Data Science Analysis - Assignment 4

Varenya Upadhyaya

February 15, 2023

1. The following code plots Fig. (1) and calculates the best fit parameters for linear, quadratic and cubic models. The code also calculates the AIC and BIC values for the three models along with the p-vlaues

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

x,y,sigma_y = np.loadtxt('1.txt', unpack=True)
data = np.vstack([x, y, sigma_y])
N = len(x)+1



def polynomial_fit(theta, x):
    return sum(t * x ** n for (n, t) in enumerate(theta))

def logL(theta, model=polynomial_fit, data=data):
    x, y, sigma_y = data
    y_fit = model(theta, x)
    return sum(stats.norm.logpdf(*args) for args in zip(y, y_fit, sigma_y))

def best_theta(degree, model=polynomial_fit, data=data):
    theta_0 = (degree + 1) * [0]
    neg_logL = lambda theta: -logL(theta, model, data)
    return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)

def aic(data,theta,model=polynomial_fit):
    deg=len(theta)
    aic =-2*logL(theta,model,data)+2*deg
    return aic
def bic(data,theta,N,model=polynomial_fit):
    k=len(theta)
    bic= -2*logL(theta,model,data)+k*np.log(N)
    return bic
theta1 = best_theta(1)
theta2 = best_theta(2)
theta3 = best_theta(3)

print('-Best fit parameters (lowest coefficient first)- ')
print('Linear:',theta1)
print('Quad:',theta2)
print('Cubic:',theta3)
print('\n-Maximum Likelihoods-')
print('Linear:', logL(best_theta(1)))
print('Quad:', logL(best_theta(2)))
print('Cubic:', logL(best_theta(3)))

print('\n-AIC Values-')
print('Linear:',aic(data,theta1))
print('Quad:',aic(data,theta2))
print('Cubic:',aic(data,theta3))

print('\n-BIC Values-')
print('Linear:',bic(data,theta1,N))
print('Quad:',bic(data,theta2,N))
print('Cubic:',bic(data,theta3,N))

```

```
#chi2 values
chi2_lin = np.sum(((y-theta1[0]-theta1[1]*x)/sigma_y)**2)
chi2_quad = np.sum(((y-theta2[0]-theta2[1]*x-theta2[2]*x**2)/sigma_y)**2)
chi2_cub = np.sum(((y-theta3[0]-theta3[1]*x-theta3[2]*x**2-theta3[3]*x**3)/sigma_y)**2)
p_lin_quad =1-stats.chi2(3-2).cdf(np.abs(chi2_quad-chi2_lin))
p_lin_cub =1-stats.chi2(4-2).cdf(np.abs(chi2_cub-chi2_lin))
print('\n-p-values wrt linear fit-')
print('Quad:',p_lin_quad)
print('Cubic:',p_lin_cub)
#plots
xfit = np.linspace(0, 1)
plt.figure(figsize=(9,7))

plt.errorbar(x,y,sigma_y, fmt='h', ms=6, color='#fd7f6f', mfc='#fd7f6f', mew=1, ecolor
    ='#fd7f6f', alpha=0.75, capsize=2.0, zorder=0, label='Data');
plt.plot(xfit, polynomial_fit(theta1, xfit), label='best linear model', color='#7eb0d5
    ', ls='--')
plt.plot(xfit, polynomial_fit(theta2, xfit), label='best quadratic model', color='#
    b2e061', ls='-.')
plt.plot(xfit, polynomial_fit(theta3, xfit), label='best cubic model', color='#bd7ebe')

plt.legend()
plt.savefig('1.png')
```

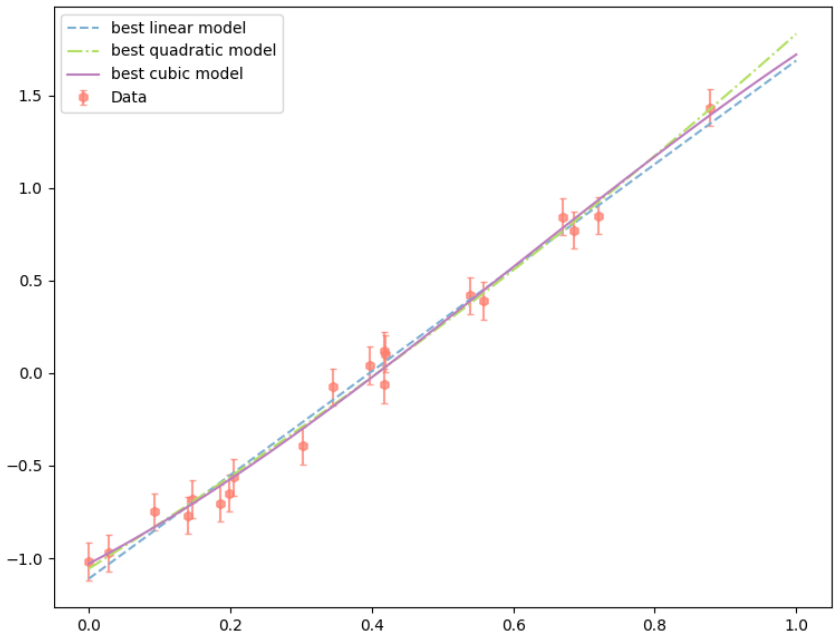

Figure 1: The best fit cubic, quadratic and linear models for the given data points

Code output:

```
-Best fit parameters (lowest coefficient first)-
Linear: [-1.11028083  2.79789862]
Quad: [-1.05578916  2.38475189  0.50261294]
Cubic: [-1.02910477  1.97184143  1.74451233 -0.96724987]

-Maximum Likelihoods-
Linear: 22.01834340803626
```

```
Quad: 22.92491031200274
Cubic: 23.130409258797624

-AIC Values-
Linear: -40.03668681607252
Quad: -39.84982062400548
Cubic: -38.26081851759525

-BIC Values-
Linear: -37.94764194062567
Quad: -36.716253310835214
Cubic: -34.08272876670156

-p-values wrt linear fit-
Quad: 0.17813275695316466
Cubic: 0.32887884419640634
```

The maximum likelihoods indicate that the cubic model would best fit the data. However, on further investigation, it is evident that the linear fit has the lowest AIC and BIC values making it the best model out of the three. This was also expected as all the models fit the data well but the AIC/BIC tests penalize a higher number of free parameters (as is the case in quadratic and cubic polynomials).

2. The following code computes the AIC and BIC values for the parameters obtained in JVDP's blog. The AIC and BIC values obtained are lower for the linear models and thus agree with the results on the blog.

```
import numpy as np
from scipy import stats, optimize

data = np.array([[ 0.42,  0.72,  0.  ,  0.3 ,  0.15,
                   0.09,  0.19,  0.35,  0.4 ,  0.54,
                   0.42,  0.69,  0.2 ,  0.88,  0.03,
                   0.67,  0.42,  0.56,  0.14,  0.2  ],
                 [ 0.33,  0.41, -0.22,  0.01, -0.05,
                  -0.05, -0.12,  0.26,  0.29,  0.39,
                   0.31,  0.42, -0.01,  0.58, -0.2 ,
                   0.52,  0.15,  0.32, -0.13, -0.09 ],
                 [ 0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1 ,
                   0.1 ,  0.1 ,  0.1 ,  0.1 ,  0.1  ]])

def polynomial_fit(theta, x):
    return sum(t * x ** n for (n, t) in enumerate(theta))
def best_theta(degree, model=polynomial_fit, data=data):
    theta_0 = (degree + 1) * [0]
    neg_logL = lambda theta: -logL(theta, model, data)
    return optimize.fmin_bfgs(neg_logL, theta_0, disp=False)
def logL(theta, model=polynomial_fit, data=data):
    """Gaussian log-likelihood of the model at theta"""
    x, y, sigma_y = data
    y_fit = model(theta, x)
    return sum(stats.norm.logpdf(*args)
               for args in zip(y, y_fit, sigma_y))
def aic(data,theta,model=polynomial_fit):
    deg=len(theta)
    aic =-2*logL(theta,model,data)+2*deg
    return aic
def bic(data,theta,N,model=polynomial_fit):
    k=len(theta)
    bic= -2*logL(theta,model,data)+k*np.log(N)
    return bic


```

```
x, y, sigma_y = data
N=len(x)
theta1 = best_theta(1)
theta2 = best_theta(2)

print('\n-AIC Values-')
print('Linear:',aic(data,theta1))
print('Quad:',aic(data,theta2))

print('\n-BIC Values-')
print('Linear:',bic(data,theta1,N))
print('Quad:',bic(data,theta2,N))
```

Code output:

```
-AIC Values-
Linear: -40.021734013225085
Quad: -39.883027173007946

-BIC Values-
Linear: -38.030269466117105
Quad: -36.895830352345975
```

The Linear model is thus, a better fit, which was expected again due to a lower number of free parameters compared to the quadratic model.

3. The Kolmogorov-Smirnov Test: Assessing the C/O Ratio Formation Diagnostic: A Potential Trend with Companion Mass
In the above paper, the carbon to oxygen ratio of multiple exoplanets is measured using the OSIRIS integral field spectrograph at the W.M. Keck Observatory. They report a C/O ratio of $0.589^{+0.148}_{-0.295}$. This ratio is then added to a list of C/O ratios for directly imaged planets. There is a trend in the ratio with companion mass (companions meaning other planets in the system) M_{Jup}, which breaks off around $4M_{Jup}$. The KS and AD tests are performed on planets above and below this mass range in the paper. The KS and AD test samples are randomly sampled 100,00 times from each parameter using a Gaussian distribution in the range $3 - 5M_{Jup}$ with $4M_{Jup}$ as the distinguishing mass. The two sets were then put through the KS test and it was reported that the p values were lesser than 1% implying that the two populations were distinct and unlikely to be drawn from the same population.
They also performed the AD test and reported similar results. Since this was only a one-dimensional analysis and the data was derived from a different model, this analysis used the KS test correctly as per the warnings on the Penn State website. Reference:

4. The following code calculates the sigma values of the discoveries:

```
from scipy import stats

#a Higgs_Boson p values for 1sigma to 6sigma
p_higgs = [1e-1,1e-2,1e-3,1e-5,1e-7,1e-9]
sig_higgs = stats.norm.isf(p_higgs)
print('Significance for 1sigma to 6sigma:',[ round(sig, 2) for sig in sig_higgs ])

#b lIGO
p_ligo = 2e-7
sig_ligo = stats.norm.isf(p_ligo)
print('Significance of the LIGO discovery:',round(sig_ligo,2))

#c Super-K discovery
chi2,dof = 65.2,67
gof = 1-stats.chi2(dof).cdf(chi2)
print('$\chi^2$ GOF for Super K:',round(gof,2))
```

Code output:

```
Significance for 1sigma to 6sigma: [1.28, 2.33, 3.09, 4.26, 5.2, 6.0]
Significance of the LIGO discovery: 5.07
$\chi^2$ GOF for Super K: 0.54
```

All the codes and figures used in this assignment can be found in this repository