Research Methods

Robert E Simpson

Singapore University of Technology & Design

robert_simpson@sutd.edu.sg

September 26, 2018

Week 3 objectives

You should be able to:

- Use the SciPy.stats library distribution functions to calculate random variables, look up statistics, compute probabilities
- Fit curves to data points
- Analyse probability distributions
- Create probability plots
- Fit curves to data
- Analyse the goodness of a curve fit
- Plot histograms and probability density functions

We will also perform an experiment.

Useful commands

Look up the following commands

`len` Returns the length of a list

`numpy.sort` Sorts an array in ascending order

`numpy.mean` Calculates the arithmetic mean

`numpy.var` Calculates the population variance

`numpy.var(mydata, ddof=1)` Calculates the sample variance

`numpy.trapz` Integrates using the trapezoidal method

Useful distribution functions

SciPy distributions:

`ss.norm`	Normal distritibution
`ss.binom`	Binomial distribution
`ss.poisson`	Poisson distribution
`ss.lognorm`	Log Normal distritibution
`ss.rayleigh`	Rayleigh distribution
`ss.weibull_min`	Weibull distribtion
`ss.gamma`	Gamma distribution
`ss.chi2`	χ^2 distribution
`ss.f`	F-distribution

First you will need to import the SciPy library:

```
import scipy.stats as ss
```

Using the scipy.stats library

`print ss.norm(0,1).pdf(x)` Returns the values of the Normal PDF at x standard deviations. In this example, $\mu = 0$ and $\sigma = 1$, i.e. a standard normal curve

`print ss.norm(10,20).rvs(100)` Returns 100 random variables from a normal distribution with a mean of 10 and standard deviation of 20

`print ss.norm(`μ`,`σ`).cdf(x)` Returns the proportion of samples with a standard deviation less than x

`ss.norm(`μ`,`σ`).ppf(P)` Returns the inverse CDF, i.e. the x-value at which the area (probability) under the CDF is equal to a particular value, P. This can used instead of looking up values in standard normal tables.

Curve Fitting

There are a number of routines for fitting curves to data, but we will mainly use `scipy.optimize.curve_fit`. Use the following procedure:

i. Import/generate the x and y data
ii. Define the function that you want to fit to the data
iii. Perform the curve fitting using: `params = curve_fit(function2fit, x, y)`
iv. The fit parameters are stored in the first line of a *tuple* in the `params`. If there are two fit parameters (A1 and A2), then they can be accessed using `[A1, A2]=params[0]`.
v. Now redraw the function using the function and its fit parameters.

Goodness of fit, R^2

The coefficient of determination is used as a measure of goodness of fit.
It is defined as:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Where the Sum of Square Residuals, $SS_{res} = \sum_i (y_i - f_i)^2$ and the total sum of squares is $SS_{tot} = \sum_i (y_i - \bar{y})^2$

Notice that we are comparing the the difference between each measured value, y_i, and its modelled value, f_i, with the corresponding difference between each measured value, y_i, and the mean of y, $\bar{y}$. This is the fraction of variance unexpected, since we are comparing the unexpected modelled variance of each point, with the expected variance of each point. $R^2 = 0.49$ means 49% variability is expected, and 51% variability is unexpected. The unexpected variability is due to a problem with the fitting model not matching the data. $R^2 = 1$, means that there is a perfect match between the model data and the measured data.

Plotting Histograms

The matplotlib library has a good function for creating histograms and PDFs.

```
import matplotlib.pyplot as plt

data=[1.6, 2, 1, 1.3, 1, 2.3, 2.6, 1.3, 3, 2.3, 3.6,
    2, 3.3, 1.6, 1, 2, 3.6, 2.6, 1.6, 2.6, 2.6, 3.6,
    2.6, 1.3, 3.3, 2., 2.6, 2.6, 2, 3.3, 4.3, 2.3, 2,
    1.6, 2, 3, 1.6, 1.6, 2, 2.6, 3.6, 1, 2.3, 2.3, 3.,
     1.3, 2, 3, 2.3]

plt.figure()
plt.hist(data, bins='auto', facecolor='red',edgecolor=
    "k")
plt.ylabel("Frequency")
```

Plotting PDFs

```
import matplotlib.pyplot as plt

data=[1.6, 2, 1, 1.3, 1, 2.3, 2.6, 1.3, 3, 2.3, 3.6,
    2, 3.3, 1.6, 1, 2, 3.6, 2.6, 1.6, 2.6, 2.6, 3.6,
    2.6, 1.3, 3.3, 2., 2.6, 2.6, 2, 3.3, 4.3, 2.3, 2,
    1.6, 2, 3, 1.6, 1.6, 2, 2.6, 3.6, 1, 2.3, 2.3, 3.,
     1.3, 2, 3, 2.3]
plt.figure()
myhist=plt.hist(data, normed=True, bins='auto',
    facecolor='green',edgecolor="k")
plt.ylabel("P(x)")
```

Case Problem 3.6

a Use the scipy library to plot the standard normal PDF from -5σ to $+5\sigma$

b Use the scipy library to plot the standard normal CDF from -5σ to $+5\sigma$

c Use your plots to estimate the standard deviation between which we would expect to include 80% of the data?

d Check your answer to (c), using the built-in inverse CDF function

Case Problem 3.7

Probability plots

Ten observations on the life time in minutes of a well known computer battery brand are as follows: 176, 191, 214, 220, 205, 192, 201, 190, 183, 185.

Does a normal distribution accurately model the observations?

(a) Write a function to create a normal probability plot from the battery lifetime data points

(b) Use the curve fit function to check the linearity

(c) Compute the R^2 for the curve fit

CP 3.6 Solution

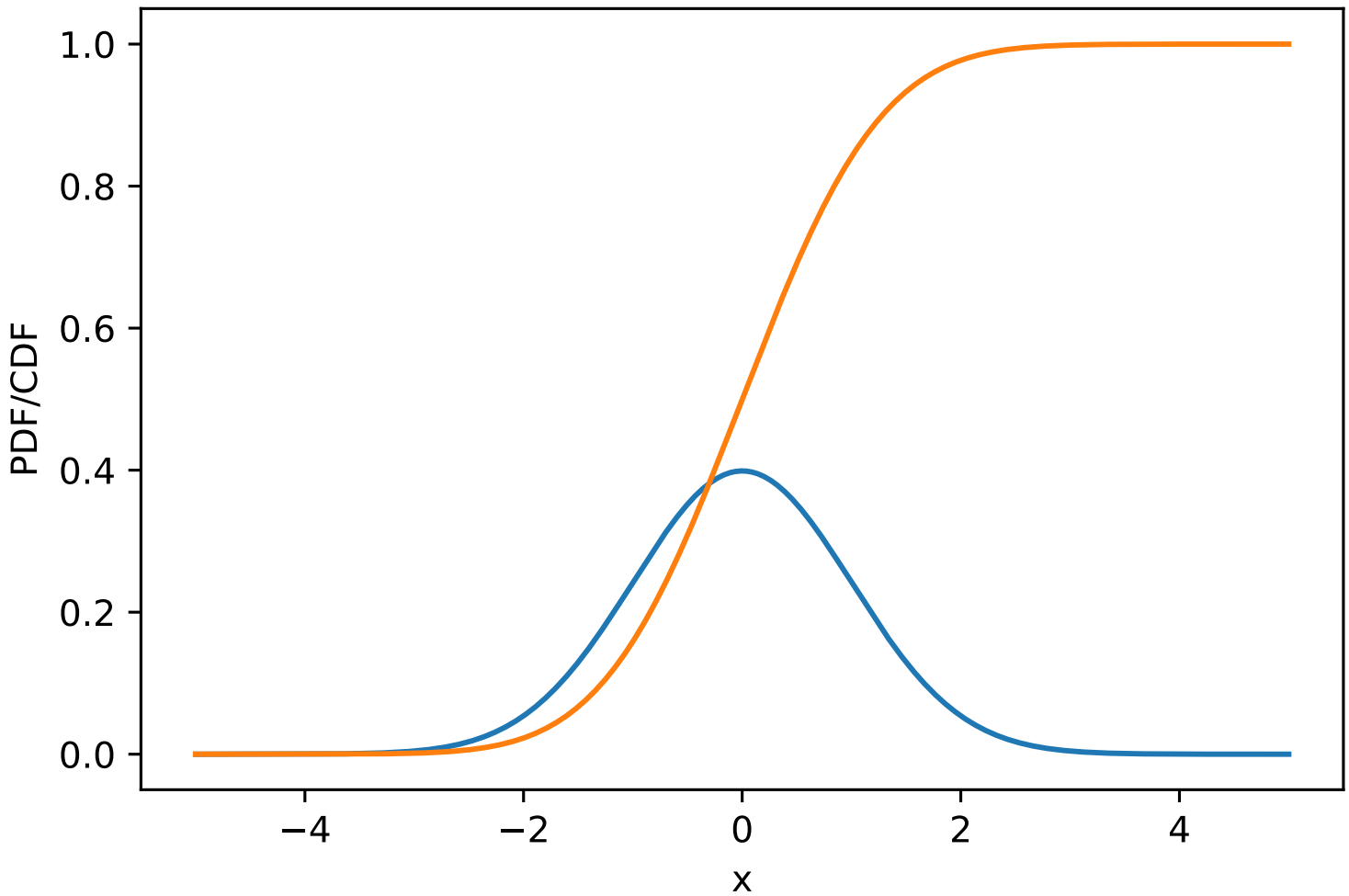
```
x=np.linspace(-5,5,1001)
y=ss.norm(0,1).pdf(x)
cdf=ss.norm(0,1).cdf(x)

x1=ss.norm(0,1).ppf(0.1)
x2=ss.norm(0,1).ppf(0.9)

print '80\% will be between', x1, 'stdev and', x2, '
    stdev'

plt.plot(x,y)
plt.plot(x,cdf)
```

Case Problem 3.6 Solution



CP 3.7 Solution

```
def linefunc(x, m, c):  #define function to fit
    return m*x + c

def r2(meas_data, model_data):  #compute R-squared
    avg=np.mean(meas_data)
    print "The mean is", avg
    i=0
    ST=[]
    SE=[]
    for line in meas_data:
        ST.append((line-avg)**2) # Square spread
        SE.append((line- model_data[i])**2)     # Sq.
            Errors
        i=i+1    #counter
    SST=sum(ST)
    SSE=sum(SE)
    return (1-(SSE/SST))

A=[176, 191, 214, 220, 205, 192, 201, 190, 183, 185]
```

CP 3.7 Solution

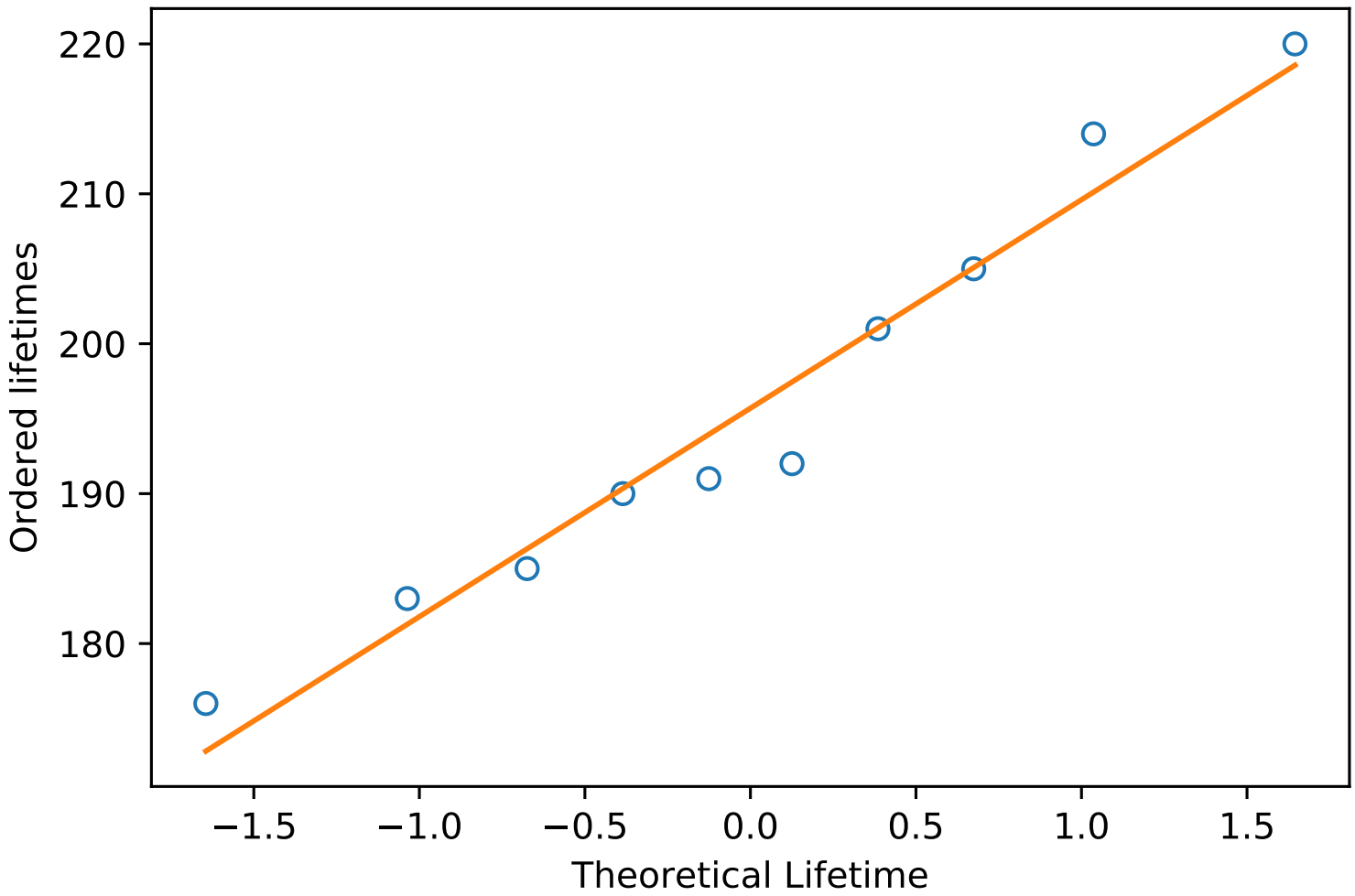
```
ranked=np.sort(A)
Z_stat=[]
for i in range(len(ranked)):
    score=((i+1)-0.5)/len(ranked)  #i starts at zero
        but should start at 1, therefore add 1 to i
    Z_stat.append(ss.norm(0,1).ppf(score))


params = curve_fit(linefunc, Z_stat, ranked)
[m, c]=params[0]

fit=[]  #creat the fit data
for z in Z_stat:
    fit.append(linefunc(z,m,c))

plt.figure()
plt.plot(Z_stat, ranked, marker='o', linestyle='none',
     markerfacecolor='None')
plt.plot(Z_stat, fit)
plt.xlabel('z-statistic')
plt.ylabel('Ordered lifetimes')
```

Case Problem 3.7 Solution



CP 3.7 Solution

The R-squared value is 0.9602659706, which is close to 1 and therefore a very good fit.

Case Problem 3.8

Geiger Experiment

a Collect samples of the background radiation counts per a second (cps).

b Plot a histogram based the results.

c Which PDF best describes the results? Test the fit by calculating R^2.

d Repeat the process with the green glasses

e What do you conclude?

Uranium Dioxide

Passage from Wikipedia:

> *Uranium dioxide or uranium(IV) oxide (UO2), also known as urania or uranous oxide, is an oxide of uranium, and is a black, radioactive, crystalline powder that naturally occurs in the mineral uraninite. It is used in nuclear fuel rods in nuclear reactors. A mixture of uranium and plutonium dioxides is used as MOX fuel. Prior to 1960, it was used as yellow and black color in ceramic glazes and glass.*

Uranium Glass