

# Exercise\_4

October 19, 2018

## 1 Exercise 4

```
In [ ]: from __future__ import print_function # For Python < 3
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

%matplotlib inline
```

### 1.1 1. Approximations to the binomial

For  $np < 10$ , large  $n$ , the Poisson distribution is a good approximation for the binomial.

- Show analytically that the binomial distribution converges to the Poisson distribution in the limit of large  $n$ . (Hint:  $e = \lim_{n \rightarrow \infty} (1 + \frac{1}{x})^x$ )
- Keeping  $np$  fixed, plot the binomial probability mass function for an increasing number of observations  $n$ , comparing in each case to the equivalent Poisson distribution ( $\lambda = np$ ). For convenience, you should use the relevant functions in `scipy.stat`.

```
In [ ]: n_trials = [5, 10, 100]
p0 = 0.8
...

# Plot the PMFs
fh, ax = plt.subplots(1, len(n_trials), sharey=True, figsize=(10,4))
for idx, nt in enumerate(n_trials):
    ...
    ax[idx].bar(...)
    ax[idx].bar(...)
...

```

For  $np > 10$ ,  $n(1-p) > 10$ , the discrete binomial distribution can be reasonably approximated by the continuous normal distribution.

- Choose a large  $n$  ( $> 30$ , with  $p$  close to 0.5). To start with, choose  $n=100$  and  $p=0.45$ . Plot the binomial pmf, and, with equivalent parameters, the normal pdf

- Calculate the probability that  $X \geq 55$  for each. Don't forget to apply the continuity correction
- What happens to the relative difference as  $n$  increases?

```
In [ ]: n_trials = 100
        p0 = 0.45
        mu = ... # Expectation
        std = ... # Standard deviation

        #...

        print('Binomial (exact):', ...)
        print('Gaussian (approximate):', ...)

        # Plots
        ...
```

## 1.2 2. Random walk

Consider a simple 1D random walk. A person starts at the position  $x = 0$ . With equal probability  $p = 0.5$ , they may take one step forwards or one step backwards, corresponding to a displacement of +1 and -1 respectively.

- Show that for an  $N$  step walk, the expected absolute distance from the starting position is given by  $\sqrt{N}$ .
- Write a function to simulate such a random walk, parameterised by the number of steps. The output should be an array, with the displacement at each step index.
- Plot a single walk.

```
In [ ]: def random_walk(n_steps):
        return ...

        n_steps = 100
        w = random_walk(n_steps)

        ...
```

- Simulate ~1000 random walks of 500 steps.
- Plot the average distance (rms) of these over the whole set with respect to step index (time). Does the average converge to the expected distance?
- (Optional) sample and plot the running average to show how the convergence improves with number of walks.

```
In [ ]: n_steps = 500
        n_walks = 1000
```

```

n = np.arange(n_steps) + 1

print('Expected distance for {} steps: {}'.format(...))

W = [] # Final distance
A = [] # Running average over whole set
T = 0
for idx in range(n_walks):
    ...

# ...

```

- Now consider the case  $p \neq 0.5$ , where the "person" is more likely to step in one direction than another. Find again analytically the expectation and the variance for the (rms) distance travelled in terms of  $N$  and  $p$ .
- Modify the random\_walk function to account for the unequal probability between the directions.
- Run a series of random walks as before, and plot again the histogram of distances travelled. On top of this, plot the Gaussian PDF with the  $\mu$  and  $\sigma$  parameters as determined above.

```

In [ ]: # Plot histogram
n_steps = 2000
n_trials = 5000
p = 0.4

V = []
for n in range(n_trials):
    ...

...

```

### 1.3 3. Small sample sizes: t-distribution

#### 1.3.1 3.1 Compare to normal distribution

Student's t-distributions are interesting for cases where you have few samples and the population variance is unknown, but the underlying distribution of the means can be assumed normal. They are parameterised by the degrees of freedom ("df"), which is usually equal the number of samples minus one. As the number of degrees of freedom increases, the t-distribution converges to the normal distribution.

- Plot the standard t-distribution for several increasing degrees of freedom and compare this to the normal PDF.
- Plot and compare the cumulative distribution functions
- Plot the variance of the t-distribution as a function of degrees of freedom. Compare to the standard normal variance (=1)
- (optional) make a Q-Q plot (see Wiki) and compare the distributions

```

In [ ]: x = np.linspace(-5, 5, 200)
        df_all = [1,2,5,10,30]

        fh, ax = plt.subplots(2,2, figsize=(10,8))

        # PDF
        for df in df_all:
            ...
            ax[0,0].plot(...)

        # CDF
        for df in df_all:
            ...
            ax[1,0].plot(...)
        ax[1,0].plot(...)

        # Variance vs degrees of freedom
        ...
        ax[0,1].set_xlabel('DOF')
        ax[0,1].set_ylabel('Var(T)')

        # Q-Q plot (optional)
        for df in df_all:
            ...
            ax[1,1].plot(...)

        plt.show()

```

### 1.3.2 3.2 Eggs

An egg producer claims to supply eggs with an average egg weight of 63 g. In a box of 12, the following weights were measured (all in g):

62.75, 56.98, 53.30, 62.65, 57.63, 57.23, 56.65, 64.89, 57.87, 60.42, 57.01, 63.65

- Calculate the sample mean and (adjusted -- see ddof option) sample standard deviation.
- What is the probability of obtaining this average weight or lighter, given the supplier's claim?

```

In [ ]: s = [62.75, 56.98, 53.30, 62.65, 57.63, 57.23, 56.65, 64.89, 57.87, 60.42, 57.01, 63.65]
        n_samples = ...
        dof = ... # degrees of freedom
        mu_samp = ... # Sampling mean
        sig_samp = np.std(s, ddof=1)
        mu_claim = ... # Claimed population mean

        ...
        print('Probability of this sample mean ({:.2f}) against claimed mean ({:.2f}): {:.2f} %'

```

```

...))

plt.figure()
...
plt.xlabel('Egg weight (g) (W)')
plt.ylabel('P(W)')

plt.show()

```

- Within what range would 95% of samples follow? And how would this compare with an equivalent normal distribution?
- Plot again the two distributions, marking the 95% intervals

```

In [ ]: ...
        print('Normal: 95% of samples between {} and {}'.format(...))
        print('T-distribution: 95% of samples between {} and {}'.format(...))

plt.figure()
...
plt.xlabel('Egg weight (g) (W)')
plt.ylabel('P(W)')

```

## 1.4 Bonus

A pair of independent, standard normal random variables can be generated by sampling a uniform distribution. One approach to this is the Box-Muller transform (see [https://en.wikipedia.org/wiki/Box%E2%80%93Muller\\_transform](https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform)).

- Generate a long sequence of numbers drawn from  $U(0,1)$
- Use the Box-Muller transform to convert these to normal random variables
- Plot the normal samples on a scatter plot - verify they are not correlated
- Plot the histograms, and superimpose the normal PDF

```

In [ ]: u1 = ...
        u2 = ...
        n1 = ...
        n2 = ...

        ...

```

### 1.4.1 Improbable events

In this example, we tabulate the amplitude deviation against the probability, odds (inverse probability), and equivalent timescale (once in 10 thousand years). Modify the code and try with different distributions - especially those which look similar to the normal distribution, but carry a fatter tail.

```

In [ ]: from IPython.display import display
        import pandas as pd

```

```

def format_days(d):
    if d < 365:
        if d > 90:
            return '{:1.0f} months'.format(d/30)
        elif d > 7:
            return '{:1.0f} weeks'.format(d/7)
        else:
            return '{:1.0f} days'.format(d)
    d /= 365

    if d > 1e9:
        return '{:1.1f} billion years'.format(d*1e-9)
    elif d > 1e6:
        return '{:1.1f} million years'.format(d*1e-6)
    elif d > 1e3:
        return '{:1.1f} millenia'.format(d*1e-3)
    else:
        return '{:1.1f} years'.format(d)

z = np.linspace(0, 10, 500)

data = []
for n in range(1,8):
    p = 2*(1-stats.norm.cdf(n))
    data.append([n, p, 1/p, format_days(1/p)])

display(pd.DataFrame(data, columns=[r'|X| ($\sigma)$', 'p', '1 in', 'time equivalent']))

```

#### 1.4.2 Code to generate "egg" distribution

```

In [ ]: # Generate small dataset for T-dist question
        # True parameters:
        sig = 3
        mu = 58
        n_samples = 12

        s = stats.norm.rvs(size=n_samples, loc=mu, scale=sig)
        print('{:.2f}, '*n_samples).format(*s)[-2])

```