lab10

December 7, 2016

1 EE 379K Lab 10

1.1 Rohan Nagar and Wenyang Fu

1. Let $\{w_t\} \sim N(0,1)$ be a white noise process. Consider the autoregressive process (this is a generative model):

$y_t = -a_1 y_{t-1} - a_2 y_{t-2} + w_t.$

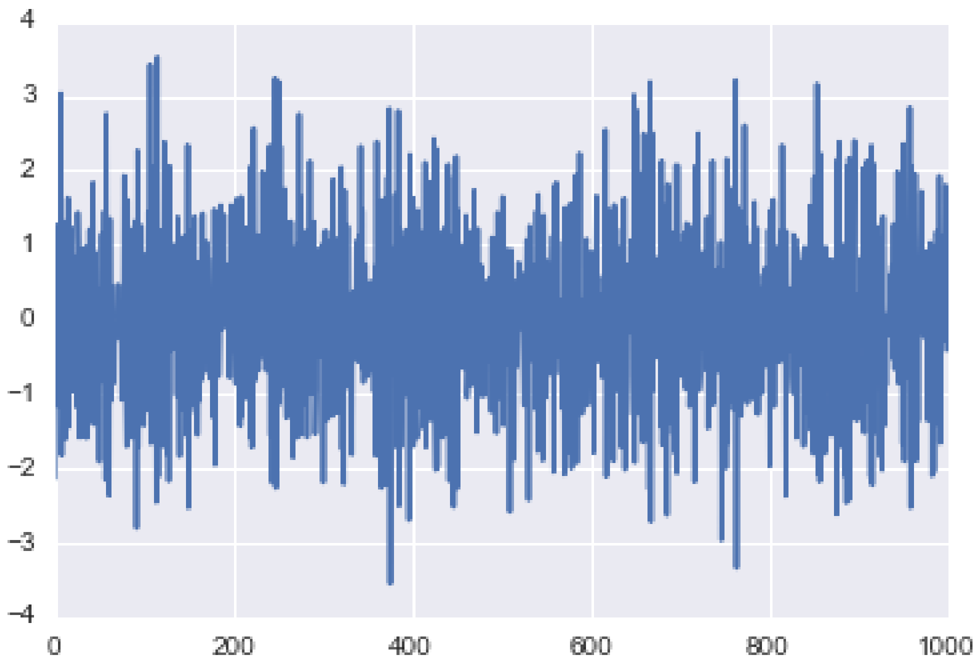
a. Let $a_1 = \frac{3}{4}$ and $z_2 = \frac{1}{8}$. Generate and plot trajectories for this time series. Do you think it is WSS?

```
In [4]: import numpy as np
        import matplotlib.pyplot as plt
        from functools import partial
        %matplotlib inline
```

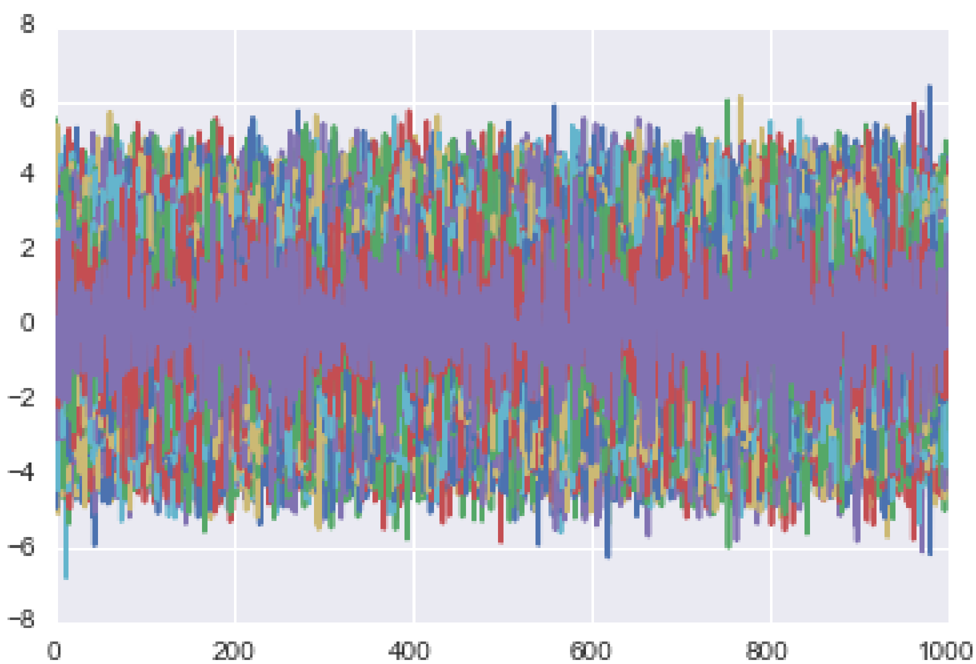
```
In [5]: def autoregression(mean, var, num_points, coef, init):
            points = []
            ys = init
            for _ in range(num_points):
                next_point = np.dot(coef, ys) + np.random.normal(0, 1)
                points.append(next_point)
                ys.pop()
                ys.insert(0, next_point)
            plt.plot(points)
            return points
```

```
In [6]: a1 = -3/4
        a2 = -1/8
        y_minus1 = 1
        y_minus2 = 0
        coef = [a1, a2]
        init = [y_minus1, y_minus2]
        mean, var = (0, 1)

        ar = partial(autoregression, mean, var, 1000, coef, init)
```

```
In [7]: l = ar()
```

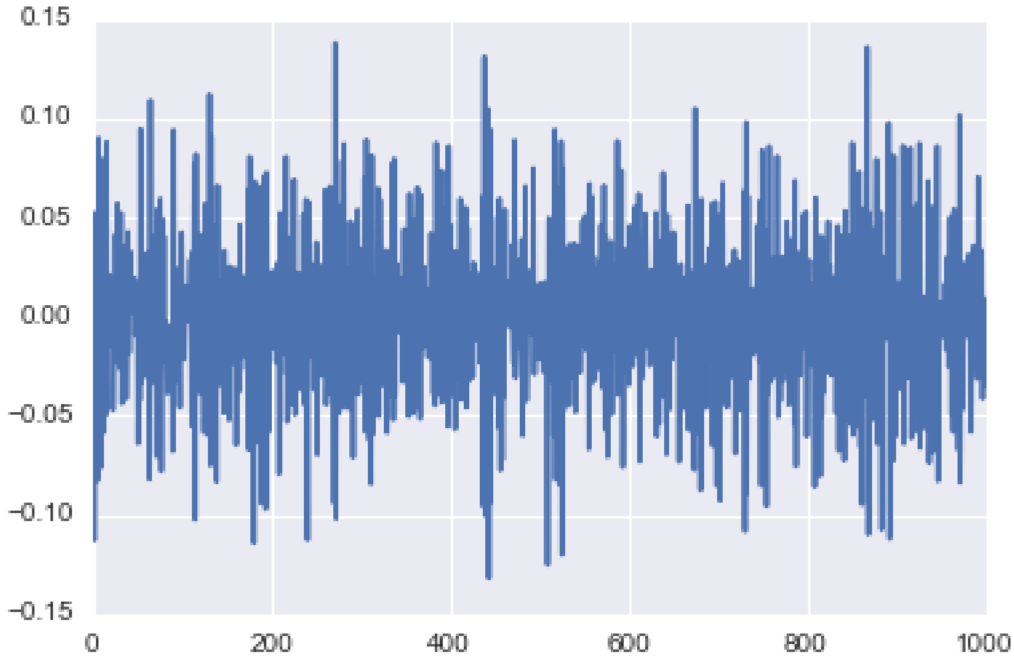



```
In [8]: # Plot 1000 trajectories with the same initial conditions
        trajectories = [ar() for _ in range(1000)]
```



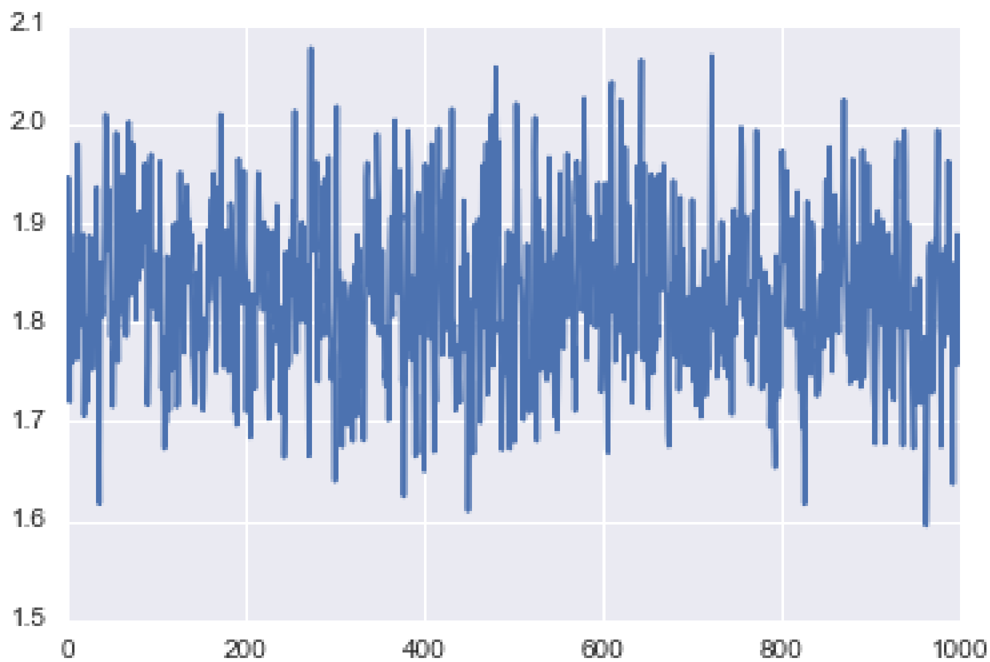

```
In [9]: plt.plot(np.array(trajectories).mean(axis=0))
```

Out[9]: [<matplotlib.lines.Line2D at 0x11faeb6a0>]



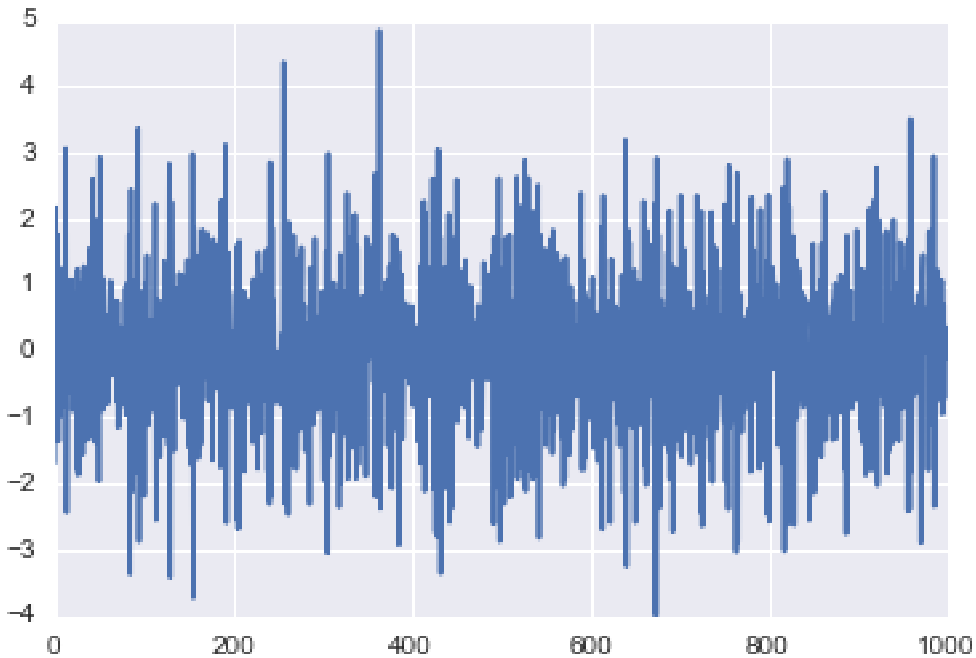

```
In [10]: plt.plot(np.array(trajectories).var(axis=0))
```

Out[10]: [<matplotlib.lines.Line2D at 0x11a5415c0>]



```
In [11]: a1 = -3/4
         a2 = -1/8
         y_minus1 = 1
         y_minus2 = 1
         coef = [a1, a2]
         init = [y_minus1, y_minus2]
         mean, var = (0, 1)
         ar = partial(autoregression, mean, var, 1000, coef, init)
```

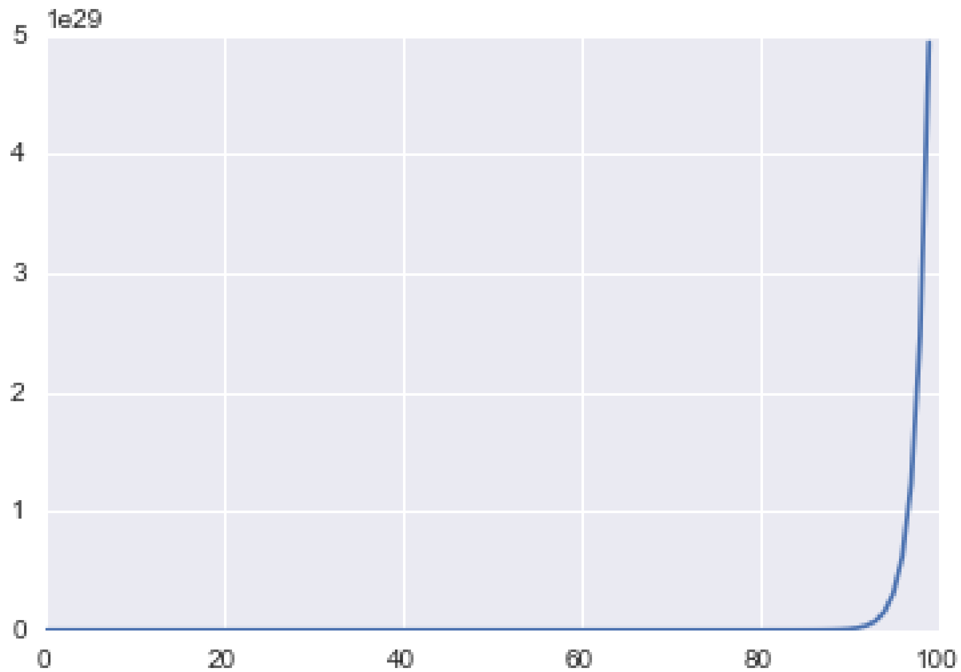
```
In [12]: l = ar()
```



Yes, this process looks like it is WSS, since the signal is consistent noise.
How about $a_1 = -2.25$ and $a_2 = 0.5$? Is this WSS?

```
In [13]: a1 = 9/4
         a2 = -1/2
         y_minus1 = 0.5
         y_minus2 = 1
         coef = [a1, a2]
         init = [y_minus1, y_minus2]
         mean, var = (0, 1)

         ar2 = partial(autoregression, mean, var, 100, coef, init)
```

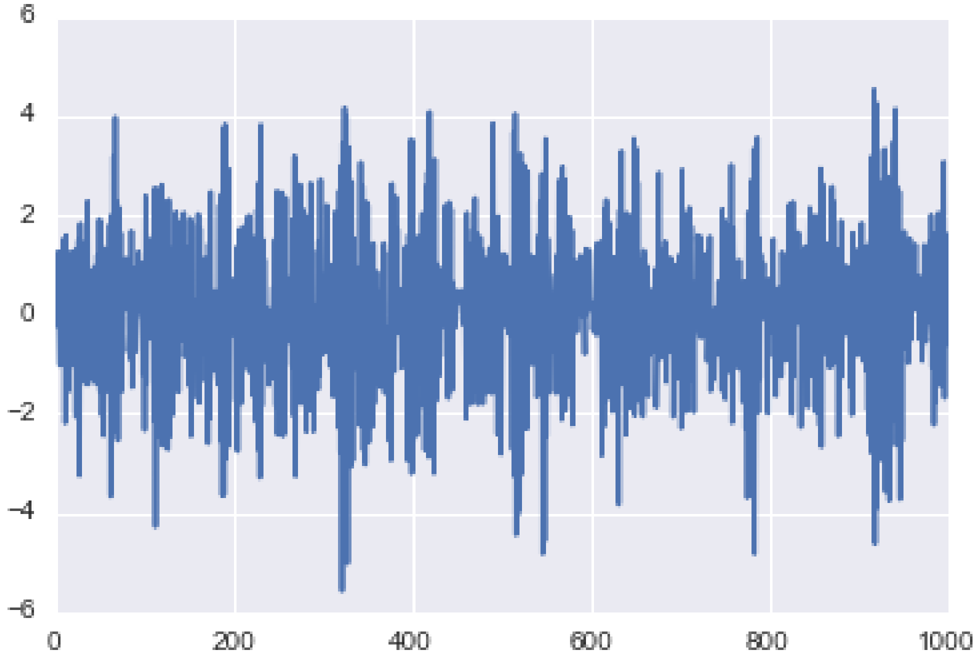
```
In [14]: l = ar2()
```



This process is not WSS, since the signal is diverging, and the mean and varaince are changing over tiem.
This is WSS:

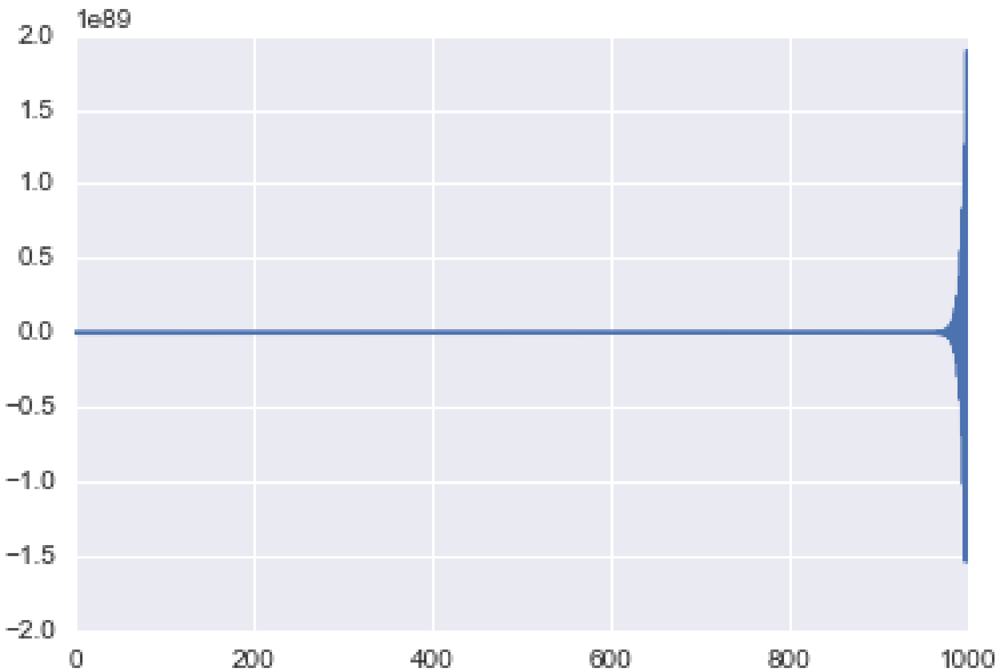
In [15]:
```
a1 = -0.5
a2 = 0.3
a3 = -0.05
a4 = 0.01
coef = [a1, a2, a3, a4]
init = [0, 0.5, 0.5, 1]
mean, var = (0, 1)

ar = partial(autoregression, mean, var, 1000, coef, init)
```

In [16]:
```
l = ar()
```



This is not WSS:

```
In [17]: a1 = -3/4
         a2 = -0.5
         a3 = -0.5
         a4 = -0.5
         coef = [-.75, .5, -.1, .01]
         init = [0, 0.5, 0.5, 1]
         mean, var = (0, 1)

         ar = partial(autoregression, mean, var, 1000, coef, init)

In [18]: l = ar()
```




```
In [19]: import pandas as pd
         import numpy as np
         import matplotlib as mpl
         import matplotlib.pyplot as plt

         %matplotlib inline
```

2 Problem 2

The output y_t of the AR process

$$y_t = -a_1 y_{t-1} - a_2 y_{t-2} - a_3 y_{t-3} - a_4 y_{t-4} + w_t$$

is the result of passing the input w_t through a linear time filter.
The z transform of the filter is the transfer function between the input and the output. This is equal to

$$H(z) = \frac{1}{AR(z)} = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}$$

We can also note that in order for $H(z)$ to be a stable filter, and therefore be Wide Sense Stationary, $AR(z)$ must have all of its roots inside the unit circle.

3 Problem 3

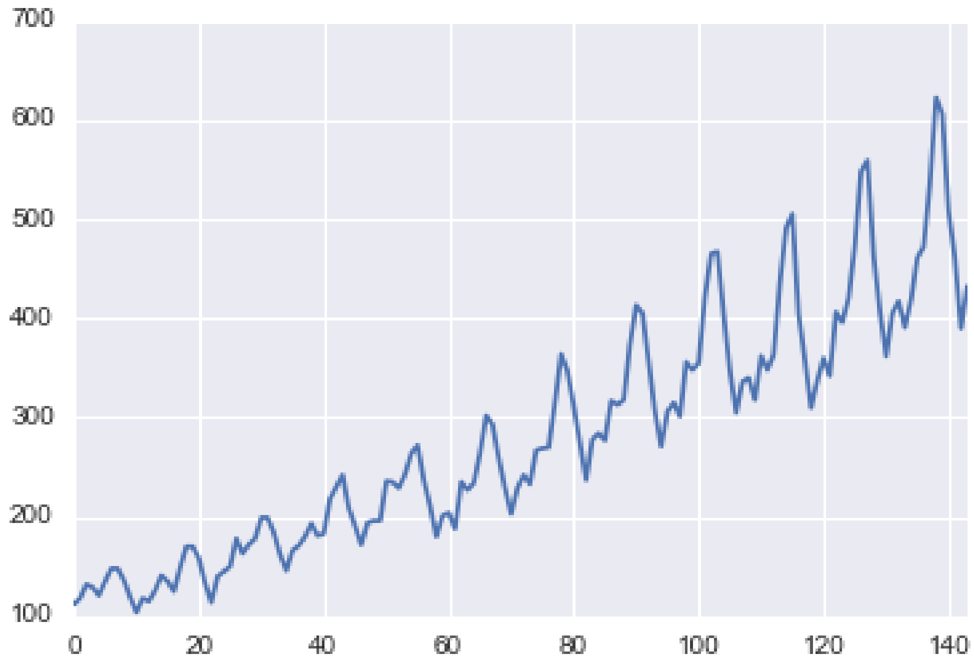
```
In [20]: airline_data = pd.read_csv('data/AirPassengers.csv', parse_dates=True)
         airline_data.head()
```

```
Out[20]:      Month  #Passengers
         0  1949-01          112
```

```
1   1949-02           118
2   1949-03           132
3   1949-04           129
4   1949-05           121
```

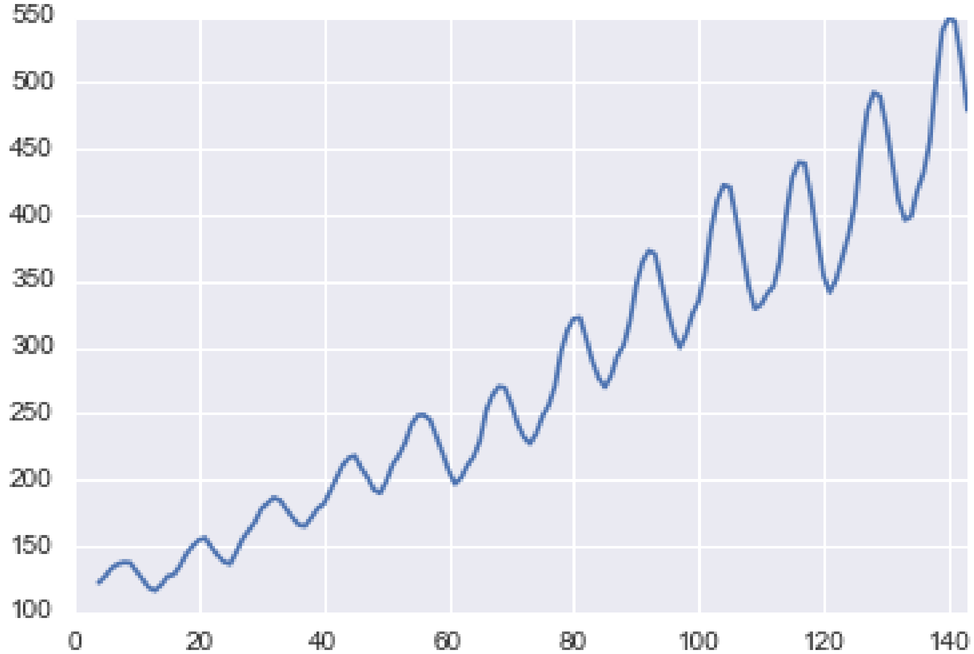
3.1 Plots

```
In [21]: airline_data['#Passengers'].plot()

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x11fbc07f0>
```



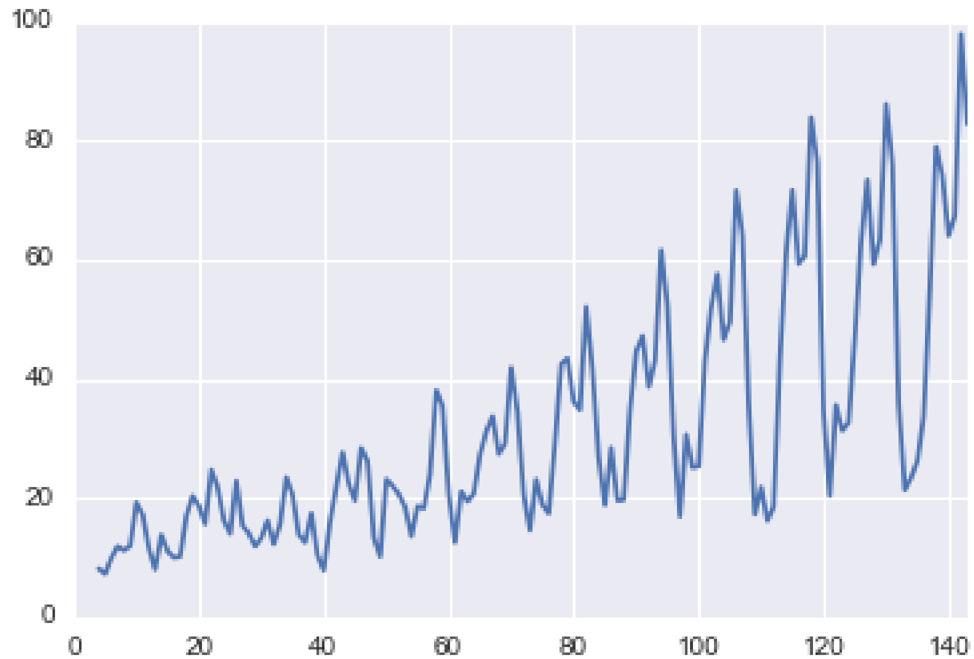
Rolling mean with window 5

```
In [22]: pd.rolling_mean(airline_data['#Passengers'], window=5).plot()

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1178024e0>
```



Rolling standard deviation with window 5

```
In [23]: pd.rolling_std(airline_data['#Passengers'], window=5).plot()

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x11aa248d0>
```

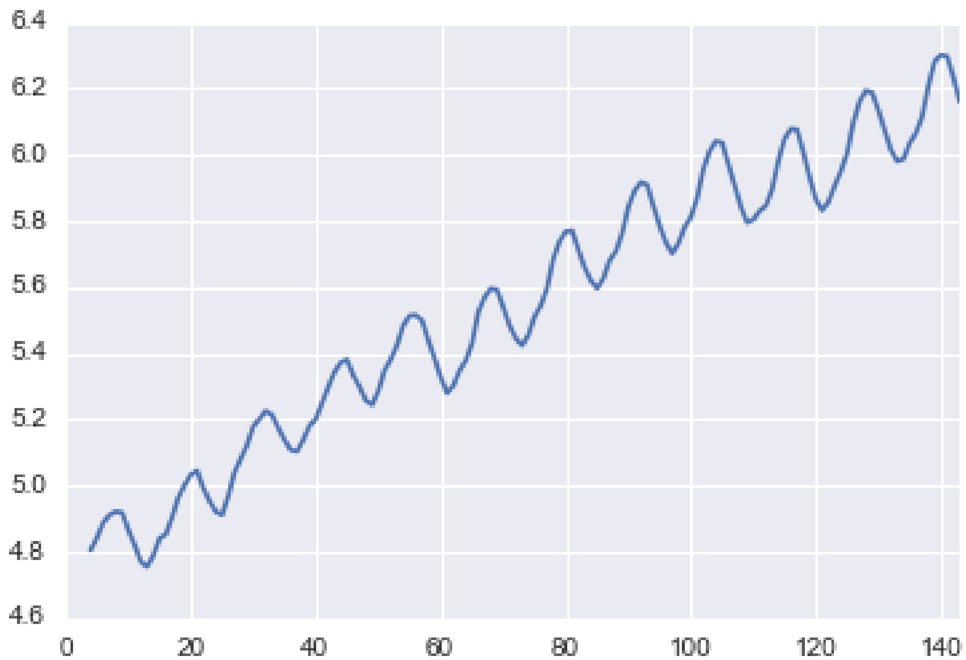


3.2 Transforms

Log Transform

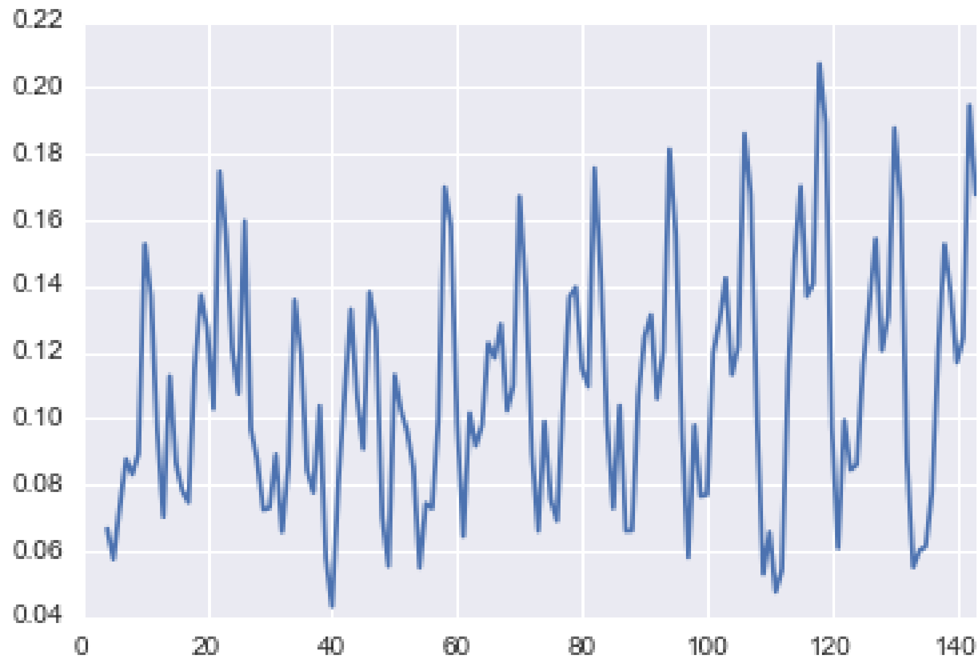
In [24]:
```
airline_data['#Passengers.log'] = np.log(airline_data['#Passengers'])
pd.rolling_mean(airline_data['#Passengers.log'], window=5).plot()
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x11e6dd5c0>



In [25]:
```
pd.rolling_std(airline_data['#Passengers.log'], window=5).plot()
```

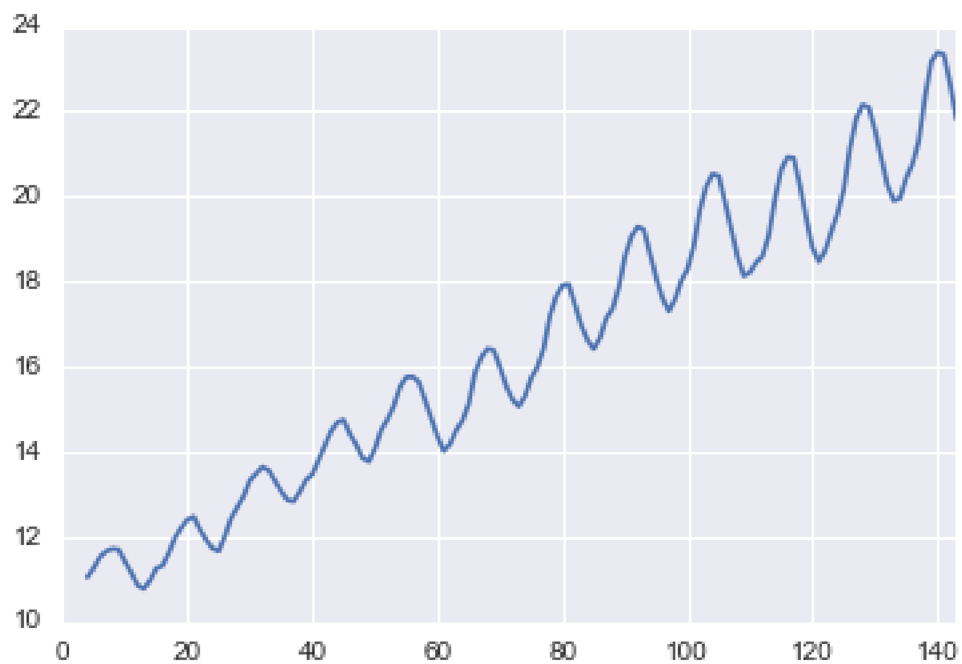
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x11ca2f0f0>



Sqrt transform

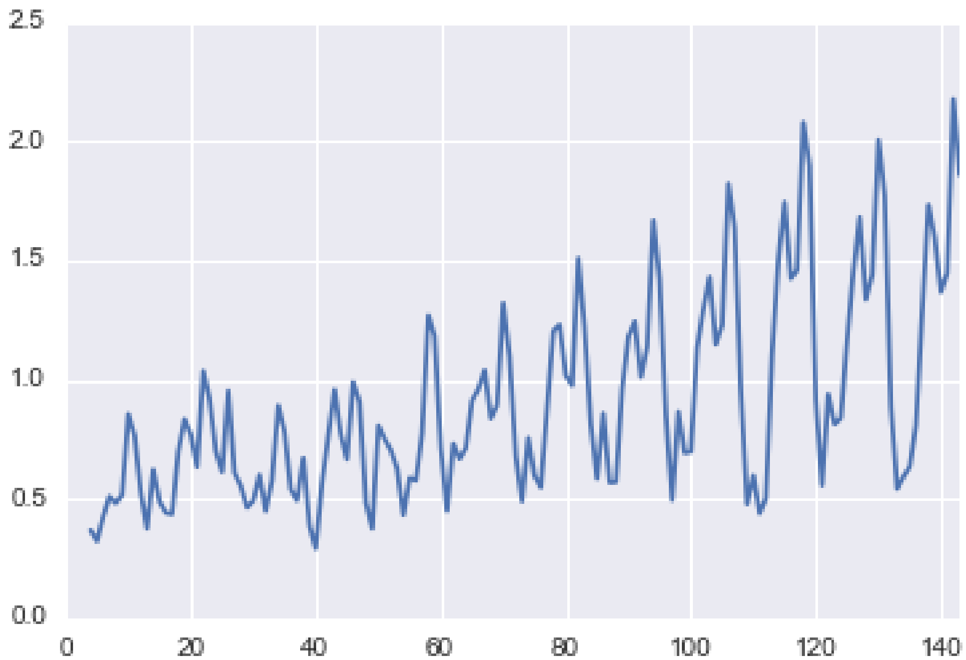
```
In [26]: airline_data['#Passengers.sqrt'] = np.sqrt(airline_data['#Passengers'])
         pd.rolling_mean(airline_data['#Passengers.sqrt'], window=5).plot()
```

```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x11dcb11d0>
```



```
In [27]: pd.rolling_std(airline_data['#Passengers.sqrt'], window=5).plot()
```

```
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x11ac07668>
```



3.2.1 Question

Which of these is better?

3.2.2 Answer

The log transform is better, since the rolling variance of the log transform is more stationary relative to the rolling variance of the sqrt transform.

3.3 Dickey-Fuller Test

```
In [28]: from statsmodels.tsa.stattools import adfuller

         adfuller(airline_data['#Passengers'])
```

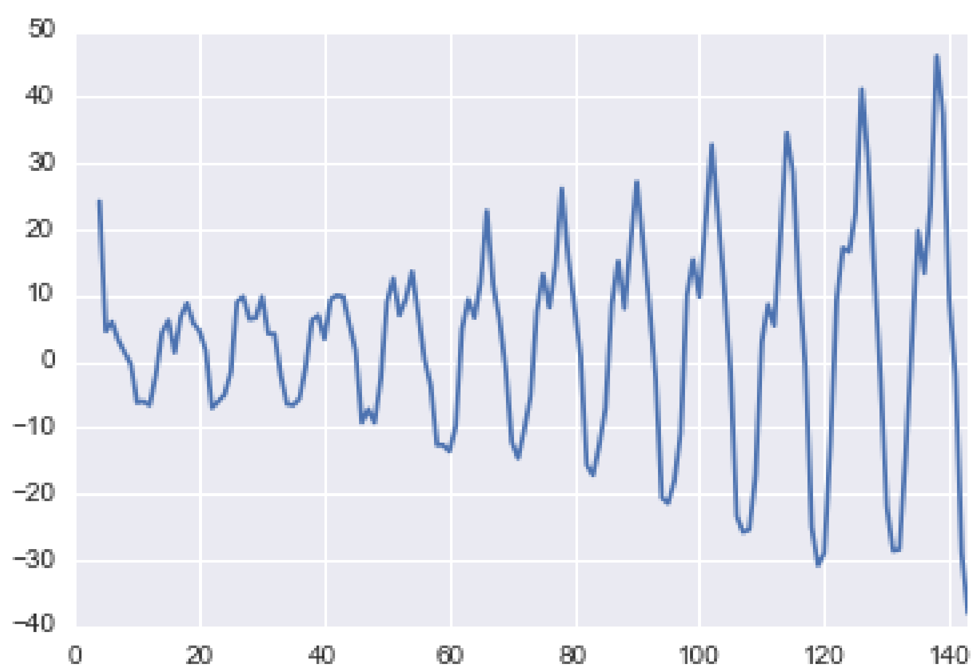
```
Out[28]: (0.81536887920605083,
          0.99188024343764103,
          13,
          130,
          {'1%': -3.4816817173418295,
           '10%': -2.5787700591715979,
           '5%': -2.8840418343195267},
          996.69293083901891)
```

3.4 Remove Mean with Transform of Differences

```
In [29]: def difference_transform(series):
             new_series = []
             for i in range(len(series)):
                 if i == 0:
                     new_series.append(series[i])
                 else:
                     new_series.append(series[i] - series[i-1])

             return new_series
```

```
In [30]: airline_data['#Passengers.diff'] = difference_transform(airline_data['#Passengers'])
         pd.rolling_mean(airline_data['#Passengers.diff'], window=5).plot()

         adfuller(airline_data['#Passengers.diff'])
```

```
Out[30]: (-3.1551124072891468,
          0.022734464303624552,
          12,
          131,
          {'1%': -3.481281802271349,
           '10%': -2.5786771965503177,
           '5%': -2.8838678916645279},
          995.40139015094792)
```



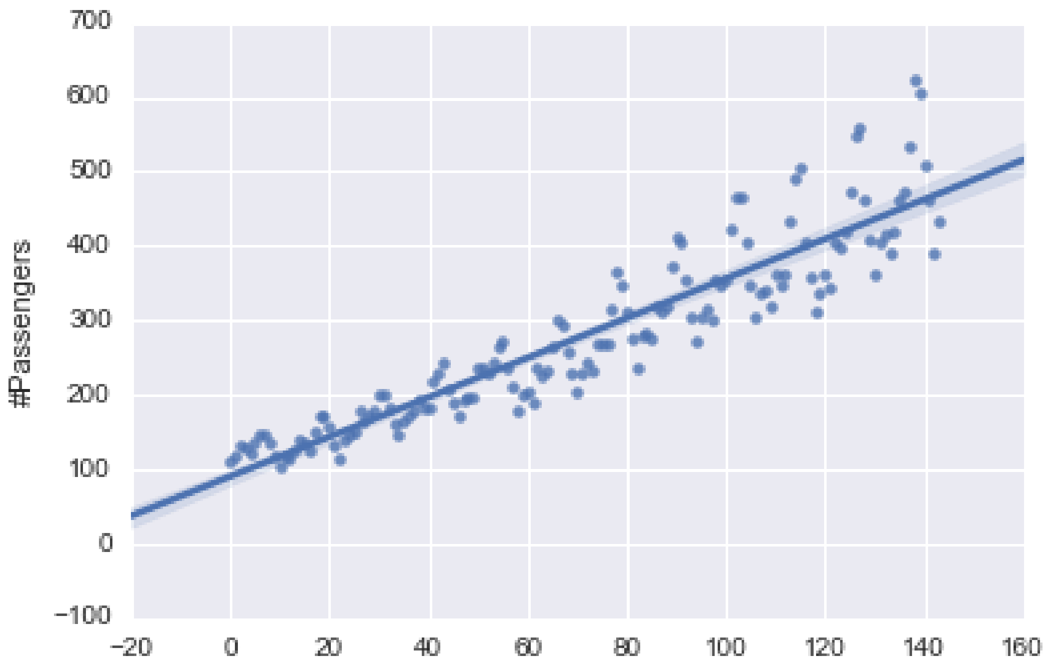
3.5 Regress Against Linear Function

```
In [31]: from sklearn.linear_model import LinearRegression
```

```
    # need to do linear regression with time steps (0, 1, 2, ..) as the X and the airline_data['#P
    regr = LinearRegression()
```

```
In [32]: import seaborn as sns

         x_timesteps = np.fromiter(range(len(airline_data['Month'])), dtype=np.int32).reshape(-1, 1)

         sns.regplot(x_timesteps, '#Passengers', airline_data)
```

```
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1177dc0b8>
```

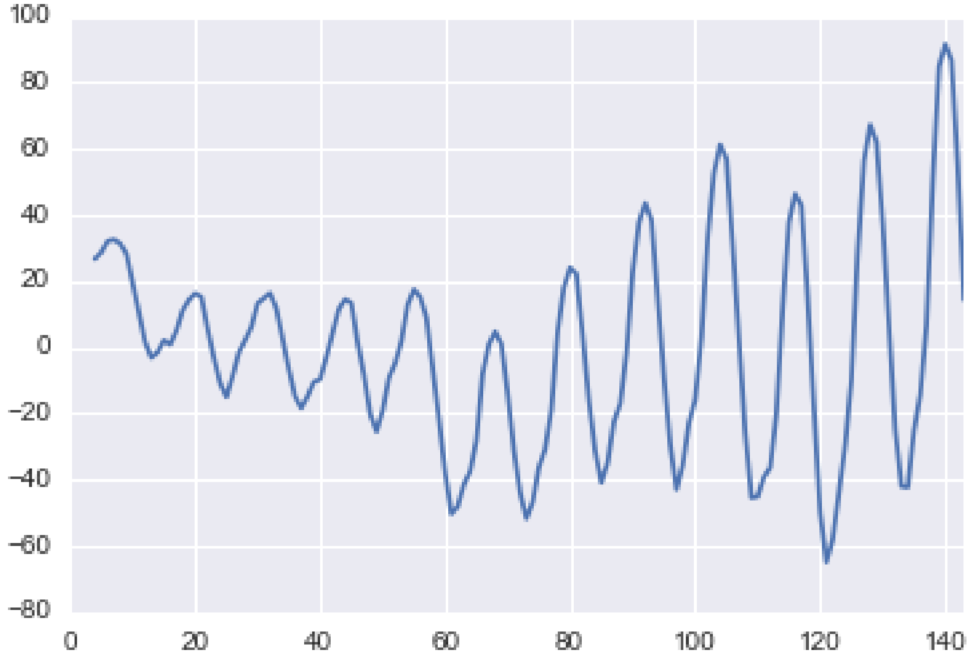


3.6 Remove Mean Using Linear Regression

Compared to using a transformation of differences, removing the mean using linear regression makes a more stationary-looking process.

```
In [33]: regr.fit(x_timesteps, airline_data['#Passengers'])
         mean_preds = regr.predict(x_timesteps)
         airline_data['#Passengers.linreg_diff'] = airline_data['#Passengers'] - mean_preds

         pd.rolling_mean(airline_data['#Passengers.linreg_diff'], window=5).plot()
         adfuller(airline_data['#Passengers.linreg_diff'])
```

```
Out[33]: (-2.1019659058880054,
          0.24372483602015288,
          13,
          130,
          {'1%': -3.4816817173418295,
           '10%': -2.5787700591715979,
           '5%': -2.8840418343195267},
          992.62934767868626)
```

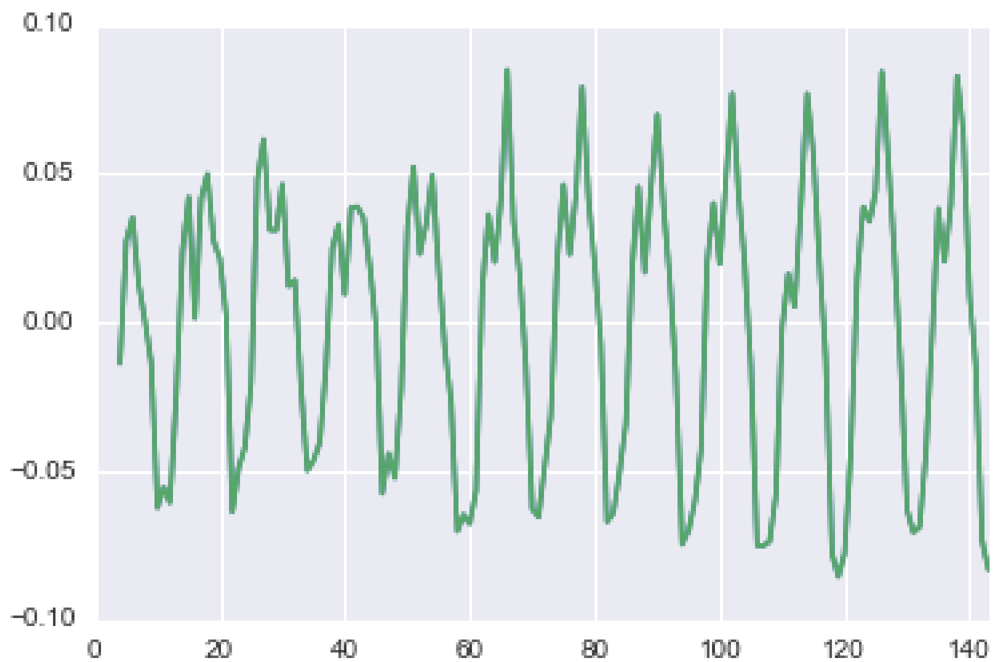


3.7 Remove the Seasonality

Using differences with a periodicity of d, we can find the best rolling difference by visually inspecting the mean and the variance, and choosing the variance that stays the most stationary, and corroborating that information with results from the Dickey-Fuller test. With these results, I choose a value of $d = 6$.

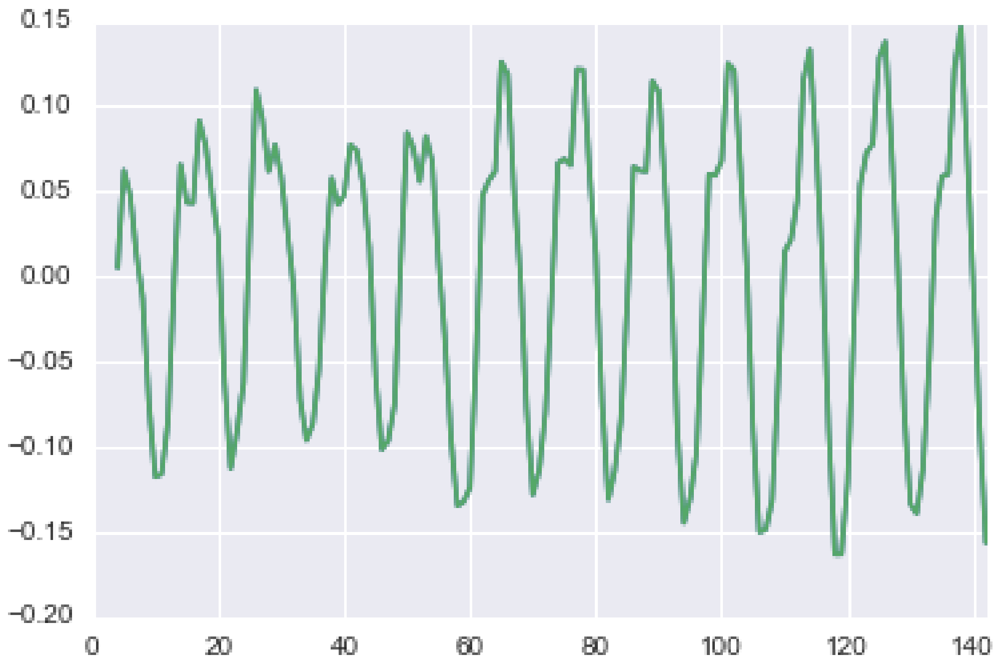
```
In [34]: def seasonality_transform(series, d):
             new_series = []
             for i in range(len(series)):
                 if i == 0:
                     new_series.append(series[i])
                 else:
                     if i - d < 0:
                         continue
                     new_series.append(series[i] - series[i-d])

             return pd.Series(new_series)

In [35]: from functools import partial

         regr = LinearRegression()
         regr.fit(x_timesteps, airline_data['#Passengers.log'])
         airline_data['#Passengers.log_linreg'] = airline_data['#Passengers.log'] - regr.predict(x_times
         d_diff = partial(seasonality_transform, airline_data['#Passengers.log_linreg'])

In [36]: seasonal_diff = [d_diff(d) for d in range(1, 10)]  # Data adjusted with log transform, linear 

In [39]: pd.rolling_mean(seasonal_diff[0], window=5).plot()
         pd.rolling_mean(seasonal_diff[0], window=5).plot()
         adfuller(seasonal_diff[0])
```

Out [39]: (-2.7225552823572818,
 0.070215882112042119,
 14,
 129,
 {'1%': -3.4820879640460261,
  '10%': -2.5788643813472749,
  '5%': -2.8842185101614626},
 -444.97455856001989)



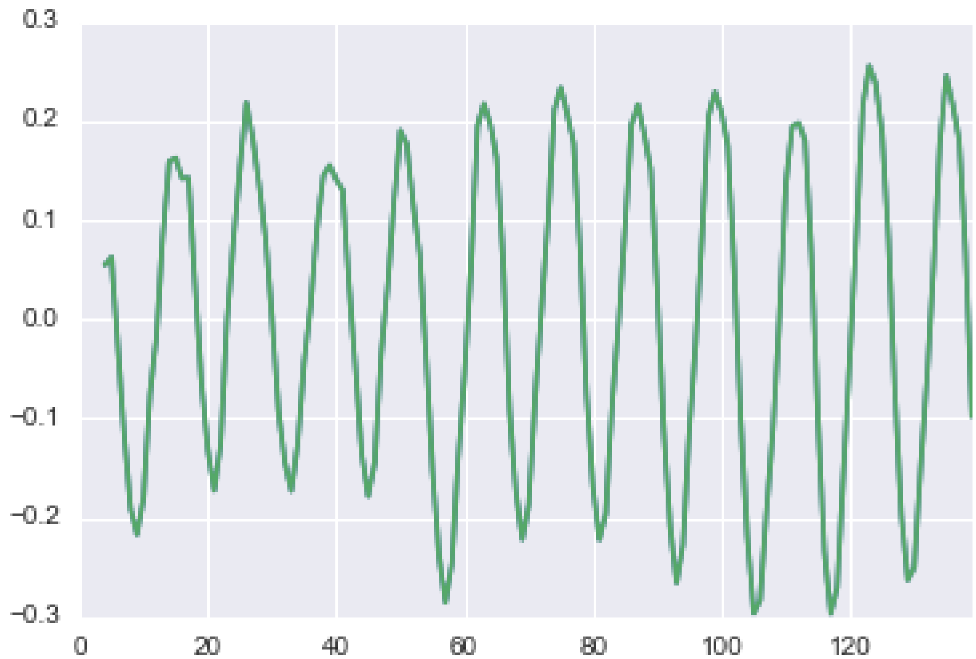
In [40]:
```
pd.rolling_mean(seasonal_diff[1], window=5).plot()
pd.rolling_mean(seasonal_diff[1], window=5).plot()
adfuller(seasonal_diff[1])
```

Out [40]: (-3.5261271516976938,
 0.0073409062272698162,
 11,
 131,
 {'1%': -3.481281802271349,
  '10%': -2.5786771965503177,
  '5%': -2.8838678916645279},
 -435.09320491172321)



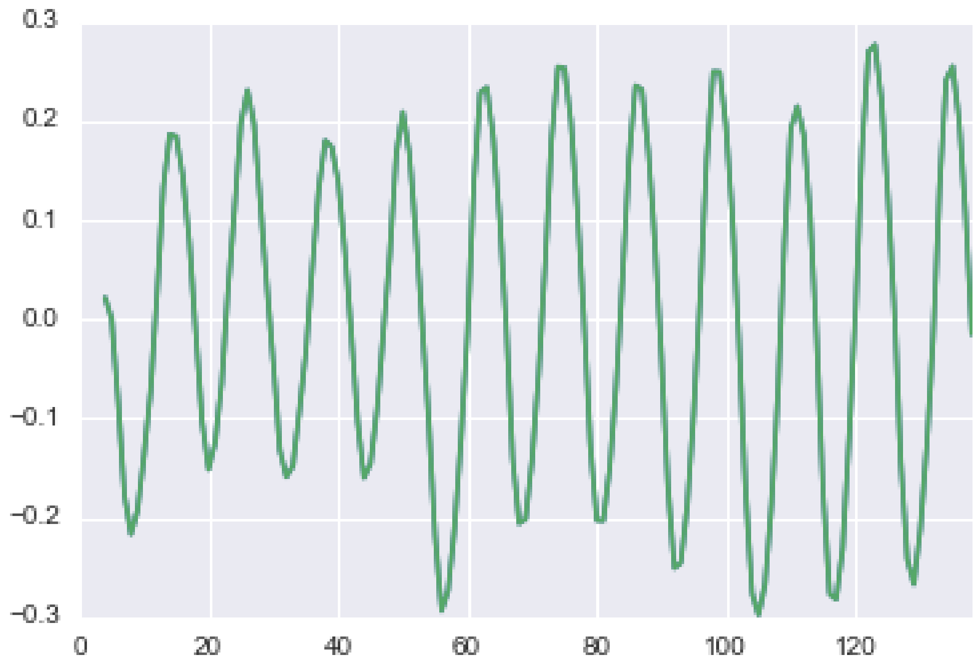

```
In [41]: pd.rolling_mean(seasonal_diff[4], window=5).plot()
         pd.rolling_mean(seasonal_diff[4], window=5).plot()
         adfuller(seasonal_diff[4])
```

```
Out[41]: (-2.7003390446143229,
          0.07397961788776139,
          14,
          125,
          {'1%': -3.4837793736959997, '10%': -2.5792569759999999, '5%': -2.88495387648},
          -384.57258005248559)
```



In [42]:
```
pd.rolling_mean(seasonal_diff[5], window=5).plot()
pd.rolling_mean(seasonal_diff[5], window=5).plot()
adfuller(seasonal_diff[5])
```

Out[42]:
```
(-3.8239341764567287,
 0.0026717136582462293,
 7,
 131,
 {'1%': -3.481281802271349,
  '10%': -2.5786771965503177,
  '5%': -2.8838678916645279},
 -426.33892741247496)
```



3.8 Regress Against Sinusoidal

In [ ]:

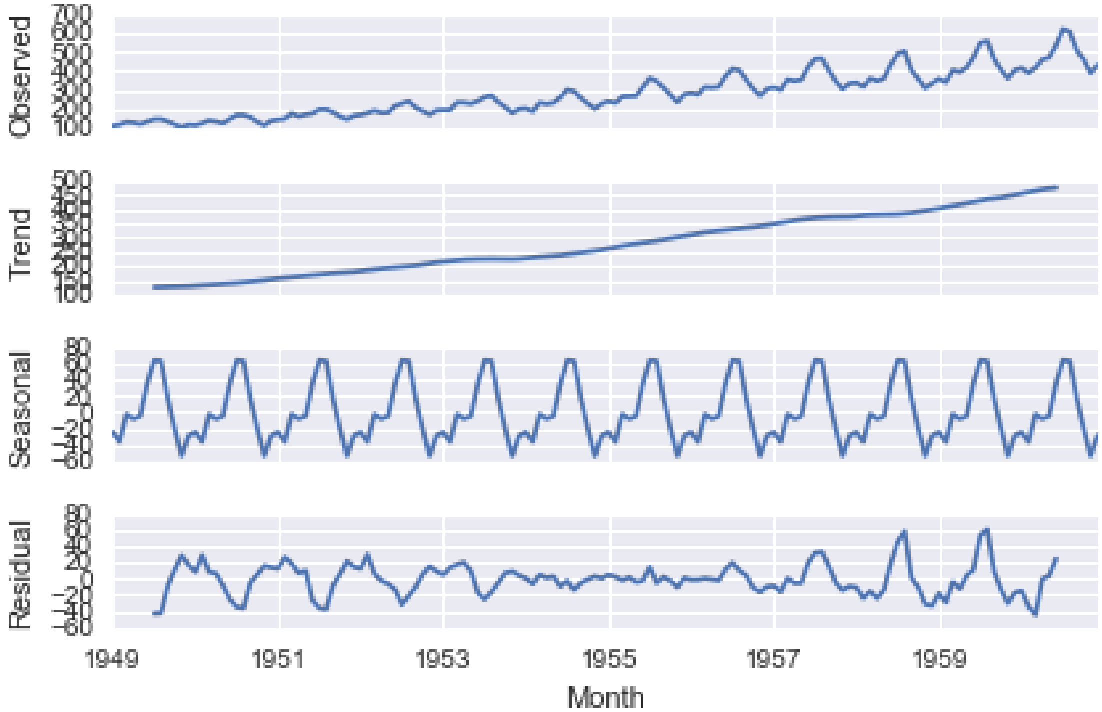
The one that gave us the best score on the Dickey-Fuller test was removing the seasonality using differences.

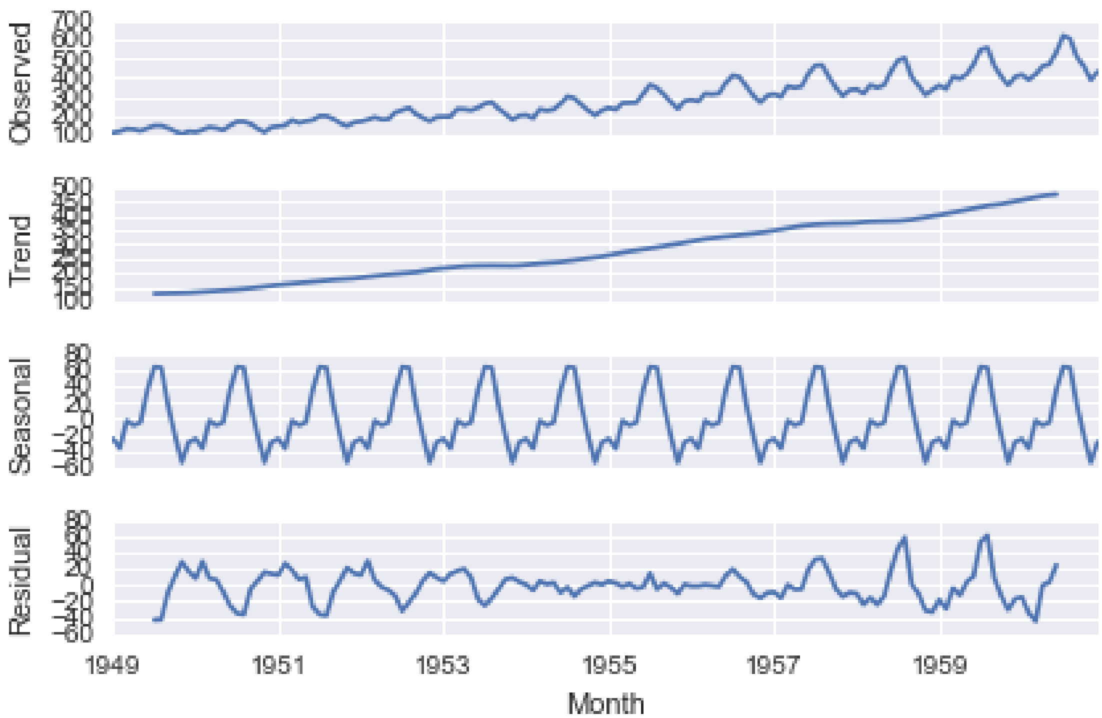
3.9 Seasonal Decompose

```
In [84]: from statsmodels.tsa.seasonal import seasonal_decompose

         indexed_airline_data = pd.read_csv('data/AirPassengers.csv', parse_dates=True, index_col=0)
         decomp = seasonal_decompose(indexed_airline_data)
         decomp.plot()
```

```
/Users/rohannagar/anaconda/lib/python3.5/site-packages/statsmodels/tsa/filters/filtertools.py:28: Visibl
  return np.r_[[np.nan] * head, x, [np.nan] * tail]
```

Out[84]:


Observed
Trend
Seasonal
Residual
1949
1951
1953
1955
1957
1959
Month


Observed
Trend
Seasonal
Residual
1949
1951
1953
1955
1957
1959
Month

```
In [68]: residual = indexed_airline_data - decomp.seasonal - decomp.trend
         residual.head(10)
```

```
Out[68]:             #Passengers
         Month
         1949-01-01          NaN
         1949-02-01          NaN
         1949-03-01          NaN
         1949-04-01          NaN
         1949-05-01          NaN
         1949-06-01          NaN
         1949-07-01   -42.622475
         1949-08-01   -42.073232
         1949-09-01    -8.478535
         1949-10-01    11.059343
```

4 Problem 4

4.0.1 Quebec Data

```
In [3]: quebec_data = pd.read_csv('data/QuebecCarsales.csv')
        quebec_data.head()
```

```
Out[3]:      Month  Monthly car sales in Quebec 1960-1968
        0  1960-01                                   6550
        1  1960-02                                   8728
        2  1960-03                                  12026
        3  1960-04                                  14395
        4  1960-05                                  14587
```

```
In [ ]:
```

4.0.2 Dow Jones Data

```
In [4]: dow_data = pd.read_csv('data/DowJones.csv')
        dow_data.head()
```

```
Out[4]:       Week  \
        0  1971-W27
        1  1971-W28
        2  1971-W29
        3  1971-W30
        4  1971-W31

          Weekly closings of the Dow-Jones industrial average, July 1971 ? August 1974
        0                                             890.19
        1                                             901.80
        2                                             888.51
        3                                             887.78
        4                                             858.43
```

```
In [ ]:
```