SIO221a_Lecture05b_python

October 9, 2023

SIO221a Notes - Alford and Gille

0.1 Lecture 5b

Fitting a function to data: least-squares fitting (Python version) *Thanks to Bia Villas Boas for this Python version.*

Now, let's return to our time series. You might remember we were looking for a linear trend for:

$$\mathbf{T} = T_o + b\mathbf{t} + \mathbf{n}, \tag{5}$$

where $\mathbf{T}$ represents our measured temperature data (as a vector), T_o is a constant (unknown), $\mathbf{t}$ is time, and b is the time rate of change (also unknown), and since this is the real world, $\mathbf{n}$ is noise (representing the part of the signal that isn't a linear trend. Formally, provided that we have more than two measurements, aside from the unknown noise vector, this is an over-determined system. Since the noise is unknown, and there are lots of independent values, the system is formally underdetermined. But we won't lose hope. We just move forward under the assumption that the noise is small.

Last time we started writing this as a matrix equation:

$$\mathbf{Ax} + \mathbf{n} = \mathbf{y}, \tag{6}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ 1 & t_3 \\ \vdots & \vdots \\ 1 & t_N \end{bmatrix}, \tag{7}$$

making $\mathbf{A}$ an $N \times 2$ matrix. And $\mathbf{y}$ is an N-element column vector containing, for example, our temperature data:

$$\mathbf{y} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_N \end{bmatrix}. \tag{8}$$

Then $\mathbf{x}$ is the vector of unknown coefficients, in this case with 2 elements (e.g. $x_1 = T_o$ and $x_2 = b$).

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{9}$$

How can we find the best solution to this equation to minimize the misfit between the data $\mathbf{y}$ and the model $\mathbf{Ax}$? The misfit could be positive or negative, and absolute values aren't mathematically tractable, so let's start by squaring the misfit.

$$\epsilon = (\mathbf{Ax} - \mathbf{y})^T(\mathbf{Ax} - \mathbf{y}) = \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - 2\mathbf{x}^T\mathbf{A}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}. \tag{10}$$

Then we can minimize the squared misfit. The natural route to minimization comes by taking the derivative, and then setting the results equal to zero. Our unknown is $\mathbf{x}$, so we'll minimize in terms of that:

$$\frac{\partial\epsilon}{\partial\mathbf{x}} = 2\mathbf{A}^T\mathbf{Ax} - 2\mathbf{A}^T\mathbf{y} = 0, \tag{11}$$

and this implies that

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{y}, \tag{12}$$

so

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}. \tag{13}$$

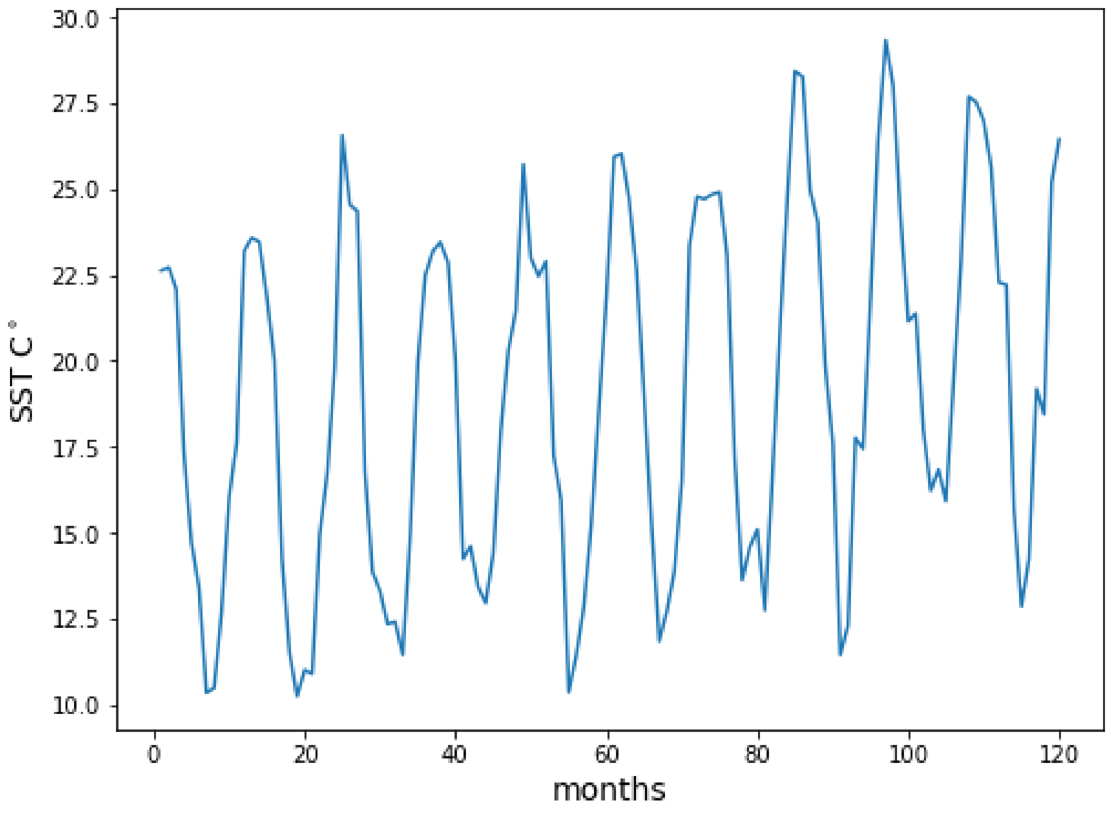
In the example below we're going to work with some fake data. This is meant to give you some insight on the power of using fake data to test your code. When you generate fake data carefully, you prescribe certain parameters (e.g. mean and standar deviation), thus you exactly what answer to expect and can detect potential problems with your code. You will see throughout this course that fake data is a very powerful tool!

[11]:
```python
import xarray as xr

# Generate fake monthly SST data
time = np.arange(1, 12*10+1) # each value is one month
mean_temp = 16
annual_cycle = 5*(np.sin(2 * np.pi * (time / 12)) + np.cos(2 * np.pi * (time /␣
 ↪12)))
trend = 0.05*time
noise = 5*(np.random.rand(len(time)) - 0.5)
temperature = mean_temp + trend + annual_cycle + noise
```

[12]:
```python
plt.figure(figsize=(8,6))
plt.plot(time, temperature)
plt.ylabel('SST C$^\\circ$', fontsize=14)
plt.xlabel('months', fontsize=14)
```

```
[12]: Text(0.5, 0, 'months')
```

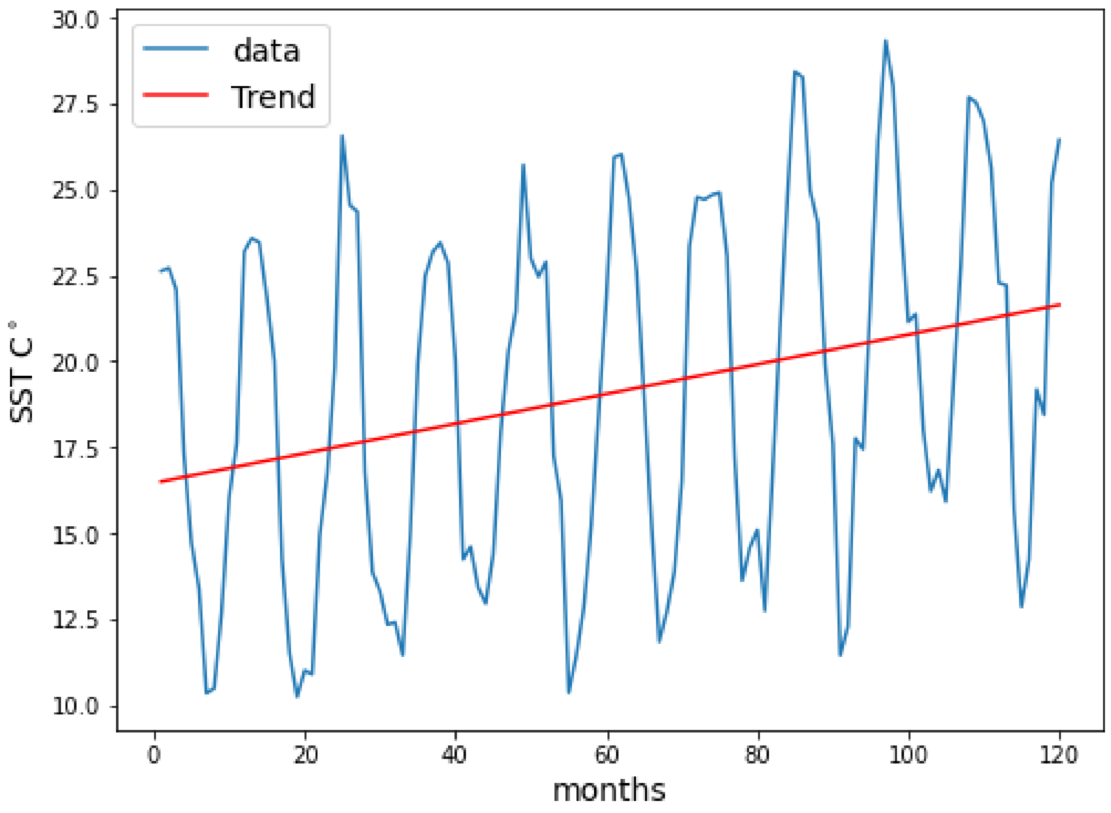


```
[13]: import numpy as np
      from scipy.linalg import inv
      nt = len(time)
      A = np.array([np.ones(nt), time]).T
      temperature = temperature.reshape([nt, 1])
      x = np.dot(inv(np.dot(A.T, A)), np.dot(A.T, temperature))
      fit = np.dot(A, x)
      print('Mean from fit:', x[0][0], 'Linear trend from fit:', x[1][0])
```

```
Mean from fit: 16.45587994273605 Linear trend from fit: 0.043162317242879965
```

```
[14]: plt.figure(figsize=(8,6))
      plt.plot(time, temperature, label = 'data')
      plt.plot(time, fit, 'r', label = 'Trend')
      plt.ylabel('SST C$^\\circ$', fontsize=14)
      plt.xlabel('months', fontsize=14)
      plt.legend(fontsize=14)
```

```
[14]: <matplotlib.legend.Legend at 0x7ff4fc8469d0>
```



As we've noted, if we want to find a trend, we define **A** to have a column of ones (to identify the mean) and a column containing the time, to identify the rate of change. We can make our model **A** progressively more complicated by adding additional columns. What do we do if we want to find the annual cycle? Before I give you any answers, take a moment to think about this.

We could use:

$$\mathbf{A} = \begin{bmatrix} 1 & \cos(t_r) & \sin(t_r) \\ \vdots & \vdots & \vdots \end{bmatrix}, \qquad (14)$$

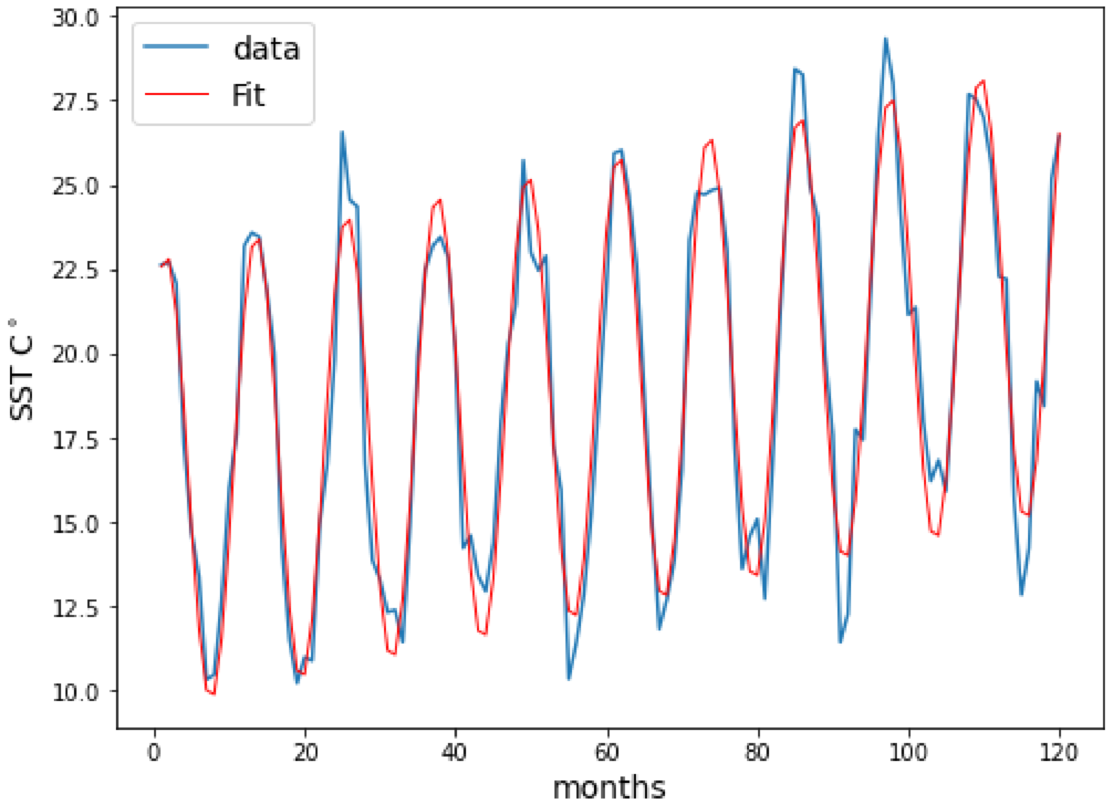
where time t is measured in days, and $t_r = 2\pi t/365.25$, is the time in radians. We need the sine and cosine because we don't know the phase of our annual cycle exactly. You might imagine that we could fit for the phase (e.g. $\sin(t_r + \phi)$), and we could, but that would be a non-linear fitting process, and the power of least-squares fitting won't work if we try that—we'd quickly be plunged into the murky world of non-linear fitting procedures, which is messy, unreliable, and not necessary in this case. Here's an example

[15]:
```
A2 = np.array([np.ones(nt), time, np.sin(2*np.pi*time/12), np.cos(2*np.pi*time/
 ↪12) ]).T
x = np.dot(inv(np.dot(A2.T, A2)), np.dot(A2.T, temperature))
fit = np.dot(A2, x)
print('Mean from fit:', x[0][0], '\nLinear trend from fit:', x[1][0],
      '\nAmplitude of sine:', x[2][0],'\nAmplitude of cosine:', x[3][0])
```

```
Mean from fit: 16.099879960655283
Linear trend from fit: 0.04904661446735552
Amplitude of sine: 4.997136470603355
Amplitude of cosine: 4.528234578109797
```

[16]:
```
plt.figure(figsize=(8,6))
plt.plot(time, temperature, label = 'data')
plt.plot(time, fit, 'r', lw=1, label = 'Fit')
plt.ylabel('SST C$^\\circ$', fontsize=14)
plt.xlabel('months', fontsize=14)
plt.legend(fontsize=14)
```

[16]:
```
<matplotlib.legend.Legend at 0x7ff4fc846990>
```


data
Fit
SST C°
months