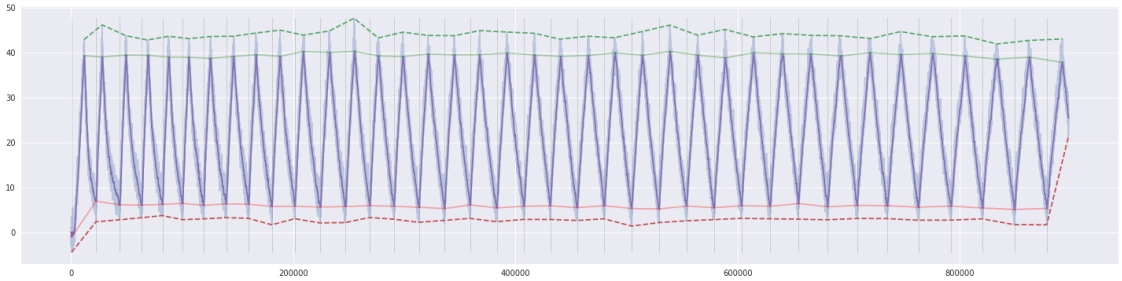
```
In [11]: # X_40 is a dataset with estimated 40 cycles.
         X_40 = ma(clean(pd.DataFrame.from_dict(
             pd.read_msgpack("data_2018-11-06-22-59-57.msgpack"))))

In [12]: # X_100 is a dataset with estimated 100 cycles and conductive paste in silicon skin.
         X_100 = ma(clean(pd.DataFrame.from_dict(
             pd.read_msgpack("data_2018-11-16-21-07-24.msgpack"))), window=150)

In [199]: # The only difference with these two tests is that they had half a tube of thermal pas
          # vs the last muscle that had one tube for one muscle.
          # We got ~40 cycles in 2.5 hours (still an improvement from ~30 in 4 hours with no the
          X_41 = ma(clean(pd.DataFrame.from_dict(
              pd.read_msgpack("data_2018-12-01-18-01-00.msgpack"))), window=150)

In [193]: # The only difference with these two tests is that they had half a tube of thermal pas
          # vs the last muscle that had one tube for one muscle.
          # We got ~40 cycles in 2.5 hours (still an improvement from ~30 in 4 hours with no the
          X_42 = ma(clean(pd.DataFrame.from_dict(
              pd.read_msgpack("data_2018-12-01-18-01-46.msgpack"))), window=150)
```

1.1 Label cycles.

```
In [13]: minima, maxima = get_extrema(X_25, ra=True) # n~=2000 is a safe order size for extrema:
         abs_min, abs_max = get_extrema(X_25, order=3000, ra=False)

In [14]: X_25 = label_cycles(X_25, minima, maxima, abs_min=abs_min, abs_max=abs_max)
         plt.plot(X_25.f_ra); plt.plot(X_25.f_ra.iloc[minima], color='r', alpha=0.3); plt.plot(X

Out[14]: [<matplotlib.lines.Line2D at 0x7fac6b89cac8>]
```

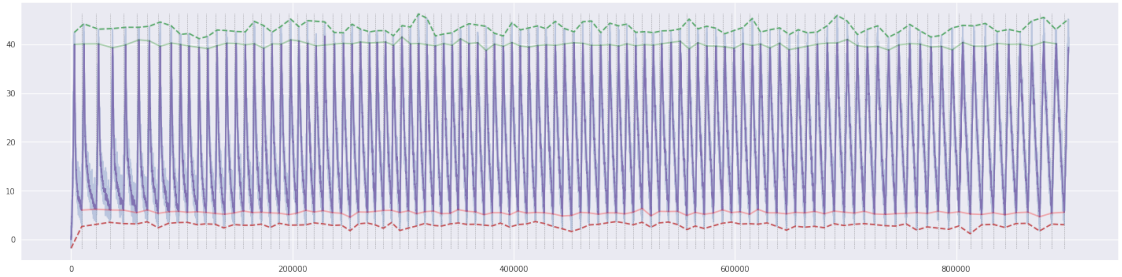



```
In [121]: minima, maxima = get_extrema(X_40, ra=True) # n~=2000 is a safe order size for extrema
          abs_min, abs_max = get_extrema(X_40, order=3000, ra=False)

In [122]: X_40 = label_cycles(X_40, minima, maxima, abs_min=abs_min, abs_max=abs_max)
          plt.plot(X_40.f_ra); plt.plot(X_40.f_ra.iloc[minima], color='r', alpha=0.3); plt.plot(

Out[122]: [<matplotlib.lines.Line2D at 0x7fac807b6eb8>]
```



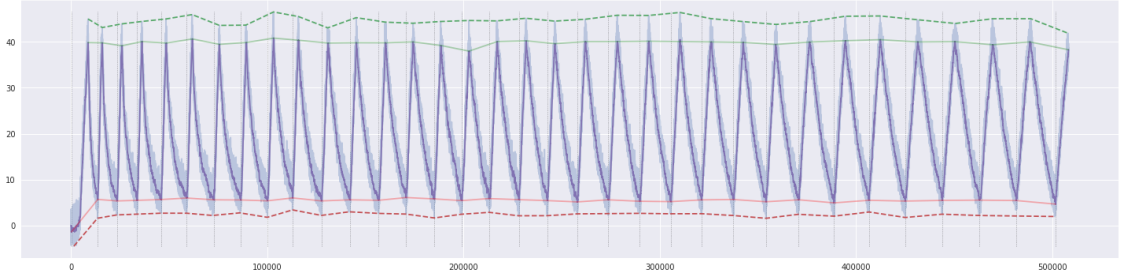

```
In [17]: minima, maxima = get_extrema(X_100, ra=True) # n~=2000 is a safe order size for extrema
         abs_min, abs_max = get_extrema(X_100, order=3000, ra=False)

In [18]: X_100 = label_cycles(X_100, minima, maxima, abs_min=abs_min, abs_max=abs_max)
         plt.plot(X_100.f_ra); plt.plot(X_100.f_ra.iloc[minima], color='r', alpha=0.3); plt.plot
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x7fac8b6b0358>]
```

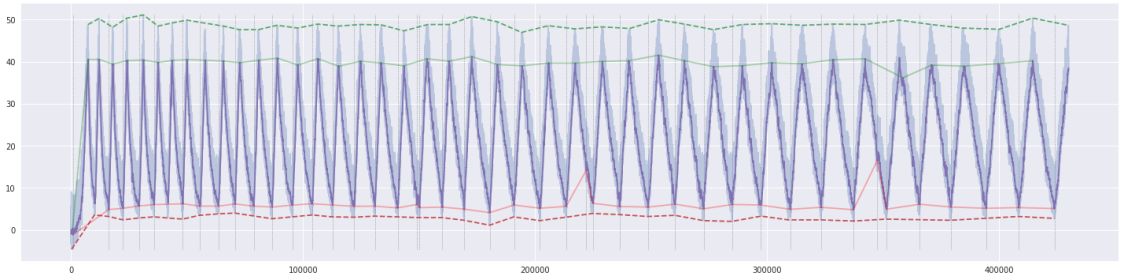



```
In [207]: minima, maxima = get_extrema(X_41, ra=True) # n~=2000 is a safe order size for extrema
          abs_min, abs_max = get_extrema(X_41, order=3000, ra=False)

In [208]: X_41 = label_cycles(X_41, minima, maxima, abs_min=abs_min, abs_max=abs_max)
          plt.plot(X_41.f_ra); plt.plot(X_41.f_ra.iloc[minima], color='r', alpha=0.3); plt.plot(
```

```
Out[208]: [<matplotlib.lines.Line2D at 0x7fac8006bba8>]
```



```
In [224]: minima, maxima = get_extrema(X_42, order=800, ra=True) # n~=2000 is a safe order size
          abs_min, abs_max = get_extrema(X_42, order=3000, ra=False)

In [225]: X_42 = label_cycles(X_42, minima, maxima, abs_min=abs_min, abs_max=abs_max)
          plt.plot(X_42.f_ra); plt.plot(X_42.f_ra.iloc[minima], color='r', alpha=0.3); plt.plot(

Out[225]: [<matplotlib.lines.Line2D at 0x7fac6e32fda0>]
```



1.2 Fix stationarity (period).

Mean, variance, and autocorrelation of each cycle are not quite constant. We can't usefully determine correlation between variables until stationarity is maintained. We can eliminate stationarity by differencing data:

$$Y_i = Z_i - Z_{i-1} \quad \forall i \in (0, \ldots, T)$$

or by fitting a trend and removing residuals:

$$Y = a + bln\, X$$

We'd also like to enforce constant period (and add our detrended period back in later) for robust regression:

$$P_t = P_{t+1} \quad \forall t \in (0, (T-1))$$

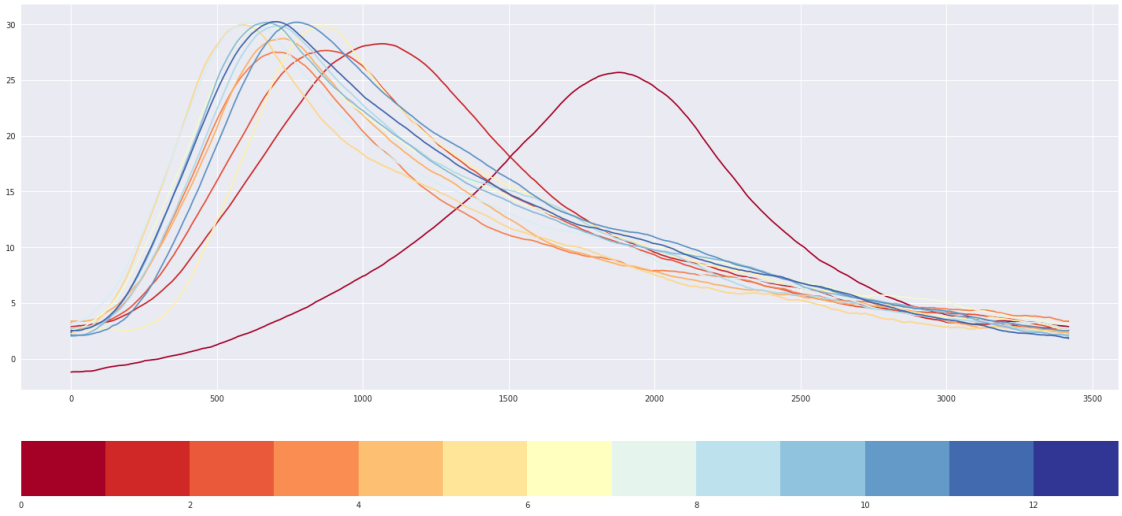
First, adjust for seasonality.

1.2.1 Fix 36-cycle periods.

We analyze variance with index of dispersion (normalized to mean):

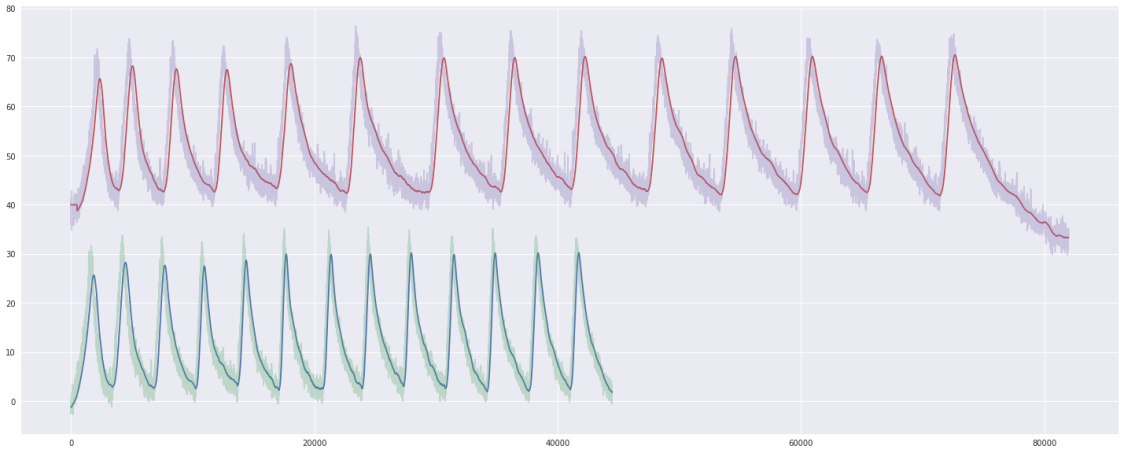
$$D = \frac{\sigma^2}{\mu}$$

```
In [129]: X_25.cycle = X_25.cycle.astype(np.int64)
          X_25S, c_to_p25 = squash_periods(X_25)
          X_25S.cycle = X_25S.cycle.astype(np.int64)
```



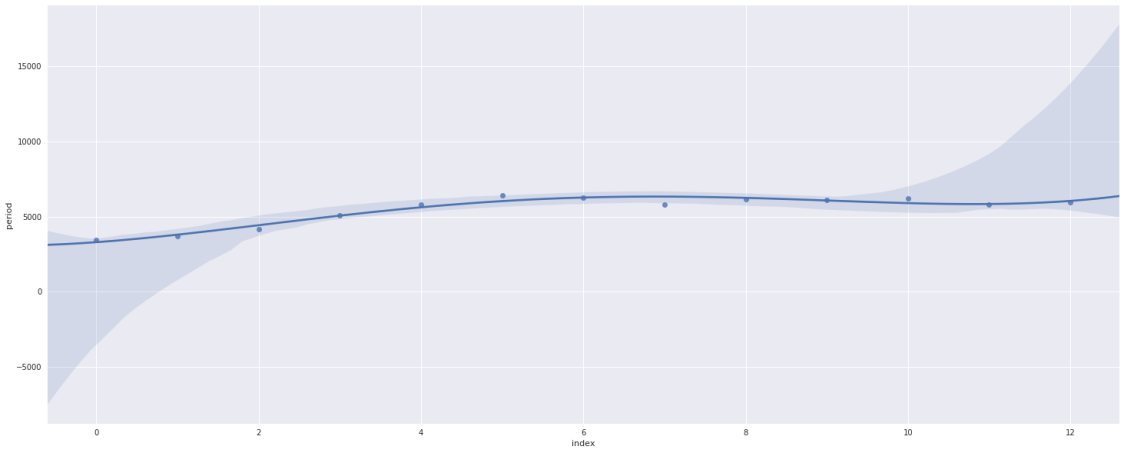
In [38]:
```
plt.plot(X_25S.f_ra); plt.plot(X_25S.f, alpha=0.3)
plt.plot(X_25.f_ra+40); plt.plot(X_25.f+40, alpha=0.3)
```

Out[38]: [<matplotlib.lines.Line2D at 0x7fac8805ce80>]



In [39]:
```
temp = pd.DataFrame(c_to_p25, index=[0]).transpose().reset_index(); temp.columns = ['in
sns.regplot('index', 'period', temp, fit_reg=True, order=4)
```

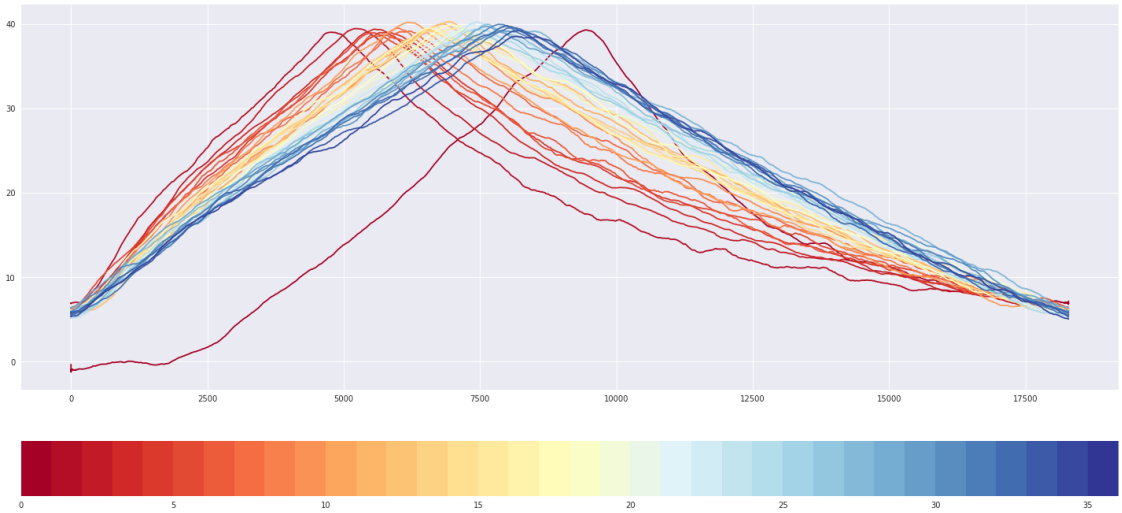
Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac8806fdd8>




```
In [22]: stats(pd.Series(list(c_to_p25.values())))

SUMMARY

index of dispersion        180.029901
-------------------------------------------------
count      13.000000
mean     5453.153846
std      1031.281553
min      3420.000000
25%      5084.000000
50%      5814.000000
75%      6159.000000
max      6414.000000
dtype: float64
```

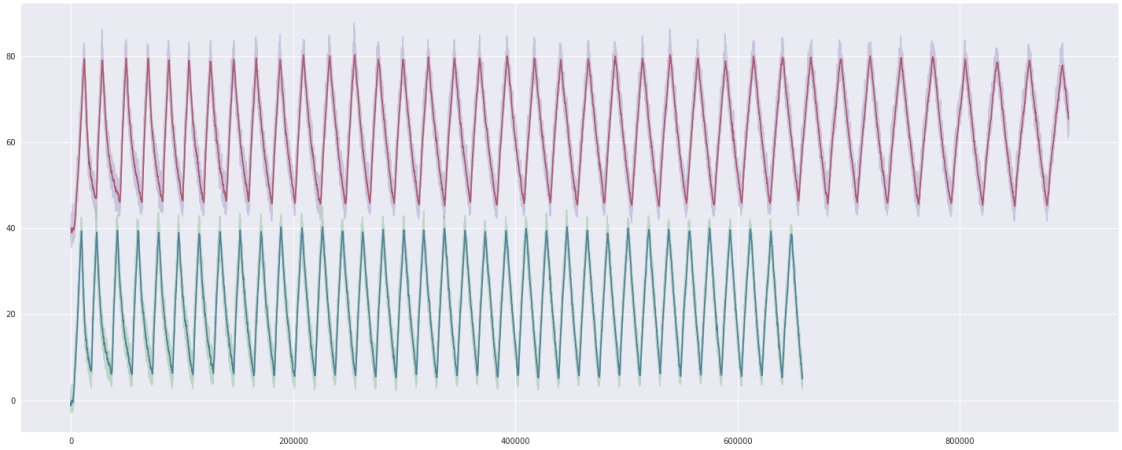
1.2.2 Fix 36-cycle periods.

```
In [130]: X_40.cycle = X_40.cycle.astype(np.int64)
          X_40S, c_to_p40 = squash_periods(X_40)
          X_40S.cycle = X_40S.cycle.astype(np.int64)
```



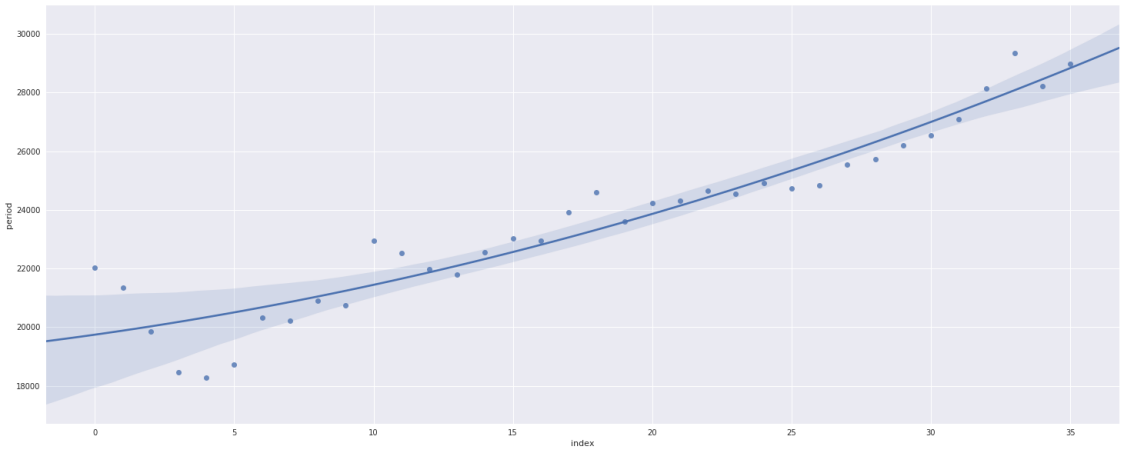
In [40]:
```
plt.plot(X_40S.f_ra); plt.plot(X_40S.f, alpha=0.3)
plt.plot(X_40.f_ra+40); plt.plot(X_40.f+40, alpha=0.3)
```

Out[40]:
```
[<matplotlib.lines.Line2D at 0x7fac80d99e80>]
```



In [41]:
```
temp = pd.DataFrame(c_to_p40, index=[0]).transpose().reset_index(); temp.columns = ['in
sns.regplot('index', 'period', temp, fit_reg=True, order=2)
```

Out[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fac80d9dc50>
```




```
In [26]: stats(pd.Series(list(c_to_p40.values())))

SUMMARY

index of dispersion        347.846432
-------------------------------------------------
count       36.000000
mean     23579.416667
std       2904.541148
min      18290.000000
25%      21685.750000
50%      23764.000000
75%      25065.000000
max      29356.000000
dtype: float64
```

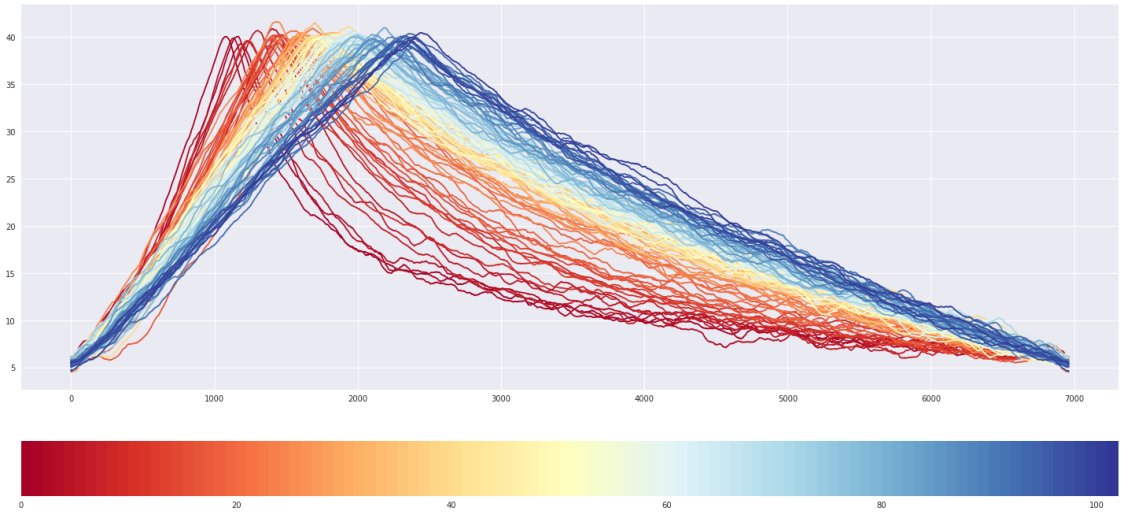
1.2.3 Fix 103-cycle periods.

We analyze variance with index of dispersion (normalized to mean):

$$D = \frac{\sigma^2}{\mu}$$

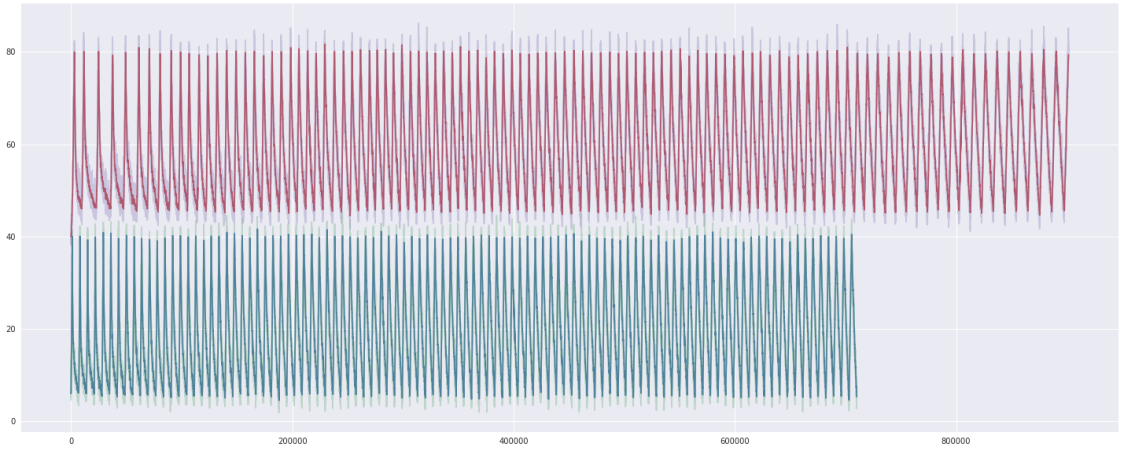
Evaporation of ethanol over time might contribute to linearity in cooling (falling edge) of cycles, since silicone has no phase change.

```
In [164]: X_100.cycle = X_100.cycle.astype(np.int64)
          X_100S, c_to_p100 = squash_periods(X_100)
          X_100S.cycle = X_100S.cycle.astype(np.int64)
```



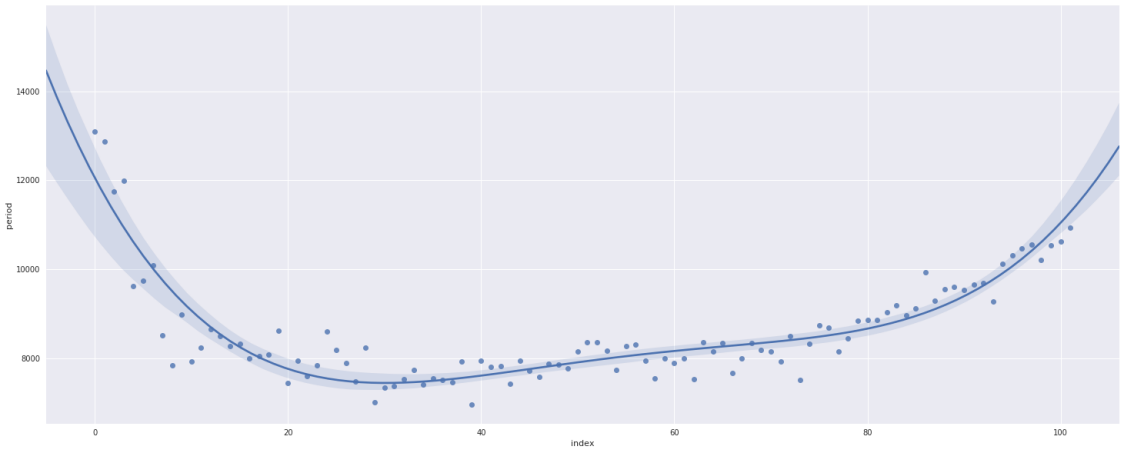

```
In [165]: plt.plot(X_100S.f_ra); plt.plot(X_100S.f, alpha=0.3)
          plt.plot(X_100.f_ra+40); plt.plot(X_100.f+40, alpha=0.3)
```

```
Out[165]: [<matplotlib.lines.Line2D at 0x7fac8011ada0>]
```




```
In [43]: temp = pd.DataFrame(c_to_p100, index=[0]).transpose().reset_index(); temp.columns = ['i
         sns.regplot('index', 'period', temp, fit_reg=True, order=4)
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac80d1c320>
```




```
In [30]: stats(pd.Series(list(c_to_p100.values())))

SUMMARY

index of dispersion        162.000749
-------------------------------------------------
count      102.000000
mean      8595.049020
std       1185.829061
min       6962.000000
25%       7844.750000
50%       8236.000000
75%       9024.750000
max      13097.000000
dtype: float64
```

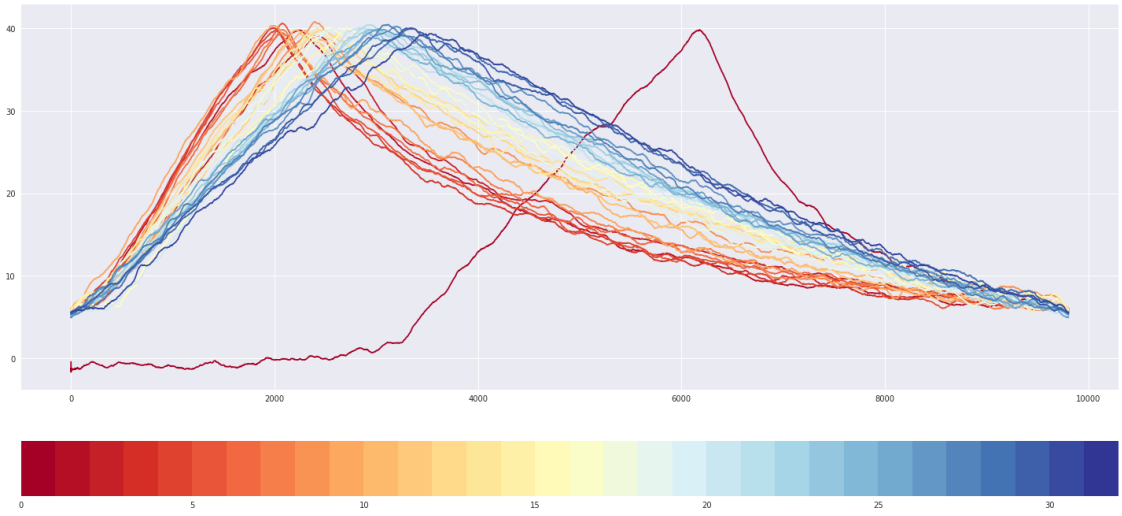
The nonlinearity all the way through `c=20` is much more pronounced than in the `36`-cycle dataset. It's possible that the conductive paste reduced period variance and dispersed temperature more uniformly, but also decreased output force by dissipating heat. We can tell by analyzing `t` in `0 < c < 30` as a reasonable approximation:

1.2.4 Fix `32`-cycle periods.

We analyze variance with index of dispersion (normalized to mean):

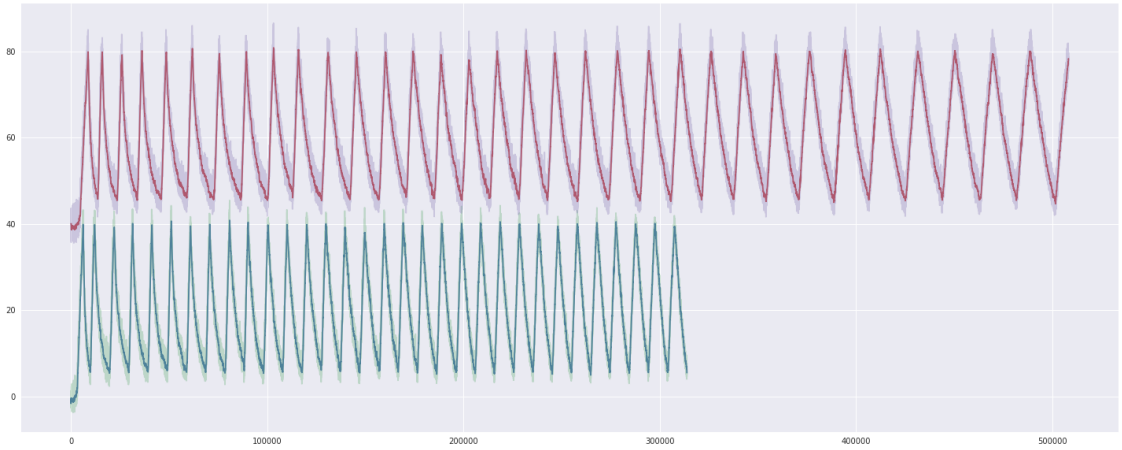
$$D = \frac{\sigma^2}{\mu}$$

```
In [213]: X_41.cycle = X_41.cycle.astype(np.int64)
          X_41S, c_to_p41 = squash_periods(X_41)
          X_41S.cycle = X_41S.cycle.astype(np.int64)
```



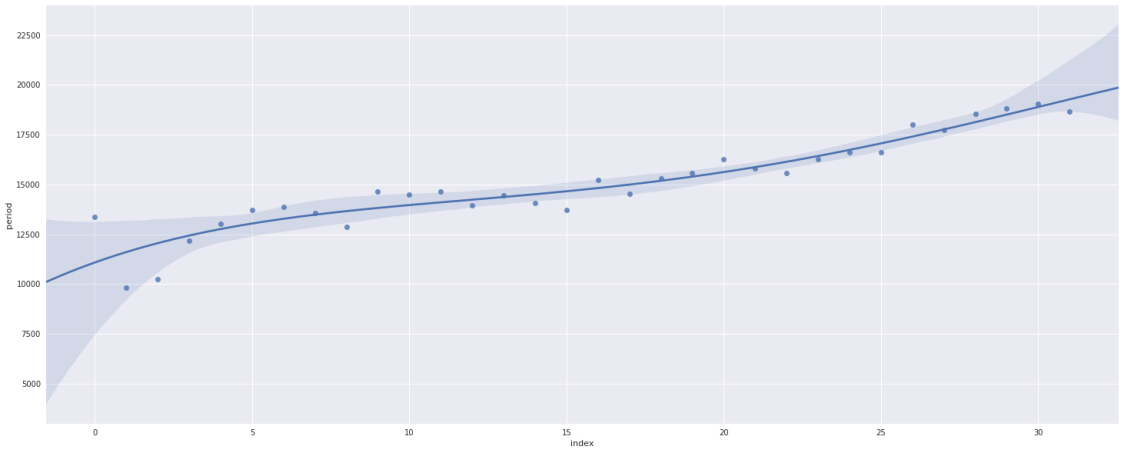
In [214]:
```
plt.plot(X_41S.f_ra); plt.plot(X_41S.f, alpha=0.3)
plt.plot(X_41.f_ra+40); plt.plot(X_41.f+40, alpha=0.3)
```

Out[214]: [<matplotlib.lines.Line2D at 0x7fac8037f9e8>]



In [215]:
```
temp = pd.DataFrame(c_to_p41, index=[0]).transpose().reset_index(); temp.columns = ['i
sns.regplot('index', 'period', temp, fit_reg=True, order=4)
```

Out[215]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac8034a160>



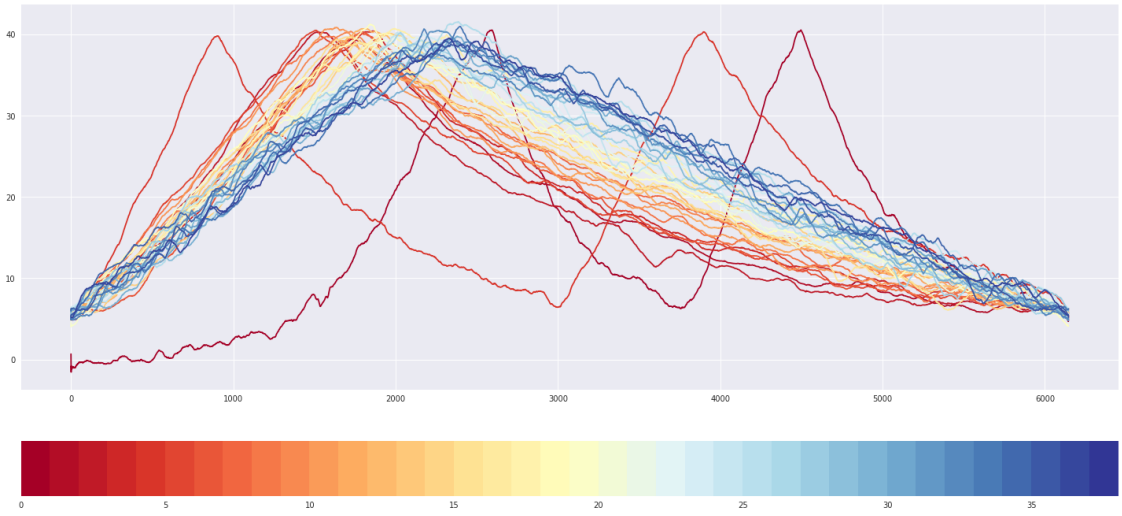

```
In [227]: stats(pd.Series(list(c_to_p41.values())))

SUMMARY

index of dispersion        336.068195
-------------------------------------------------
count       32.000000
mean     15041.718750
std       2284.318099
min       9806.000000
25%      13716.000000
50%      14658.500000
75%      16362.750000
max      19054.000000
dtype: float64
```

1.2.5 Fix 42-cycle periods.

We analyze variance with index of dispersion (normalized to mean):

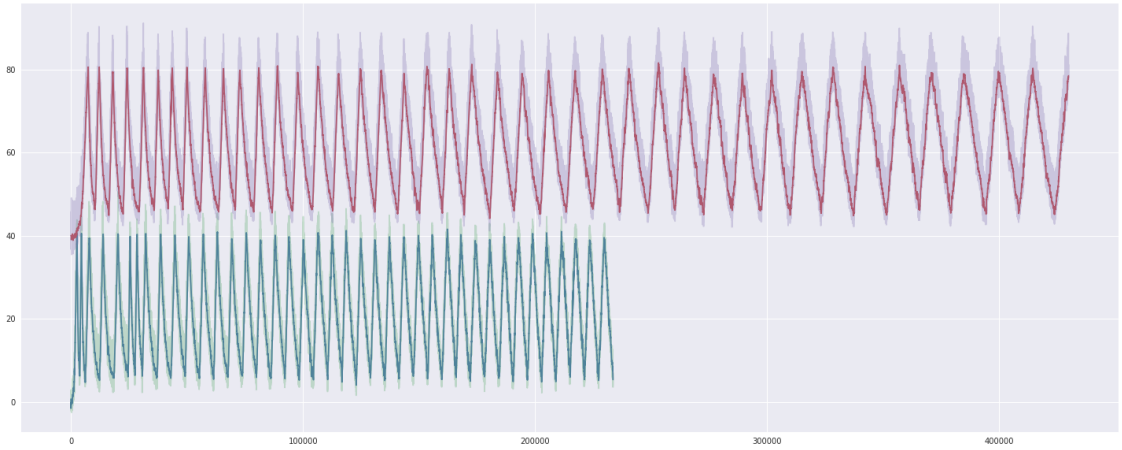
$$D = \frac{\sigma^2}{\mu}$$

```
In [220]: X_42.cycle = X_42.cycle.astype(np.int64)
          X_42S, c_to_p42 = squash_periods(X_42)
          X_42S.cycle = X_42S.cycle.astype(np.int64)
```



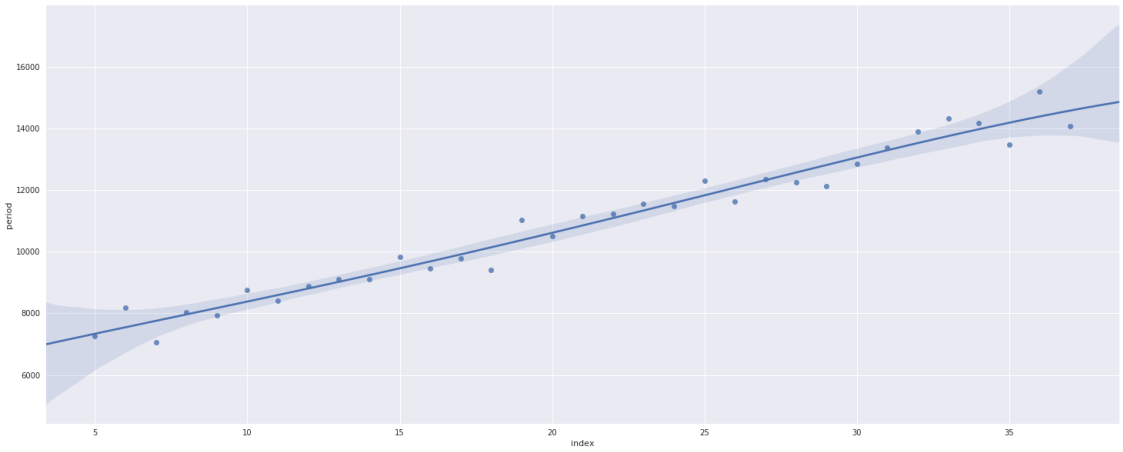
In [221]:
```
plt.plot(X_42S.f_ra); plt.plot(X_42S.f, alpha=0.3)
plt.plot(X_42.f_ra+40); plt.plot(X_42.f+40, alpha=0.3)
```

Out[221]: [<matplotlib.lines.Line2D at 0x7fac3f964208>]



In [361]:
```
temp = pd.DataFrame(c_to_p42, index=[0]).transpose().reset_index().iloc[5::]; temp.col
sns.regplot('index', 'period', temp, fit_reg=True, order=4)
```

Out[361]: <matplotlib.axes._subplots.AxesSubplot at 0x7fac5cfc8438>



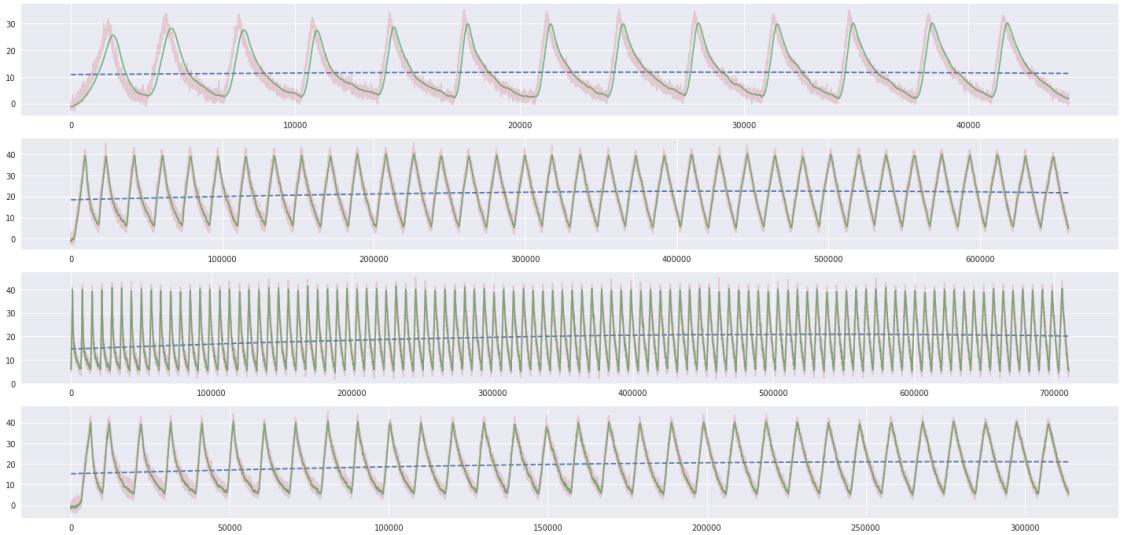
1.3 Detrend data.

by differencing:

$$\Delta X_t = X_t - X_{t-1}$$

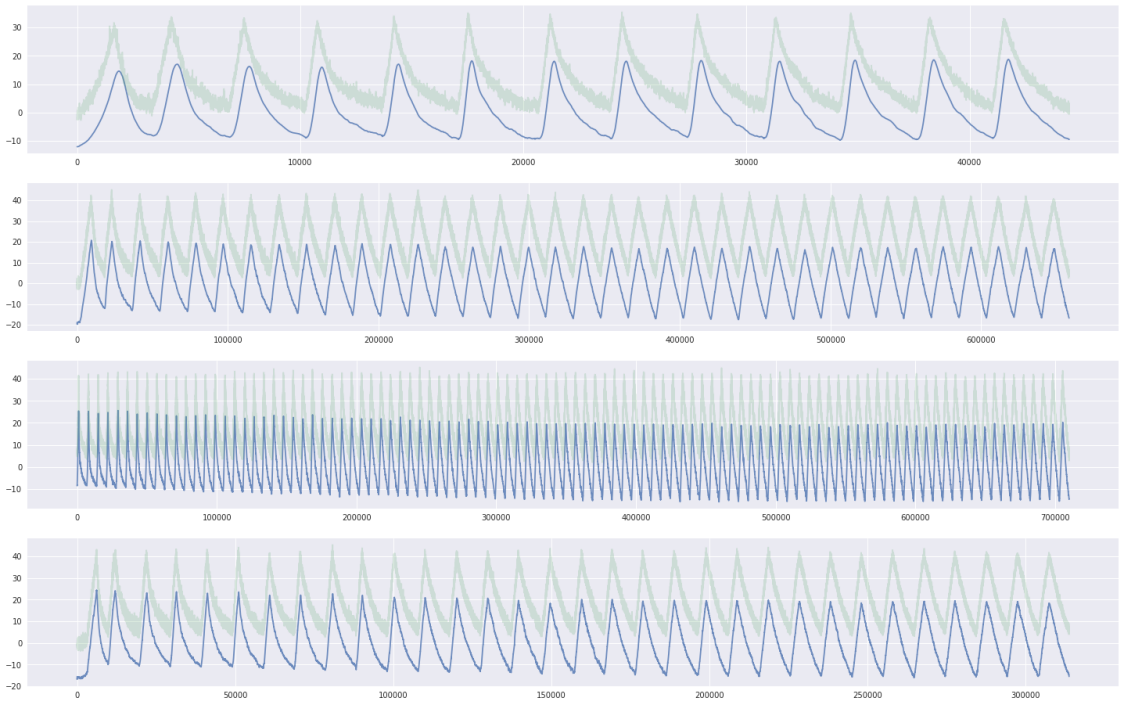
and by regression. Pre-detrend:

```
In [228]: reg_25 = reg_detrend(X_25S).flatten(); X_25S['f_det'] = X_25S.f_ra - reg_25
          reg_40 = reg_detrend(X_40S).flatten(); X_40S['f_det'] = X_40S.f_ra - reg_40
          reg_100 = reg_detrend(X_100S).flatten(); X_100S['f_det'] = X_100S.f_ra - reg_100
          reg_41 = reg_detrend(X_41S).flatten(); X_41S['f_det'] = X_41S.f_ra - reg_41

In [231]: fig, ax = plt.subplots(figsize=(25, 12), nrows=4, ncols=1)
          ax.flat[0].plot(reg_25, '--'); ax.flat[0].plot(X_25S.f_ra, alpha=0.8); ax.flat[0].plot
          ax.flat[1].plot(reg_40, '--'); ax.flat[1].plot(X_40S.f_ra, alpha=0.8); ax.flat[1].plot
          ax.flat[2].plot(reg_100, '--'); ax.flat[2].plot(X_100S.f_ra, alpha=0.8); ax.flat[2].pl
          ax.flat[3].plot(reg_41, '--'); ax.flat[3].plot(X_41S.f_ra, alpha=0.8); ax.flat[3].plot

Out[231]: [<matplotlib.lines.Line2D at 0x7fac6e0e10f0>]
```



After detrending:

```
In [241]: fig, ax = plt.subplots(figsize=(25, 16), nrows=4, ncols=1)
          ax.flat[0].plot(X_25S.f_det, alpha=0.8); ax.flat[0].plot(X_25S.f, alpha=0.2)
          ax.flat[1].plot(X_40S.f_det, alpha=0.8); ax.flat[1].plot(X_40S.f, alpha=0.2)
          ax.flat[2].plot(X_100S.f_det, alpha=0.8); ax.flat[2].plot(X_100S.f, alpha=0.2)
          ax.flat[3].plot(X_41S.f_det, alpha=0.8); ax.flat[3].plot(X_41S.f, alpha=0.2)

Out[241]: [<matplotlib.lines.Line2D at 0x7fac6da892b0>]
```

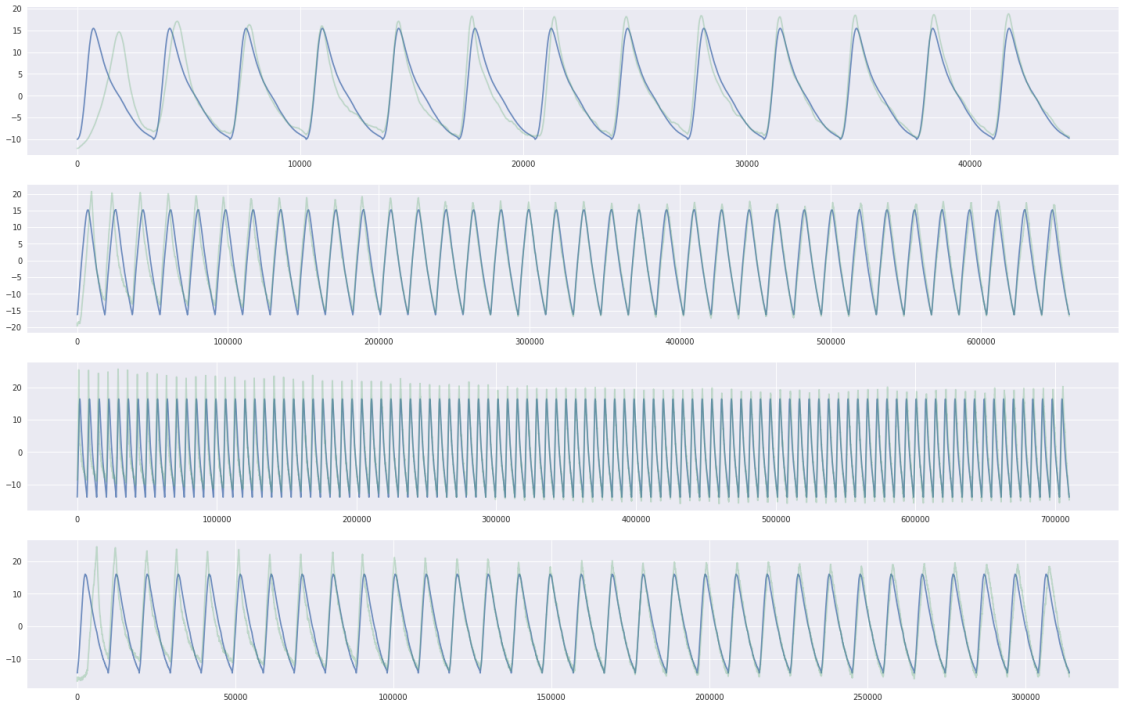


2 Average seasonality.

Gives us a reasonably robust seasonal component, from which we can determine multiplicative noise. For stationary cycle (rising edge to rising edge) in n discrete time chunks, this is simply

$$\begin{matrix} \mathbb{E}[f_{t_0}] \\ \vdots \\ \mathbb{E}[f_{t_n}] \end{matrix}$$

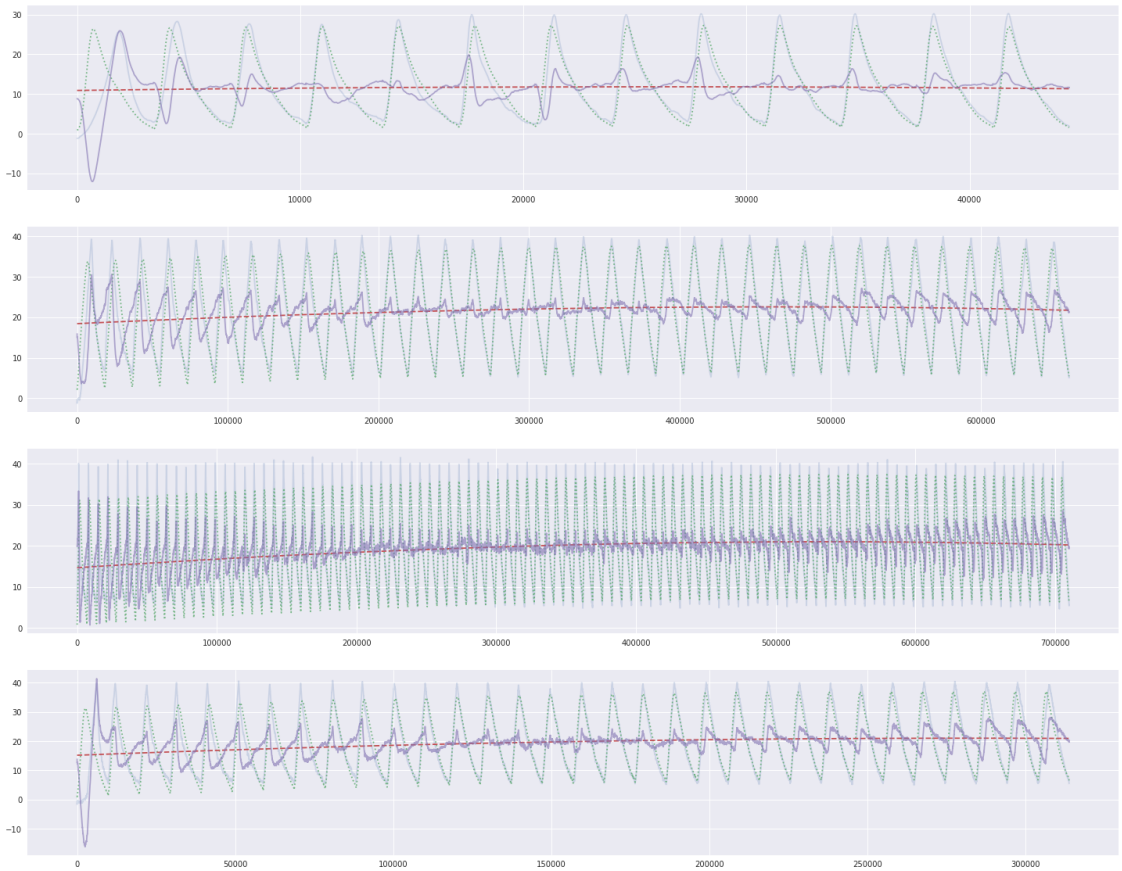
```
In [242]: fig, ax = plt.subplots(figsize=(25, 16), nrows=4, ncols=1)
          for i, X in enumerate([X_25S, X_40S, X_100S, X_41S]):
              ax.flat[i].plot(average(X) * (X.cycle.max()+1), alpha=0.8); ax.flat[i].plot(X.f_de
```



2.1 Noise component (period independent)

Period not returned yet / still some seasonality in this noise

```
In [343]: fig, ax = plt.subplots(figsize=(25, 20), nrows=4, ncols=1)
          for i, Xr in enumerate([(X_25S, reg_25), (X_40S, reg_40), (X_100S, reg_100), (X_41S, r
              seasonal = average(Xr[0]) * (Xr[0].cycle.max()+1)
              ax.flat[i].plot(Xr[0].f_ra, alpha=0.2)
              ax.flat[i].plot(seasonal + Xr[1], ':', alpha=0.8)
              ax.flat[i].plot(Xr[1], '--')
              ax.flat[i].plot(Xr[0].f_ra - seasonal, alpha=0.6)
```



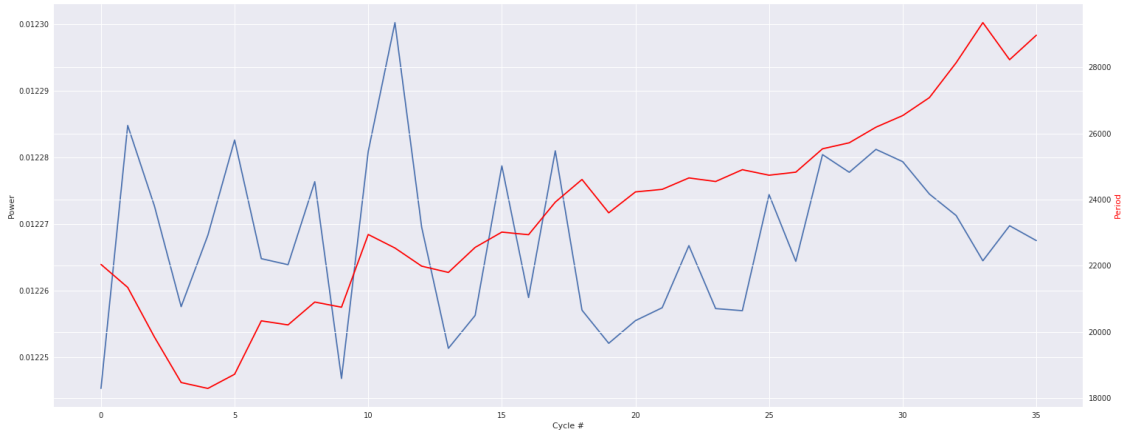
2.2 Noise component (period dependent)

```
In [ ]: reg_detrend('c', '')
```

3 PWM response.

```
In [322]: pwm = X_40S.pwm.iloc[::(X_40S[X_40S.cycle==1].shape[0])].values
          power(pwm, X_40S.rc.iloc[0], X_40S.rw.iloc[0], np.asarray(list(c_to_p40.values())))
```

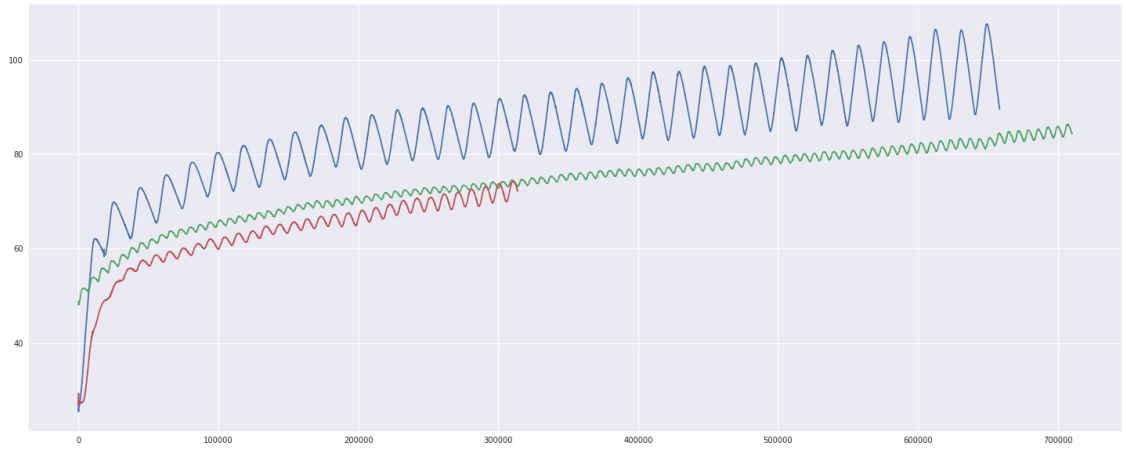
```
Out[322]: (<matplotlib.axes._subplots.AxesSubplot at 0x7fac6b3d10b8>,
           <matplotlib.axes._subplots.AxesSubplot at 0x7fac5d64cf60>,
           <matplotlib.figure.Figure at 0x7fac5d6719b0>)
```



3.1 Temperatures.

```
In [338]: plt.plot(X_40S.t)
          plt.plot(X_100S.t)
          plt.plot(X_41S.t)
```

```
Out[338]: [<matplotlib.lines.Line2D at 0x7fac5d22b0b8>]
```



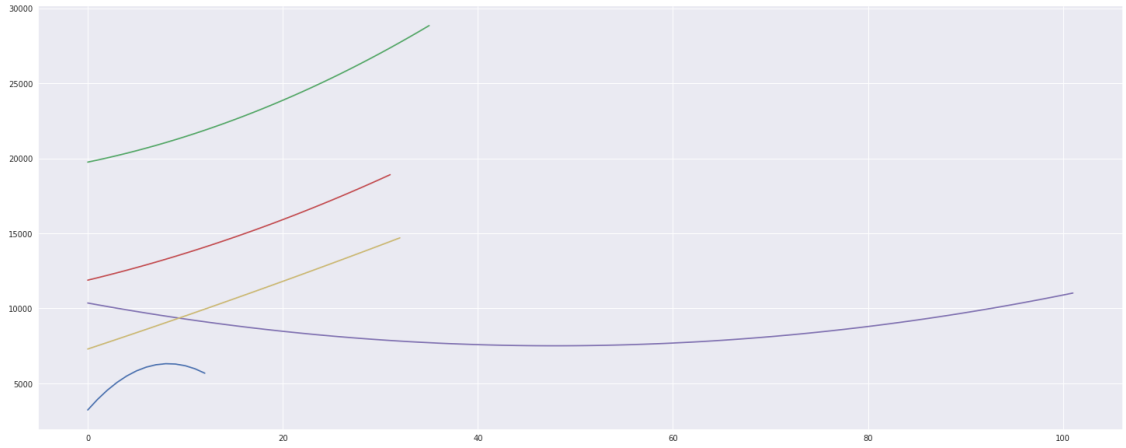
3.1.1 Covariance between PWM/temp/force

3.2 Force response given PWM, current (relative) temperature

First, fit periods.

```
In [364]: for i in range(5):
              if i in c_to_p42.keys():
                  del c_to_p42[i]
```

```
In [366]: period_trend = []
          for mapping in [c_to_p25, c_to_p40, c_to_p41, c_to_p100, c_to_p42]:
              X = pd.DataFrame(data=np.asarray(list(mapping.values())),
                               index=np.arange(len(mapping.keys()))
                              ).reset_index()
              X.columns = ['cycle', 'period']
              plt.plot(reg_detrend(X, x='cycle', y='period', order=2))
```



3.3 Cyclic temperature

and correlation with force.

Could pre-heating muscle increase longevity?

3.4 Muscle linearity** (viability as an actuator)

Compare power response (wrt rc, rw, pwm) of 36-c vs 103-c datasets (R+r not constant, PWM varies, everything else held constant)

4 Other questions

PWM response linearity?

Get response from PWM input