PH2255 Course: Introduction to Statistical Methods Exercise 1

Thomas Bass

15 January 2021

Abstract

Exercise One in the PH2255 course begins with an introduction to using the least-squares method provided in SciPy's curve_fit method, and using the error propagation formula described below to generate the standard deviation of the generated parameters. Plots are generated to show how well a least-squares method fits the data, using first, second, and third order polynomials.

First, we must ingest the data provided in the lab script into Python. We use Numpy's np.array as this will allow us to manipulate the data later.

```
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])
```

Now that the data is ingested, we can use SciPy's curve_fit method to generate the optimal values for the parameters, so that the sum of the squared residuals is minimized.

First, we define a general polynomial function at a given value of x and list of coefficients theta, using this general form of the polynomial function:

$$f(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + ... \tag{1}$$

In python...

```
def polynomial(x, *theta):
  return sum([theta[i]*x**i for i in range(len(theta))])
```

This is a very efficient "one-liner" in python, using list comprehension to iterate over i for a range defined by the length of theta. The use of *theta allows the parameter vector $\boldsymbol{\theta} = (\theta_0, \theta_1)$ to have arbitrary length, useful for coding higher order polynomials.

Next, we use the curve_fit to generate our parameters and covariance matrices for each of the first, second, and third order polynomials:

```
p0_first = np.array([1.0, 1.0])
theta_hat_first, covariance_first = curve_fit(polynomial, x, y,
    p0_first, sig, absolute_sigma=True)
```

The same code was repeated, with renamed variables and corresponding `p0` arrays for the second and third order polynomials.

Now that we have the parameters, we can plot our fitted line using the corresponding parameters as polynomial coefficients. To do this in MatPlotLib, we define a function `fit`, similar to `polynomial` but without the asterisk unpacking the array:

```
def fit(x, theta):
  return sum([theta[i]*x**i for i in range(len(theta)) ])
```

The exercise asks us to plot the standard deviation of the fitted function as well as the fit its self, so we use Equation 26 from the script (slightly modified for this specific use):

$$\sigma_f^2 \approx \sum_{i,j=0}^{m} \frac{\partial f}{\partial \hat{\theta}_i} \frac{\partial f}{\partial \hat{\theta}_i j} U_{ij} \tag{2}$$

Where $U_{ij} = \mathrm{cov}[i,j]$. In our example, f is the polynomial function, so the derivatives rapidly simplify with use of the Kronecker Delta function:

$$\frac{\partial f(x, \vec{\hat{\theta}})}{\partial \hat{\theta}_i} = \frac{\partial}{\partial \hat{\theta}_i} \sum_{k=0}^{N} \hat{\theta}_k x^k = \sum_{k=0}^{N} \delta_{ik} x^k = x^i \tag{3}$$

This allows us to write the entire standard deviation formula very compactly:

$$\sigma_f^2 = \sum_{i,j=0}^{N} x^{i+j} \mathrm{cov}[i,j] \tag{4}$$

In python...

```
def std_dev(x, cov):
  return sum([(x**(i+j))*cov[i][j] for i in range(len(cov)) for j
      in range(len(cov))])
```

This again uses compact list comprehension to iterate over `i` and `j`, as defined by the length of the covariance matrix `cov`. *N.B.:* we take the square root of this function's results later, to reflect std. dev. $= \sqrt{\sigma^2}$.

We now use these two functions to plot our data in MatPlotLib (figure setup is omitted here):

```
ax.errorbar(x, y, yerr=sig, fmt='kx', label="test data")
ax.plot(x_val, fit(x_val, theta_hat_first), 'r', label="first order
    fit")
ax.fill_between(x_val, fit(x_val, theta_hat_first)-np.sqrt(std_dev(
    x_val, covariance_first)), fit(x_val, theta_hat_first)+np.sqrt(
    std_dev(x_val, covariance_first)), label="one standard
    deviation")
```

This, generates the following plots for the first order fit, and modified and repeated code for the second and third order fits:

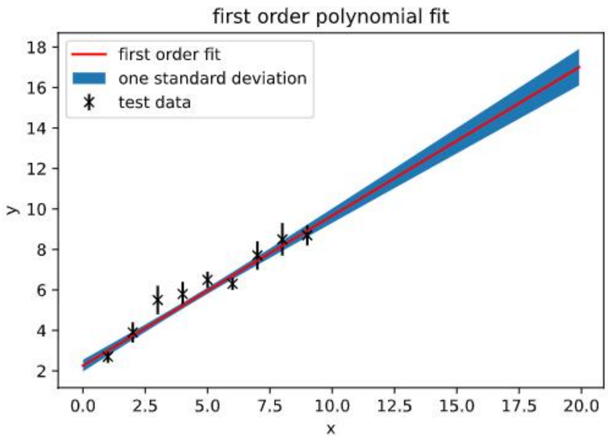

Figure 1: First-order polynomial fit, showing fit curve and one standard deviation above and below

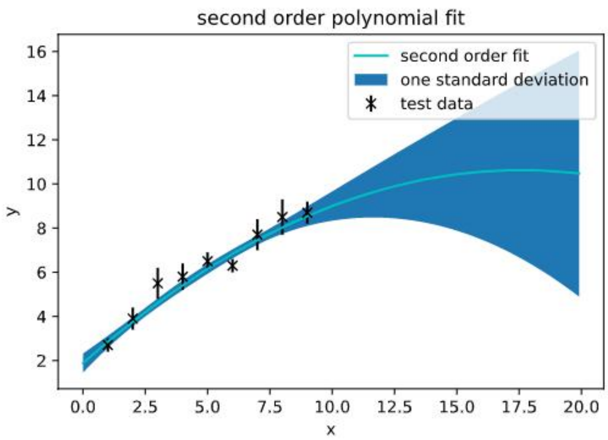

Figure 2: Second-order polynomial fit, showing fit curve and one standard deviation above and below

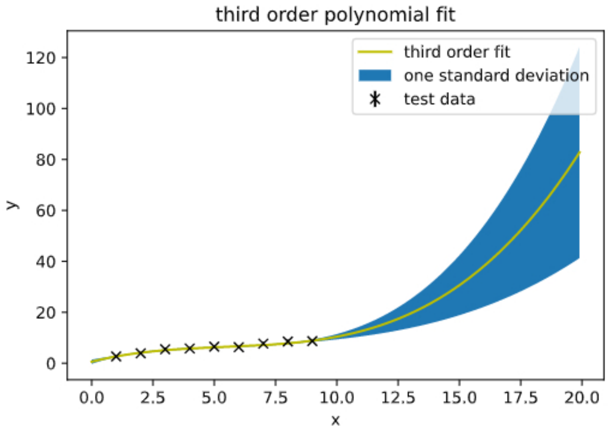

Figure 3: Third-order polynomial fit, showing fit curve and one standard deviation above and below

From these graphs, we can qualitatively see that the standard deviation for each fit extrapolated beyond the data sets increases more rapidly for higher-order polynomials.

We can also generate chi-squared values for each of these fits, using the following code:

```
sum(((y - polynomial(x, *theta_hat))/sig)**2)
```

The chi-squared values and degrees of freedom for each fit are tabulated below:

Polynomial Order	Chi-Squared	Degrees of Freedom
1	8.2515361178354	7
2	6.842115296038535	6
3	3.747761582200386	5

Table 1: Chi-square values and degrees of freedom for each fit.

A Python Code

The complete code used in this exercise is presented below, as well as the text outputs of the script:

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib
import matplotlib.pyplot as plt

## Input data arrays
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])

## Polynomial function
def polynomial(x, *theta): return sum([theta[i]*x**i for i in range
    (len(theta))])

## First order
p0_first = np.array([1.0, 1.0])
theta_hat_first, covariance_first = curve_fit(polynomial, x, y,
    p0_first, sig, absolute_sigma=True)

## Second order
p0_second = np.array([1.0, 1.0, 1.0])
theta_hat_second, covariance_second =  curve_fit(polynomial, x, y,
    p0_second, sig, absolute_sigma=True)

## Third order
p0_third = np.array([1.0, 1.0, 1.0, 1.0])
theta_hat_third, covariance_third =  curve_fit(polynomial, x, y,
    p0_third, sig, absolute_sigma=True)

## Fitting
def fit(x, theta): return sum([theta[i]*x**i for i in range(len(
    theta)) ])

## Standard Deviation
def std_dev(x, cov): return sum([(x**(i+j))*cov[i][j] for i in
    range(len(cov)) for j in range(len(cov))])

## Create plots
fig1 = plt.figure()
fig2 = plt.figure()
fig3 = plt.figure()
ax1 = fig1.add_subplot()
ax1.set_title('first order polynomial fit')
ax2 = fig2.add_subplot()
ax2.set_title('second order polynomial fit')
ax3 = fig3.add_subplot()
ax3.set_title('third order polynomial fit')
ax1.set_ylabel("y")
ax2.set_ylabel("y")
ax3.set_ylabel("y")
ax1.set_xlabel("x")
ax2.set_xlabel("x")
ax3.set_xlabel("x")

x_val = np.arange(0, 20, 0.1)

## Plot given data with sigma error bars
```

```python
ax1.errorbar(x, y, yerr=sig, fmt='kx', label="test data")
ax2.errorbar(x, y, yerr=sig, fmt='kx', label="test data")
ax3.errorbar(x, y, yerr=sig, fmt='kx', label="test data")

## Plot the three fits
ax1.plot(x_val, fit(x_val, theta_hat_first), 'r', label="first order fit")
ax1.fill_between(x_val, fit(x_val, theta_hat_first)-np.sqrt(std_dev(x_val, covariance_first)), fit(x_val, theta_hat_first)+np.sqrt(std_dev(x_val, covariance_first)), label="one standard deviation")
ax1.legend()

ax2.plot(x_val, fit(x_val, theta_hat_second), 'c', label="second order fit")
ax2.fill_between(x_val, fit(x_val, theta_hat_second)-np.sqrt(std_dev(x_val, covariance_second)), fit(x_val, theta_hat_second)+np.sqrt(std_dev(x_val, covariance_second)), label="one standard deviation")
ax2.legend()

ax3.plot(x_val, fit(x_val, theta_hat_third), 'y', label="third order fit")
ax3.fill_between(x_val, fit(x_val, theta_hat_third)-np.sqrt(std_dev(x_val, covariance_third)), fit(x_val, theta_hat_third)+np.sqrt(std_dev(x_val, covariance_third)), label="one standard deviation")
ax3.legend()

## Chi-squared and degrees of freedom
print (f"chisq first order = {sum(((y - polynomial(x, *theta_hat_first))/sig)**2)},\tndof = {len(x) - len(p0_first)}")
print (f"chisq second order = {sum(((y - polynomial(x, *theta_hat_second))/sig)**2)},\tndof = {len(x) - len(p0_second)}")
print (f"chisq third order = {sum(((y - polynomial(x, *theta_hat_third))/sig)**2)},\tndof = {len(x) - len(p0_third)}")
```

py.py

```
chisq first order = 8.25153611783541, ndof = 7
chisq second order = 6.842115296038535, ndof = 6
chisq third order = 3.747761582200386,  ndof = 5
```