1 W3 Lab-1

Risto Rushford, 04/26/2020

1.1 Introduction

For this first official lab exercise, I will first focus on completing the **Level 1 Difficulty** problems as I continue to build my familiarity with Python

1.1.1 Front Matter Code

[1]:
```
import numpy as np
import scipy.stats as stats
import scipy as sp
import pandas as pd
import math
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# Setting a random seed for reproducibility
np.random.seed(19860604)
```

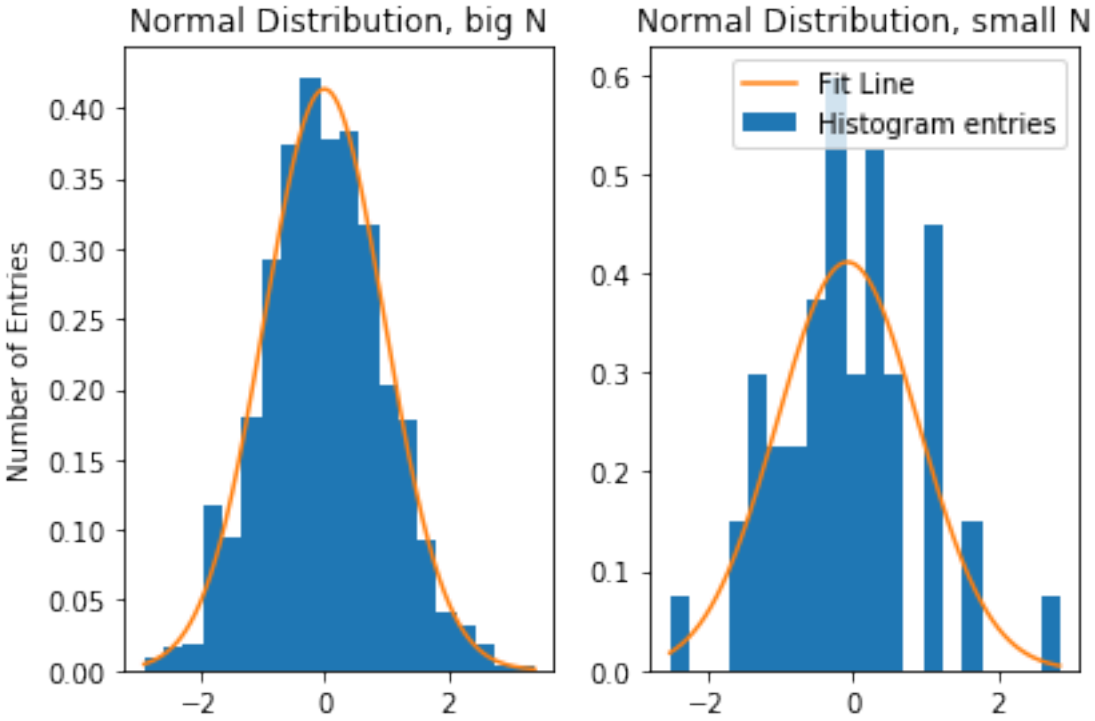
1.2 L1.1 - Random Numbers and Probability Distributions

1.2.1 Normal Distribution Plots

[2]:
```
# First we will create a larger and a smaller sample:
n_large = int(input('Input a number greater than 1000: '))
n_small = int(n_large/20)
n_bins = 20
print('Your large sample size is ' + str(n_large) + ' and your small sample size␣
 ↪is ' + str(n_small) + '.')

# And we generate the large and small distributions, with fit
# information for the lineplots
xl = np.random.normal(size=n_large)
xs = np.random.normal(size=n_small)

xl_fit = np.arange(min(xl), max(xl), 1/(n_large))
yl_fit = stats.norm.pdf(xl_fit, np.mean(xl), np.std(xl))

xs_fit = np.arange(min(xs), max(xs), 1/(n_small))
ys_fit = stats.norm.pdf(xs_fit, np.mean(xs), np.std(xs))

# Now lets plot both distributions side-by-side using subplots:
fig, axs = plt.subplots(1, 2, tight_layout=True)

axs[0].hist(xl, density=True, bins=n_bins)
```

```
axs[0].plot(xl_fit,yl_fit)
axs[0].set_title('Normal Distribution, big N')
axs[0].set_ylabel('Number of Entries')

axs[1].hist(xs, density=True, bins=n_bins, label=r'Histogram entries')
axs[1].plot(xs_fit,ys_fit, label=r'Fit Line')
axs[1].set_title('Normal Distribution, small N')
axs[1].legend(loc='best')

plt.show()
```

```
Input a number greater than 1000:  1001

Your large sample size is 1001 and your small sample size is 50.
```



The first thing that I did was copy/paste the code from the PDF for this week's lab and then start thinking about how I could "make the code mine". I chose to use an alternate plot library from what was selected so as to challenge myself a little bit more.

I found the tutorial at https://realpython.com/python-matplotlib-guide/ to be a decent resource for using matplotlib and used it to learn how to generate the two histogram plots side-by-side. The two main things that kept giving me trouble were that I had to figure out what made the line plot actually fit the normal distribution. Once I actually analyzed the definition of the x_fit and y_fit variables it became obvious.

1.2.2 Exponential Distribution

[3]:
```
# I had to look up how to do much of this with a tutorial, see notes below

def fit_function(x, A, beta, B, mu, sigma):
    return (A * np.exp(-x/beta) + B * np.exp(-1.0 * (x - mu)**2 / (2 *␣
 ↪sigma**2)))

# Generating the exponential data sets and binning the data
bins = np.linspace(0, 6, 61)

exp_large = np.random.exponential(size = n_large)
exp_ldata, bins_1 = np.histogram(exp_large, bins=bins)
l_bincenters = np.array([0.5 * (bins[i] + bins[i+1]) for i in␣
 ↪range(len(bins)-1)])

# Fitting the functions to the histogram data
l_popt, l_pcov = curve_fit(fit_function, xdata=l_bincenters, ydata=exp_ldata,␣
 ↪p0=[20000, 2.0, 2000, 3.0, 0.3])


exp_small = np.random.exponential(size = n_small)
exp_sdata, bins_2 = np.histogram(exp_small, bins=bins)
s_bincenters = np.array([0.5 * (bins[i] + bins[i+1]) for i in␣
 ↪range(len(bins)-1)])

s_popt, s_pcov = curve_fit(fit_function, xdata=s_bincenters, ydata=exp_sdata,␣
 ↪p0=[20000, 2.0, 2000, 3.0, 0.3])


# Generate enough x values to make the curves look smooth.
xspace = np.linspace(0, 6, 100000)

# Plot for N large
fig, axs = plt.pyplot.subplots(1, 2, sharey=True)

axs[0].bar(l_bincenters, exp_ldata, width=bins[1] - bins[0], label=r'Histogram␣
 ↪entries')
axs[0].plot(xspace, fit_function(xspace, *l_popt), color='orange', linewidth=2.
 ↪5, label=r'Fitted function')

# Make the plot nicer.
axs[0].set_ylabel(r'Number of Entries')
axs[0].set_title(r'Exponential Distribution')


# And finally plot for N small
```

```
fig, axs = plt.subplot(1, 2, 2)
axs[1].bar(s_bincenters, exp_sdata, width=bins[1] - bins[0], label=r'Histogram␣
 ↪entries')
axs[1].plot(xspace, fit_function(xspace, *s_popt), color='orange', linewidth=2.
 ↪5, label=r'Fitted function')
axs[1].set_title(r'Exponential Distribution')
axs[1].legend(loc='best')
```

```
---------------------------------------------------------------------------

RuntimeError                              Traceback (most recent call last)

<ipython-input-3-cadd049a45d9> in <module>
 12
 13 # Fitting the functions to the histogram data
---> 14 l_popt, l_pcov = curve_fit(fit_function, xdata=l_bincenters,␣
↪ydata=exp_ldata, p0=[20000, 2.0, 2000, 3.0, 0.3])
 15
 16


~\Anaconda3\lib\site-packages\scipy\optimize\minpack.py in curve_fit(f,␣
↪xdata, ydata, p0, sigma, absolute_sigma, check_finite, bounds, method, jac,␣
↪**kwargs)
    754         cost = np.sum(infodict['fvec'] ** 2)
    755         if ier not in [1, 2, 3, 4]:
--> 756             raise RuntimeError("Optimal parameters not found: " +␣
↪errmsg)
    757     else:
    758         # Rename maxfev (leastsq) to max_nfev (least_squares), if␣
↪specified.


RuntimeError: Optimal parameters not found: Number of calls to function␣
↪has reached maxfev = 1200.
```

Trying to plot the exponential distribution was more challenging than I expected after figuring out how to plot the normal distribution above. Getting the histogram was rather easy, I simply changed the nump.random parameter from .normal to .exponential. Fitting the curve, however was more challenging. I couldn't find any tutorials online which could give me a nice explanation that would enable me to simply adapt the previous code. I ended up finding this tutorial at https://riptutorial.com/scipy/example/31081/fitting-a-function-to-data-from-a-histogram which helped me for most of the exercise except to create a side-by-side plot as I was able to do for the normal distribution.

1.3 L1.2 - Approximating the Binomial Distribution

[4]:
```
Total_Flips = 10

Heads = sp.linspace(0, Total_Flips, Total_Flips+1)
Probability = sp.stats.binom.pmf(Heads, Total_Flips, 0.5)
df_fair = pd.DataFrame(data=Probability, columns=["Probability"])
df_fair.index.names = ["Heads"]

print(df_fair)
print('The probability of getting heads 3 out of ten coin flips with a fair coin␣
 ↪is ' + str(df_fair.iat[3,0]))
```

```
       Probability
Heads
0         0.000977
1         0.009766
2         0.043945
3         0.117188
4         0.205078
5         0.246094
6         0.205078
7         0.117188
8         0.043945
9         0.009766
10        0.000977
The probability of getting heads 3 out of ten coin flips with a fair coin is
0.11718750000000014
```

[5]:
```
Total_Flips = 10

Heads = sp.linspace(0, Total_Flips, Total_Flips+1)
Probability = sp.stats.binom.pmf(Heads, Total_Flips, 0.65)
df_unfair = pd.DataFrame(data=Probability, columns=["Probability"])
df_unfair.index.names = ["Heads"]

print(df_unfair)
print('The probability of getting heads 3 out of 10 coin flips with an unfair␣
 ↪coin is ' + str(df_unfair.iat[3,0]))
```

```
       Probability
Heads
0         0.000028
1         0.000512
2         0.004281
3         0.021203
4         0.068910
5         0.153570
```

```
6          0.237668
7          0.252220
8          0.175653
9          0.072492
10         0.013463
The probability of getting heads 3 out of 10 coin flips with an unfair coin is
0.02120301528515624
```

For a while this problem confused me a little because I am still unfamiliar with Monte Carlo methods AND with some of the functions used. I went back to thinking about Monte Carlo and then came back to this problem and took a careful look at what had been provided and realized that in the variable "Heads", an array to represent each number of coin flips. Then "Probability" takes the array and the number of flips as inputs with the probability per turn and computes the probability of occurences for each value in the array up to the total number of coin flips.

1.4 L1.4 Estimating Pi via Monte Carlo Methods

[6]:
```
# First we'll prepare the variables for number of iterations and
# number of points within the unit circle
num_runs = int(input('How many iterations will we try? '))

# And to track the xy coordinates of those points for plotting:
x = 0
y = 0

x_in = []
y_in = []
x_out = []
y_out = []

# And for sampling for errors
sample_numbers = np.logspace(1, 6, num=100)
Pi = np.pi

# Now we'll run the Monte Carlo
def estimate_pi(num_runs):
    hits = 0
    for i in range(num_runs):
        x, y = np.random.uniform(0, 1, size=2)

        if x**2 + y**2 <= 1:
            hits += 1
            x_in.append(x)
            y_in.append(y)
        else:
            x_out.append(x)
            y_out.append(y)
        return 4*hits/num_runs
```

```
pi_estimate = estimate_pi(num_runs)
print('Pi is estimated at ' + str(pi_estimate))

# And we'll plot the results
fig, ax = plt.subplots()
ax.set_aspect('equal')
ax.scatter(x_in, y_in, color='g', marker='.')
ax.scatter(x_out, y_out, color='r', marker='.')
fig.show()

# Now we'll look at the error
approximations = [estimate_pi(int(i))
                  for i in sample_numbers]
error = [np.abs(i-Pi)/Pi
         for i in approximations]
plt.plot(error)
```

```
How many iterations will we try?  10000

Pi is estimated at 0.0004

C:\Users\risto\Anaconda3\lib\site-packages\ipykernel_launcher.py:41:
UserWarning: Matplotlib is currently using
module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot
show the figure.
```

[6]:
```
[<matplotlib.lines.Line2D at 0x1e7a1394108>]
```

I've come to an impasse for now on this Monte Carlo simulation. To provide the error estimates, I realized that I would need to have it set as a function to do it effectively, but I can't seem to make it work as a function just yet.

See below for code that performs the simulation without defining a function, and I look forward to any insights that will help me to proceed.

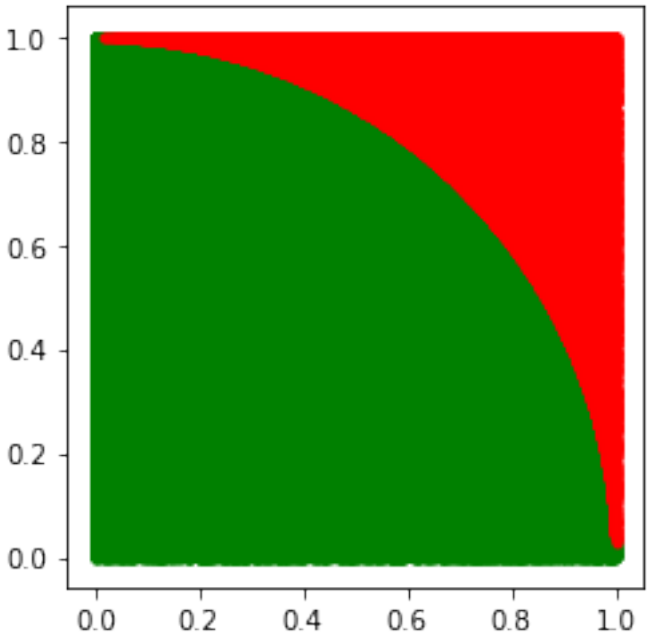
[8]:
```
# First we'll prepare the variables for number of iterations and
# number of points within the unit circle
num_runs = int(input('How many iterations will we try? '))
hits = 0
Pi = np.pi

# And to track the xy coordinates of those points for plotting:
```

```
x_in = []
y_in = []
x_out = []
y_out = []

# Now we'll run the Monte Carlo
for i in range(num_runs):
    x, y = np.random.uniform(0, 1, size=2)
    r = (x**2 + y**2)**.5

    if x**2 + y**2 <= 1:
        hits += 1
        x_in.append(x)
        y_in.append(y)
    else:
        x_out.append(x)
        y_out.append(y)

pi_estimate = 4*hits/num_runs
print('Pi is estimated at ' + str(pi_estimate))
print('The estimated error is ' + str(100*abs(Pi-pi_estimate)/Pi) + '%')

# Prepping the circle to superimpose on the plot
t = np.arange(Pi, -Pi, 0.1)
x_circ = np.sin(t)
y_circ = np.cos(t)

# And we'll plot the results
fig, ax = plt.subplots(1,1)
ax.set_aspect('equal')
ax.scatter(x_in, y_in, color='g', marker='.')
ax.scatter(x_out, y_out, color='r', marker='.')
ax.plot(x_circ, y_circ)
```

```
How many iterations will we try?  1000000
```

```
Pi is estimated at 3.139712
The estimated error is 0.05986306301182362%
```

[8]: [<matplotlib.lines.Line2D at 0x1e7a5393b48>]



I also can't seem to get the circle to stack on top of the scatterplot of the circle, however I've fulfilled the parameters of this assignment and so I will submit this for now.