L6.6 Returning to Fitting Supernova Data

| Top | Previous Section | Exercises | Next Section |

Example Project Solution

Let's pretend that this is your project peer-review solution. We expect your submission to be code followed by an explanation. Specifically, we expect you to do the following in the three sections below:

- Section 1: Develop a computational procedure to achieve the objective.
- Section 2: Explain your procedure.
- Section 3: Describe your results and characterize the significance (e.g., chi-squared, relevant parameters, etc.)

We expect a good level of detail describing what your code is doing and what results you see. View the explaination and results section after the code for an example of this.

NOTE: Submit this notebook by going to File->Print->Save as PDF (there is an example doc for how to do this if you are unsure). This functionality might work best for smaller notebooks so, if possible, **duplicate** your project notebook and delete all non-relevant cells. Remember: **duplicate** do not delete your original notebook.

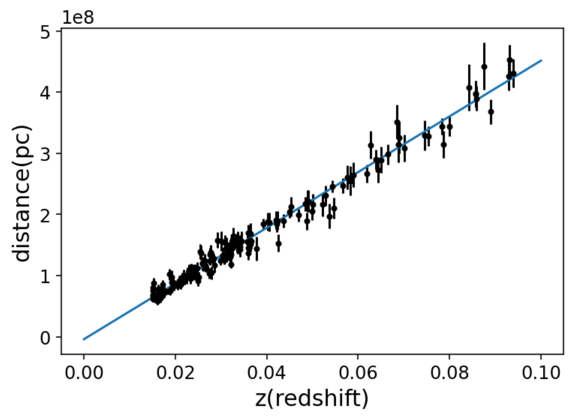
Section 1: Develop a procedure

Begin your work below.

```
#>>>RUN: L6.6-runcell01
import math
import numpy as np
import csv
import matplotlib.pyplot as plt
from scipy import stats

%matplotlib inline

#Let's try to understand how good the fits we made in last Lesson are, let's load the supernova data again
label='data/L04/sn_z_mu_dmu_plow_union2.1.txt'

def distanceconv(iMu):
    power=iMu/5+1
    return 10**power

def distanceconverr(iMu,iMuErr):
    power=iMu/5+1
    const=math.log(10)/5.
    return const*(10**power)*iMuErr

def load(iLabel,iMaxZ=0.1):
    redshift=np.array([])
    distance=np.array([])
    distance_err=np.array([])
    with open(iLabel,'r') as csvfile:
        plots = csv.reader(csvfile, delimiter='\t')
        for row in plots:
            if float(row[1]) > iMaxZ:
                continue
            redshift = np.append(redshift,float(row[1]))
            distance = np.append(distance,distanceconv(float(row[2])))
            distance_err = np.append(distance_err,distanceconverr(float(row[2]),float(row[3])))
    return redshift,distance,distance_err
```

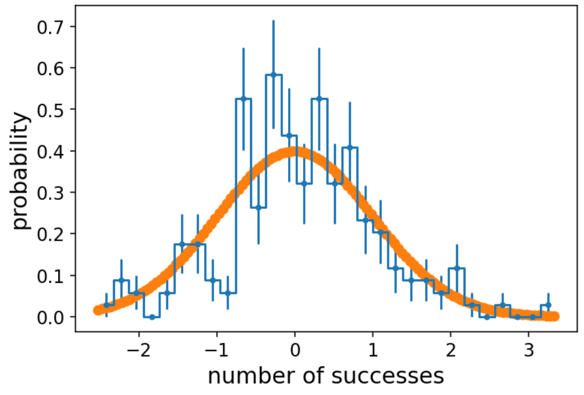
✓ 0s completed at 8:32 AM

```
#Now let's run the regression again
def variance(isamples):
    mean=isamples.mean()
    n=len(isamples)
    tot=0
    for pVal in isamples:
        tot+=(pVal-mean)**2
    return tot/n

def covariance(ixs,iys):
    meanx=ixs.mean()
    meany=iys.mean()
    n=len(ixs)
    tot=0
    for i0 in range(len(ixs)):
        tot+=(ixs[i0]-meanx)*(iys[i0]-meany)
    return tot/n

def linear(ix,ia,ib):
    return ia*ix+ib

redshift,distance,distance_err=load(label)
var=variance(redshift)
cov=covariance(redshift,distance)
A=cov/var
const=distance.mean()-A*redshift.mean()
xvals = np.linspace(0,0.1,100)
yvals = []
for pX in xvals:
    yvals.append(linear(pX,A,const))

plt.plot(xvals,yvals)
plt.errorbar(redshift,distance,yerr=distance_err,marker='.',linestyle = 'None', color = 'black')
plt.xlabel("z(redshift)")
plt.ylabel("distance(pc)")
plt.show()
```



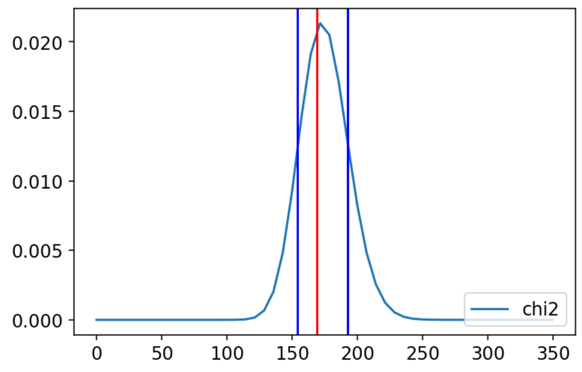
```
#>>>RUN: L6.6-runcell02

def residualsComp(redshift,distance,distance_err):
    #Compute residuals
    residuals=np.array([])
    for i0 in range(len(redshift)):
        pResid=linear(redshift[i0],A,const)-distance[i0]
        residuals = np.append(residuals,pResid/distance_err[i0])

    #Make a histogram
    y0, bin_edges = np.histogram(residuals, bins=30)
    bin_centers = 0.5*(bin_edges[1:] + bin_edges[:-1])
    norm0=len(residuals)*(bin_edges[-1]-bin_edges[0])/30.
    plt.errorbar(bin_centers,y0/norm0,yerr=y0**0.5/norm0,marker='.',drawstyle = 'steps-mid')

    #Plot a Gaussian
    k=np.arange(bin_edges[0],bin_edges[-1],0.05)
    normal=stats.norm.pdf(k,0,1)
    #First let's look at the moments
    normalpoints=stats.norm.rvs(0,1,1000)
    print_moments(residuals,"residuals")
    print_moments(normalpoints,"normal distribution")

    #Now let's plot it
    plt.plot(k,normal,'o-')
    plt.xlabel("number of successes")
    plt.ylabel("probability")
    plt.show()
    return residuals

residuals=residualsComp(redshift,distance,distance_err)
```

```
residuals mean: 0.0674380962700456
residuals var: 0.9619236126333892
residuals skew: 0.1553936844426706
residuals kurtosis: 3.274051707095445
normal distribution mean: 0.024298062521039762
normal distribution var: 1.0262324979726034
normal distribution skew: -0.035902378671777778
normal distribution kurtosis: 3.1562090305767825
```



```
#>>>RUN: L6.6-runcell03

#now let's look at the chi2
chi2=np.sum(residuals**2)

print("Total chi2:",chi2,"NDOF",len(residuals)-2)
print("Normalized chi2:",chi2/(len(residuals)-2))
print("Probability of chi2:",1-stats.chi2.cdf(chi2,(len(residuals)-2)))
print()

#Let's plot it for good measure too
x = np.linspace(0,len(residuals)*2)
chi2d=stats.chi2.pdf(x,len(residuals-2)) # 40 bins
plt.plot(x,chi2d,label='chi2')
plt.axvline(chi2, c='red')
plt.legend(loc='lower right')

ndof=(len(residuals)-2)
chi2ppf0=stats.chi2.ppf(0.5,ndof)
chi2ppf1=stats.chi2.ppf(0.15,ndof)
chi2ppf2=stats.chi2.ppf(0.85,ndof)
#chi2ppf1=stats.chi2.ppf(0.025,ndof)
#chi2ppf2=stats.chi2.ppf(1-0.025,ndof)
print("Central Vvalue",chi2ppf0)
print("Sigma Low",chi2ppf1-chi2ppf0)
print("Sigma High",chi2ppf2-chi2ppf0)

plt.axvline(chi2ppf1, c='blue')
plt.axvline(chi2ppf2, c='blue')
plt.show()
```

```
Total chi2: 169.1325141558355 NDOF 173
Normalized chi2: 0.977644590496159
Probability of chi2: 0.5688982622097782

Central Vvalue 172.3337919816806
Sigma Low -18.526264786058277
Sigma High 19.957893469399238
```



Section 2: Explanation of procedure

Explain your code here: In this example, we fit the cosmological data with redshifts(z) from [0, 0.10] as a function of distance(d). This was fit with a linear function, $f(z) = az + b$. In our usual slope-intercept form, slope was calculated to be the ratio of the variance to covariance: $a = \frac{var_z}{cov_{z,d}}$. Variance was defined as to be the $\mathrm{var\_z} = \frac{\sum (z_i - \bar{z})^2}{n-1}$ while covariance was defined as $\mathrm{cov}_{z,d} = \frac{\sum (z_i - \bar{z})(d_i - \bar{d})}{N-1}$ with covariance being taken between the redshift and distance. This gives us an intercept of $b = \overline{d} - a\overline{z}$. This gives us a final form of

$$f(z) = \frac{var_z}{cov_{z,d}} z + \left(\overline{d} - \frac{var_z}{cov_{z,d}} \overline{z} \right)$$

Next, residuls were plotted to observe any higher dimensional trends in the data. Residuals were computed for each i-th datapoint:

$$\frac{f(z_i) - d_i}{\sigma_i}$$

where σ_i is the i-th datapoint's error provided from the dataset. Finally, the χ^2 value was calculated as the sum of squares of the residuals:

$$\chi^2 = \sum_i \left(\frac{f(z_i) - d_i}{\sigma_i} \right)^2$$

Section 3: Explanation of results

Describe your results here: We first see that the linear fit from z = [0, 0.10] looks pretty good, especially given that the datapoints fall within error bars of the linear fit. Another promising sign is that the residuals are close to 0 with a variance of 1, meaning that our residuals are Gaussian distributed. We see that the $\chi^2 = 169.13$ is very close to the number of degrees of freedom, $\mathrm{NDF} = 173$. Recall that the degrees of freedom need to be adjusted to incorporate the fit parameters introduced:

$$\mathrm{NDF} = 175(\text{length of residuals}) - 1(\text{fit parameter a}) - 1(\text{fit parameter b}) = 173$$

This produces a normalized $\chi^2/\mathrm{NDF} \approx 1$, which is a legitimately good fit! We can thus say that the probability of this chi-squared

$$\mathbb{P}(\chi^2) = 1 - \mathrm{CDF}(\chi^2, \mathrm{NDF}) = 56.8\%$$

which far exceeds our threshold of 5% (1 in 20 chance of randomly occuring). Thus, our linear fit from z = [0, 0.10] is a great fit!

```
print("Total chi2:",chi2,"NDOF",len(residuals)-2)
print("Normalized chi2:",chi2/(len(residuals)-2))
print("Probability of chi2:",1-stats.chi2.cdf(chi2,(len(residuals)-2)))
```

```
    Total chi2: 169.1325141558355 NDOF 173
    Normalized chi2: 0.977644590496159
    Probability of chi2: 0.5688982622097782
```