Homework 3

Intro to Statistics - Part 2

This notebook is arranged in cells. Texts are usually written in the markdown cells, and here you can use html tags (make it bold, italic, colored, etc). You can double click on this cell to see the formatting.

The ellipsis (...) are provided where you are expected to write your solution but feel free to change the template (not over much) in case this style is not to your taste.

Hit "Shift-Enter" on a code cell to evaluate it. Double click a Markdown cell to edit.

Link Okpy

In [ ]:

```
from client.api.notebook import Notebook
ok = Notebook('hw3.ok')
_ = ok.auth(inline = True)
```

Imports

In [ ]:

```
import numpy as np
from scipy.integrate import quad
#For plotting
import matplotlib.pyplot as plt
%matplotlib inline
```

Problem 1 - Fitting Gaussian Contours to a 2D Data

Gaussian distribution function plays a central role in statistics and is the most ubiquitous distribution in physics. It often provides a good approximation to the true probability density function (pdf) even in cases where its application is not strictly correct.

In this problem, suppose that you have measured 1000 pairs of values $(x_1, y_1), \ldots, (x_{1000}, y_{1000})$ of two variables x, y. You saved these measurements to a .dat file ("Problem1_data.dat"). Plot their 1-dimensional pdf's and determine how well Gaussian pdf can approximate them. Compute the mean, variance, median, mode, 68% and 95% confidence intervals, and correlation coefficient.

1. Plot 1-dimensional pdf for x.

In [ ]:

```
# Load a given 2D data
data = np.loadtxt("Problem1_data.dat")
x = data[:,0]
y = data[:,1]
```

In [ ]:

```
# Plot a normalized histogram (Hint - https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.hist.html)
# Try to have 25 elements per each bin

...
```

2. Calculate mean, variance, and median of x. First, do it "by hand" without using any in-built functions. Then, check your answers using in-built functions from numpy.

In [ ]:

```
# Calculating "by hand"
mean_x = ...
variance_x = ...

median_x = ...

print("For x, mean = ", mean_x, ", variance = ", variance_x, ", and median = ", median_x)

# Using in-built functions from numpy
mean_x = ...
variance_x = ...
median_x = ...

print("For x, mean = ", mean_x, ", variance = ", variance_x, ", and median = ", median_x)
```

3. Smoothly interpolate the discrete probability density from Part 1. Then, find the mode and symmetric 68%, 95% confidence intervals. (Suggestion - Read https://docs.scipy.org/doc/numpy/reference/generated/numpy.histogram.html and https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html)

(Hint - For 68% confidence interval, find the range where 68% of the given sample occurs. We assume that such interval is symmetrically placed around the mean.
In other words, find a such taht

$$0.68 = \int_{\mu-a}^{\mu+a} P(x)$$

where $P(x)$ is x's pdf, and μ is the mean.
One way to find a is to define a cumulative distribution function (cdf) $G(x)$ and find a such that $G(\mu + a) - G(\mu - a) = .68$.)

In [ ]:

```
# NOTE: The following skeleton code is only a suggestion.
from scipy.interpolate import UnivariateSpline

p, xvar = np.histogram( ... , normed = True)
# Find x values at the center of each bin (call it xvar)
xvar = xvar[:-1] + (xvar[1] - xvar[0])/2

# Find the interpolated function f
f = UnivariateSpline(xvar, p, s=n)
# Change the amount of smoothing
f.set_smoothing_factor(1.e-8)

# Plot both histogram and interpolated function
plt.hist( ... , normed = True, rwidth = 0.8)
plt.plot( ... , '--', label = "Interpolated")
...
plt.show()
```

In [ ]:

```
# Find the mode
mode_x = ...

print("For x, mode = ", mode_x)

# Find 68% and 95% confidence intervals

...
```

Assuming that the distribution is Gaussian, 68% and 95% confidence interval corresponds to $\mu \pm 1\sigma$ and $\mu \pm 2\sigma$. </i>

In [ ]:

```
print("Assuming Gaussian distribution, 68% confidence interval is", mean_x, "±",
 np.sqrt(variance_x),
     ", and 95% interval is", mean_x, "±", 2*np.sqrt(variance_x))
```

You should find that the Gaussian distribution is a reasonable approximation in this case.

4. Plot Gaussian distribution with the mean and variance from Part 1 on top of probability density histogram. Make sure to label each plot.

In [ ]:

```
# Define Gaussian distribution
def gaussian(x, mu, sigma):
    return ...
```

In [ ]:

```
# Plot histogram
...
# Plot Gaussian distribution on top
...
```

5. Repeat part 1-4 for y.

In [ ]:

```
# Find mean, variance and median
...
# Plot histogram (discrete pdf) and interpolate it
...
# Find mode and 68%, 95% confidence interval
...
# Plot Gaussian fit on top of the histogram
...
```

6. Make a 2-d scatter plot with Gaussian contours (ellipses). Then, compute the covariance (C_{xy}) of x and y as well as the correlation coefficient $\rho = \frac{C_{xy}}{\sigma_x \sigma_y}$.

In [ ]:

```
# Compute the covariance and correlation coefficient
cov = ...

print("The covariance between x and y is", cov)

print("The correlation coefficient of x and y is", ...)

# Make a 2-d scatter plot with Gaussian contours

import matplotlib.mlab as mlab

# Create coordinate matrices from coordinate vectors.
gridx = np.linspace(9000, 11000, 100)
gridy = np.linspace(-1500, 1500, 100)
X, Y = np.meshgrid(gridx, gridy)

# Create bivariate Gaussian distribution for equal shape X, Y (https://matplotli
b.org/api/mlab_api.html)
Z = mlab.bivariate_normal(X, Y, np.sqrt(variance_x), np.sqrt(variance_y), mean_x
, mean_y, cov)

# Make plot
plt.figure(figsize = (10, 7))
# Scatter plot
plt.scatter(x, y, marker = 'x')
# Gaussian contour plots
plt.contour(X, Y, Z)

plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

The above contour plot is a bird eye view of the 3-d mesh plot; these are ellipses of equal probability. The coloring represents the intensity. Yellow central ellipse is the region of highest probability; the peak of 2-d Gaussian distribution is at the center of this ellipse. As we move away from the peak, the probability lowers.

Problem 2 - Central Limit Theorem

Plot the binomial distribution $P(N_A, N)$ for different values of N and plot the Gaussian with mean and variance for the binomial. Similarly, plot the Poisson distribution with the mean varying from 1 to 10. See if both binomial and Poisson approach Gaussian as the mean/N increases.

(Reference - Kardar p. 41) For the binomial distribution, consider a random variable with two outcomes A and B of relative probabilities p_A and $p_B = 1 - p_A$. The probability that in N trials the event A occurs exactly N_A times is given by the binomial distribution:

$$p_N(N_A) = \binom{N}{N_A} p_A^{N_A} (1 - p_A)^{N - N_A}.$$

1. Plot the binomial distribution $P(N_A, N)$ for $N = 5, 20, 40, 100, 300$ and plot the Gaussian with mean and variance for the binomial. Let $p_A = 0.5$ and 0.1. Make sure to label each plot.

In [ ]:

```
# Import packages for the bionomial coefficient
from scipy.special import binom

# Define the probability for the binomial distribution
def pdf_binom(p_A, N, N_A):
    return ...

# Define Gaussian distribution
def gaussian(x, mu, sigma):
    return ...
```

In [ ]:

```
N = [5, 20, 40, 100, 300]

# Make plot

# For p_A = 0.1
N_A = np.linspace(0, 40, 1000)
plt.figure(figsize= (10, 8))
p_A = 0.1
...
plt.show()

# For p_A = 0.5
N_A = np.linspace(0, 70, 1000)
plt.figure(figsize= (10, 8))
p_A = 0.5
...
plt.show()
```

In class, we find that the binomial distribution is approximately normal (with mean Np_A and variance $Np_A(1-p_A)$) as $N \to \infty$, by the central limit theorem. The proof of this theorem can be carried out using Stirling's approximation:

$$N! \approx N^N e^{-N} \sqrt{2\pi N}$$

2. Plot the above Stirling's formula approximation (i.e. Compare $N!$ with Stirling's approximation. Compute the residual: (actual-estimate)/actual.)
(Hint: $\Gamma(n+1) = n!$)

In [ ]:

```
from scipy.special import gamma

Nvals = np.linspace(1, 40, 1000)

actual = ...
estimate = ...

plt.plot(Nvals, (actual-estimate)/actual)
plt.xlabel('$N$')
plt.ylabel('residual')
plt.show()
```

You should find that residual $\to 0$ as $N \to \infty$.

Next, consider the Poisson distribution (Kardar p. 42):

$$P(\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

where k is the number of occurrences. Its mean and variance are λ.

3. Plot $P(\lambda)$ as a function of k for $\lambda = 1, 3, 5, 10, 20$ and plot the Gaussian with mean and variance for the Poisson. Make sure to label.

In [ ]:

```
# Define the Poisson distribution
...

# Make plot
...
```

4. What happens as the mean/N increases?

Answer:
...

Problem 3 - Fitting Data to a Straight Line (Linear Regression)

(Reference - NR 15.2) We fit a set of 50 data points (x_i, y_i) to a straight-line model $y(x) = a + bx$. The uncertainty σ_i associated with each measurement y_i is known, and we assume that the x_i's are known exactly. To measure how well the model agrees with the data, we use the chi-square merti function:

$$\chi^2(a, b) = \sum_{i=0}^{N-1} \left(\frac{y_i - a - bx_i}{\sigma_i} \right)^2.$$

Make a scatter plot of data (including uncertainties) and find the best-fit line. Compute the errors on the two parameters a and b and plot lines where the two are changed by $\pm 1\sigma$.

1. Plot data (make sure to include error bars). (Hint - https://matplotlib.org/api/_as_gen/matplotlib.axes.Axes.errorbar.html)

In [ ]:

```
# Load a given 2D data
data = np.loadtxt("Problem3_data.dat")
x = data[:,0]
y = data[:,1]
sig_y = data[:,2]
```

In [ ]:

```
# Make plot
plt.figure(figsize = (10, 7))
# Scatter plot
...
plt.show()
```

(NR p. 781) We should minimize the above chi-square function to determine a and b. At its minimum, derivatives of χ^2 with respect to a, b vanish:

$$\frac{\partial \chi^2}{\partial a} = -2 \sum \frac{y_i - a - bx_i}{\sigma_i^2} = 0 \qquad (1)$$

$$\frac{\partial \chi^2}{\partial b} = -2 \sum \frac{x_i(y_i - a - bx_i)}{\sigma_i^2} = 0 \qquad (2)$$

These conditions can be rewritten in a convenient form if we define the following sums:

$$S = \sum \frac{1}{\sigma_i^2},\ S_x = \sum \frac{x_i}{\sigma_i^2},\ S_y = \sum \frac{y_i}{\sigma_i^2}$$

$$S_{xx} = \sum \frac{x_i^2}{\sigma_i^2},\ S_{xy} = \sum \frac{x_i y_i}{\sigma_i^2}$$

With these, we can rewrite (1), (2) as:

$$a * S + b * S_x = S_y$$
$$a * S_x + b * S_{xx} = S_{xy}$$

The solution to these is calculated as:

$$\Delta = SS_{xx} - (S_x)^2$$

$$a = \frac{S_{xx}S_y - S_x S_{xy}}{\Delta}$$
$$b = \frac{SS_{xy} - S_x S_y}{\Delta}$$

2. Find parameters a, b which minimize the chi-square function and plot the best-fit line on top of the data.

In [ ]:

```
S = ...
Sx = ...
Sy = ...
Sxx = ...
Sxy = ...
Delta = ...
```

In [ ]:

```
a = ...
b = ...

...
```

In [ ]:

```
# Make plot
plt.figure(figsize = (10, 7))
...
plt.show()
```

Now, we must estimate the probable uncertainties in the estimates of a and b, since obviously the measurement errors in the data must introduce some uncertainty in the determination of those parameters. If the data are independent, then each contributes its own bit of uncertainty to the parameters. Consideration of propagation of errors show that the variance σ_f^2 in the value of any function will be

$$\sigma_f^2 = \sum \sigma_i^2 (\frac{\partial f}{\partial y_i})^2$$

For the straight line, the derivatives of a and b with respect to y_i can be directly evaluated from teh solution:

$$\frac{\partial a}{\partial y_i} = \frac{S_{xx} - S_x x_i}{\sigma_i^2 \Delta}$$
$$\frac{\partial b}{\partial y_i} = \frac{S x_i - S_x}{\sigma_i^2 \Delta}$$

Summing over the points, we get

$$\sigma_a^2 = S_{xx}/\Delta$$
$$\sigma_b^2 = S/\Delta$$

3. Compute the errors (σ_a, σ_b) on the two parameters a, b and plot lines where the two are changed by $\pm 1\sigma$. (Hint - Try to plot the 1σ confidece band as in http://astropython.blogspot.com/2011/12/ (http://astropython.blogspot.com/2011/12/). You can use plt.fill_between to shade the region between plots.)

In [ ]:

```
# Calculate sigma_a, sigma_b

sigma_a = ...
sigma_b = ...

print('We estimate that a =', a ,"±", sigma_a, "and b =", b, "±", sigma_b)
```

In [ ]:

```
plt.figure(figsize = (10, 7))
...
plt.show()
```

To Submit

Execute the following cell to submit. If you make changes, execute the cell again to resubmit the final copy of the notebook, they do not get updated automatically.
We recommend that all the above cells should be executed (their output visible) in the notebook at the time of submission.
Only the final submission before the deadline will be graded.

In [ ]:

```
_ = ok.submit()
```