hw3

October 3, 2018

1 Meteo 515 – Assignment 3 – Linear Regression

In [1]:
```
from __future__ import division, print_function
#from collections import OrderedDict
from itertools import chain
#import datetime as dt

#import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.optimize as so
import scipy.stats as ss
#import sklearn as skl
import statsmodels.api as sm
```

In [2]:
```
plt.style.use('seaborn-darkgrid')
%matplotlib notebook
```

1.1 Load the data

We are using this dataset, the not-detrended and unsmoothed AMO index from NOAA ESRL. More info here: https://esrl.noaa.gov/psd/data/timeseries/AMO/

Note: Currently (27-Sep-18 10:00 EST) the data on the site is messed up for 1980 onwards, though it was fine when I downloaded it the day before.

In [3]:
```
amo_fpath = './data/amon.us.long.mean.data'

amo_raw = np.genfromtxt(amo_fpath, skip_header=1, skip_footer=7)[:,1:]

with open(amo_fpath, 'r') as f: yr_range = f.readline().split()
t_amo = pd.date_range(start='{}/01/01'.format(yr_range[0]), freq='MS', periods=amo_raw
amo_ndt_us = amo_raw.reshape((amo_raw.size,))

amo = pd.DataFrame({'amo': amo_ndt_us}, index=t_amo)
amo['julian_date'] = amo.index.to_julian_date()             # Julian Date (decimal d
amo['t_elapsed'] = amo.julian_date - amo.julian_date.iloc[0]  # elapsed time (decimal
amo['year'] = amo.index.year                                # integer year for each
```

```
amo['decyear'] = amo.index.year + (amo.index.month-1)/12      # decimal year

amo[amo == -99.99] = np.nan
amo.dropna(inplace=True)

#> don't include 2018 in the annual means, since 4 months are missing
grouped = amo.loc[amo.year<=2017, :].groupby(pd.Grouper(freq='A'))
amo_annual_mean = grouped.mean()
amo_annual_mean['sem_annual'] = grouped.amo.std().values / np.sqrt(12)  # stdev of the
# ^ note that this gives indices that are the last day of the year
#   and {year}.46 for decimal year, since initially the datetimes are first day of the
```

In [4]:
```
#> plotting functions!
figsize_lin_reg = (9, 4.0)
figsize_res = (8, 3)
#degC = u'\u00B0C'
degC = u' (\u00B0C)'

def res_plot(y, y_hat, df_res=2, figid='ts_res'):
    """Plot the residuals
    could pass the OLS result in, but not doing that currently...
    """
    f, a = plt.subplots(figsize=figsize_res, num=figid)

    y_bar = y.mean()
    SSE = np.sum((y-y_hat)**2)  # sum of squared error
    SSR = np.sum((y_hat-y_bar)**2)  # residual sum of squares
    SST = np.sum((y-y_bar)**2)
    assert( np.isclose(SST, SSE+SSR) )
    #MSE = SSE/(res.df_resid)  # MSE == sample variance of the residuals
    MSE = SSE/df_res
    #assert( np.isclose(MSE, res.mse_resid) )  # res.mse_model == MSR; F = MSR/MSE
    s = '''
    SSE = {:.3g}
    SSR = {:.3g}
    MSE = {:.4g}
    '''.format(SSE, SSR, MSE)

    a.plot(t_plot, y-y_hat, '.-', c='purple', ms=6, lw=1, label=s)
    #a.set_ylabel('residual AMO index {:s}'.format(degC))
    a.set_ylabel('residual AMO index' + degC)
    a.text(1.01, 0.5, s,
           va='center', ha='left', transform=a.transAxes)
    #a.legend(loc='center left', bbox_to_anchor=(0.99, 0.5), labelspacing=1.5)

    f.tight_layout(rect=(0, 0, 0.85, 1.0))

    return f
```

```
def lin_reg_plot(t_plot, y, y_hat, s_model, figid='ts'):
    """Plot the data and least-squares regression result"""
    f, a = plt.subplots(figsize=figsize_lin_reg, num=figid)

    #xbar, se = y, y.std()
    #a.fill_between(amo_annual_mean.index, xbar-1.*se, xbar+1.*se, alpha=0.3, label='S
    a.plot(t_plot, y, 'b.-', alpha=0.6, ms=6, lw=1, label='annual means')
    a.plot(t_plot, y_hat, '-', c='r', lw=2, label=s)

    #a.text(0.02, 0.98, s,
    #       va='top', ha='left', transform=a.transAxes)

    a.set_ylabel('AMO index' + degC)
    a.legend(loc='center left', bbox_to_anchor=(0.99, 0.5), labelspacing=1.5)

    f.tight_layout()

    return f
```

1.2 1) Simple linear regression with year as predictor

Note that using the monthly data vs the annual means gives slightly different answers, mainly for the error. Change `df_reg` to see.

```
In [5]: df_reg = amo_annual_mean  # {amo_annual_mean, amo}
        t_reg = df_reg.year
        t_plot = df_reg.index

        res = sm.OLS(df_reg['amo'], sm.add_constant(t_reg), ).fit()

        print(res.summary())  # note: can plot res using `sm.graphics.abline_plot(model_result

        param_lines = '\n'.join([r'$\beta_{:d} = {{:.3g}} \ ({{:.3g}})$'.format(i) for i in ran
        s = r'$y=\beta_1 t + \beta_0$' + '\n\n' + param_lines
        s = s.format(*chain.from_iterable(zip(res.params, res.bse)))

        y = df_reg['amo'].values
        y_hat = t_reg*res.params[1]+res.params[0]

        f1 = lin_reg_plot(t_plot, y, y_hat, s, 'ts_ols1')

        f1b = res_plot(y, y_hat, df_res=res.df_resid, figid='ts_ols1_res');
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    amo   R-squared:                       0.340
```

```
Model:                            OLS   Adj. R-squared:                  0.336
Method:                 Least Squares   F-statistic:                     82.56
Date:                Wed, 03 Oct 2018   Prob (F-statistic):           3.73e-16
Time:                        21:33:48   Log-Likelihood:                 47.402
No. Observations:                 162   AIC:                            -90.80
Df Residuals:                     160   BIC:                            -84.63
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         15.3469      0.591     25.952      0.000      14.179      16.515
year           0.0028      0.000      9.086      0.000       0.002       0.003
==============================================================================
Omnibus:                        3.980   Durbin-Watson:                   0.612
Prob(Omnibus):                  0.137   Jarque-Bera (JB):                2.915
Skew:                          -0.182   Prob(JB):                        0.233
Kurtosis:                       2.454   Cond. No.                     8.02e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.02e+04. This might indicate that there are
strong multicollinearity or other numerical problems.


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

1.2.1 Discussion:

Our R^2 here is pretty low. The value (0.34) indicates that our model explains only 34% of the variation in AMO index. In the plot of the residuals, we see a clear oscillatory pattern, which next we add to our model to see how that improves things...

1.3 2) Adding a prescribed oscillation

In [6]:
```
phase = -21   # years
period = 65   # in years
```

```python
y = df_reg['amo']  # AMO index

x0 = np.ones(df_reg['amo'].shape)  # intercept
x1 = df_reg['year'].values  # year
x2 = np.sin(2*np.pi/period*(x1-phase))  # oscillation with prescribed period and phase

#X = np.vstack((x2, x1, x0)).T
X = pd.DataFrame(data={'0-const': x0, '1-year': x1, '2-osc_amp': x2}, index=df_reg.inde

res = sm.OLS(y, X).fit()

y_hat = np.dot(X, res.params)

print(res.summary())

param_lines = '\n'.join([r'$\beta_{:d} = {{:.3g}} \ ({{:.3g}})$'.format(i) for i in ran
s = r'''
$y = \beta_2 \sin\left( \frac{{2 \pi}}{{\mathrm{{period}}}} (t - \mathrm{{phase}}) \rig
      $+ \beta_1 t + \beta_0$

period = {:d}
phase = {:d}
''' + param_lines
s = s.format(period, phase, *chain.from_iterable(zip(res.params, res.bse)))

f2 = lin_reg_plot(t_plot, y, y_hat, s, 'ts_ols2')

f2b = res_plot(y, y_hat, df_res=res.df_resid, figid='ts_ols2_res');
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    amo   R-squared:                       0.655
Model:                            OLS   Adj. R-squared:                  0.651
Method:                 Least Squares   F-statistic:                     151.2
Date:                Wed, 03 Oct 2018   Prob (F-statistic):           1.65e-37
Time:                        21:33:48   Log-Likelihood:                 99.994
No. Observations:                 162   AIC:                            -194.0
Df Residuals:                     159   BIC:                            -184.7
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
0-const       15.4580      0.429     36.044      0.000      14.611      16.305
1-year         0.0027      0.000     12.234      0.000       0.002       0.003
2-osc_amp      0.1779      0.015     12.056      0.000       0.149       0.207
==============================================================================
Omnibus:                        5.696   Durbin-Watson:                   1.167
```

```
Prob(Omnibus):                  0.058   Jarque-Bera (JB):                3.405
Skew:                           0.153   Prob(JB):                        0.182
Kurtosis:                       2.359   Cond. No.                     8.03e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.03e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

1.3.1 Discussion:

Adding a prescribed oscillation to our model has greatly improved the fit: our R^2 has increased by about a factor of 2, and MSE correspondingly decreased by about the same. The plot of the residuals is now more like what we would like to see (looking just noisy). However, the values in the first few decades seem to be noticeably and consistently a bit higher than later years, suggesting that maybe the increasing trend didn't start until sometime after 1857. In (3) we will add that to the model and see how things improve...

But first:

1.4 2.5) Find a better oscillation using optimization

```
In [7]: def fit1(t, slope, const, amp, period, phase):
            return amp*np.sin(2*np.pi/period*(t-phase)) + slope*t + const

        popt, pcov = so.curve_fit(fit1, df_reg.year, df_reg.amo,
                                  p0=(0.0027, 15.5, 0.18, 65, 75))

        popt[-1] = popt[-1] #% popt[-2]

        y = df_reg.amo
        y_hat = fit1(df_reg.year, *popt)

        #print(res.summary())

        s = r'''
```

```
        $y = \beta_2 \sin\left( \frac{{2 \pi}}{{\mathrm{{period}}}} (t - \mathrm{{phase}}) \rig
              $+ \beta_1 t + \beta_0$

        $\beta_1 = {:.4g}$
        $\beta_0 = {:.4g}$
        $\beta_2 = {:.4g}$
        period = {:.4g}
        phase = {:.4g}
        '''.format(*popt)

        f2p5 = lin_reg_plot(t_plot, y, y_hat, s, 'ts_ols2.5')

        n = df_reg.index.size
        f2p5b = res_plot(y, y_hat, df_res=n-5, figid='ts_ols2.5_res');
```

```
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

1.4.1 Discussion:

The improvement in fit is more modest this time, and in fact pretty similar to what we will see below in (3).

1.5 3) Same as (2), but linear trend predictor starts in 1900

Also, we have the option of using the period and phase from the optimized fit in (2.5), but this turned out to not be the best option, since that fit did not include the specification that year only starts as a predictor in 1900, so instead we use the same values as in (2)

Note that the following code for conf and prediction intervals is strictly valid only for **simple linear regression, not multiple.**

```
#> compute confidence and prediction intervals for the predicted
#   x: year
#   y: AMO
n = len(df_reg.index)
x = df_reg['year'].values
x_bar = x.mean()
s_x = np.sqrt( np.sum((x-x_bar)**2) / (n-1) )  # x.std() does population version!!
y_hat = np.dot(X, fit.params)  # should give same as fit.predict(X).values, or fit.fittedvalue
resid = y - y_hat
```