Assignment 1 - Probability, Linear Algebra, Programming, and Git

Anna Berman

Netid: *aeb100*

Probability and Statistics Theory

1

Let $f(x) = \begin{cases} 0 & x < 0 \\ \alpha x^2 & 0 \leq x \leq 2 \\ 0 & 2 < x \end{cases}$

For what value of α is $f(x)$ a valid probability density function?

Note: for all assignments, write out all equations and math for all assignments using markdown and LaTeX (https://tobi.oetiker.ch/lshort/lshort.pdf) and show all work

$\int_0^2 \alpha x^2 dx = \alpha \int_0^2 x^2 dx$

$= \alpha \left| \frac{x^3}{3} \right|_0^2$

$= \alpha(\frac{2^3}{3} - \frac{0^3}{3})$

$= \alpha \frac{8}{3} = 1$

$\alpha = \frac{3}{8}$

2

What is the cumulative distribution function (CDF) that corresponds to the following probability distribution function? Please state the value of the CDF for all possible values of x.

$f(x) = \begin{cases} \frac{1}{3} & 0 < x < 3 \\ 0 & \text{otherwise} \end{cases}$

$$
\begin{aligned}
f(x) &= \int_{\alpha}^{x} \frac{1}{b-a} dw \\
&= \int_{0}^{x} \frac{1}{3} dw \\
&= \left| \frac{w}{3} \right|_{0}^{x} \\
&= \frac{x}{3}
\end{aligned}
$$

CDF: $f(x) = \begin{cases} 0 & x \leq 0 \\ \frac{x}{3} & 0 < x < 3 \\ 1 & 3 \leq x \end{cases}$

3

For the probability distribution function for the random variable X,

$$f(x) = \begin{cases} \frac{1}{3} & 0 < x < 3 \\ 0 & \text{otherwise} \end{cases}$$

what is the (a) expected value and (b) variance of X. *Show all work*.

(a) Expected value:

$$E(x) = \int_{x_0}^{x_1} x \cdot f(x)dx$$
$$= \int_0^3 x \cdot \frac{1}{3}dx$$
$$= \left| \frac{x^2}{6} \right|_0^3$$
$$= \frac{9}{6} - 0$$
$$E(x) = \frac{3}{2}$$

(b) Variance:

$$Var(x) = \int_{x_0}^{x_1} x^2 \cdot f(x)dx - \left(\int_{x_0}^{x_1} x \cdot f(x)dx \right)^2$$
$$= \int_0^3 \frac{x^2}{3}dx - \left(\frac{3}{2} \right)^2$$
$$= \left| \frac{x^3}{9} \right|_0^3 - \frac{9}{4}$$
$$= \frac{27}{9} - \frac{9}{4}$$
$$= \frac{12}{4} - \frac{9}{4}$$
$$Var(x) = \frac{3}{4}$$

4

Consider the following table of data that provides the values of a discrete data vector $\mathbf{x}$ of samples from the random variable X, where each entry in $\mathbf{x}$ is given as x_i.

Table 1. Dataset N=5 observations

	x_0	x_1	x_2	x_3	x_4
$\mathbf{x}$	2	3	10	-1	-1

What is the (a) mean, (b) variance, and the of the data?

Show all work. Your answer should include the definition of mean, median, and variance in the context of discrete data.

(a) Mean

$$\mu = \frac{\Sigma x_i}{n}$$
$$\mu = \frac{2 + 3 + 10 - 1 - 1}{5}$$
$$\mu = \frac{13}{5} = 2.6$$

(b) Variance

$$\sigma^2 = \frac{\Sigma(x - \mu)^2}{n - 1}$$
$$= \frac{(2 - \frac{13}{5})^2 + (3 - \frac{13}{5})^2 + (10 - \frac{13}{5})^2 + (-1 - \frac{13}{5})^2 + (-1 - \frac{13}{4})^2}{5}$$
$$= \frac{(-\frac{3}{5})^2 + (\frac{2}{5})^2 + (\frac{37}{5})^2 + (-\frac{18}{5})^2 + (-\frac{18}{5})^2}{4}$$
$$= \frac{\frac{9}{25} + \frac{4}{25} + \frac{1369}{25} + \frac{324}{25} + \frac{324}{25}}{4}$$
$$= \frac{\frac{2030}{25}}{4}$$
$$= \frac{2030}{100}$$
$$\sigma^2 = 20.3$$

5

Review of counting from probability theory.

(a) How many different 7-place license plates are possible if the first 3 places only contain letters and the last 4 only contain numbers?

(b) How many different batting orders are possible for a baseball team with 9 players?

(c) How many batting orders of 5 players are possible for a team with 9 players total?

(d) Let's assume this class has 26 students and we want to form project teams. How many unique teams of 3 are possible?

Hint: For each problem, determine if order matters, and if it should be calculated with or without replacement.

(a)

$$26 \cdot 26 \cdot 26 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 175,760,000$$

(b)

$$9! = 362,880$$

(c)

$$\frac{\frac{n!}{(n-k)!}}{\frac{9!}{(9-5)!}}$$
$$\frac{\frac{9!}{(4)!}}{9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 = 15,120}$$

(d)

$$\frac{\frac{\frac{n!}{k!(n-k)!}}{\frac{26!}{3!(26-3)!}}}{\frac{26!}{3!(26-3)!}}$$
$$\frac{\frac{26 \cdot 25 \cdot 24}{3 \cdot 2}}{2,600}$$

Linear Algebra

6

Matrix manipulations and multiplication. Machine learning involves working with many matrices, so this exercise will provide you with the opportunity to practice those skills.

Let $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix}$, $\mathbf{c} = \begin{bmatrix} 4 \\ -3 \\ 6 \end{bmatrix}$, and $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Compute the following or indicate that it cannot be computed:

1. $\mathbf{AA}$
2. $\mathbf{AA}^T$
3. $\mathbf{Ab}$
4. $\mathbf{Ab}^T$
5. $\mathbf{bA}$
6. $\mathbf{b}^T\mathbf{A}$
7. $\mathbf{bb}$
8. $\mathbf{b}^T\mathbf{b}$
9. $\mathbf{bb}^T$
10. $\mathbf{b} + \mathbf{c}^T$
11. $\mathbf{b}^T\mathbf{b}^T$
12. $\mathbf{A}^{-1}\mathbf{b}$
13. $\mathbf{A} \circ \mathbf{A}$
14. $\mathbf{b} \circ \mathbf{c}$

Note: The element-wise (or Hadamard) product is the product of each element in one matrix with the corresponding element in another matrix, and is represented by the symbol "$\circ$".

1. $\mathbf{AA}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1+4+9 & 2+8+15 & 3+10+18 \\ 2+8+15 & 4+16+25 & 6+20+30 \\ 3+10+18 & 6+20+30 & 9+25+36 \end{bmatrix}$$

$$\begin{bmatrix} 14 & 25 & 31 \\ 25 & 45 & 56 \\ 31 & 56 & 70 \end{bmatrix}$$

1. $\mathbf{AA}^T$

$$\mathbf{AA}^T = \mathbf{AA} = \begin{bmatrix} 14 & 25 & 31 \\ 25 & 45 & 56 \\ 31 & 56 & 70 \end{bmatrix}$$

1. $\mathbf{Ab}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}\begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix}$$

$$\begin{bmatrix} -1+6+24 \\ -2+12+40 \\ -1+15+48 \end{bmatrix}$$

$$\begin{bmatrix} 29 \\ 50 \\ 60 \end{bmatrix}$$

1. $\mathbf{Ab}^T$ Cannot be commuted
2. $\mathbf{bA}$ Cannot be commuted
3. $\mathbf{b}^T\mathbf{A}$

$$\begin{bmatrix} -1 & 3 & 8 \end{bmatrix}\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} -1+6+24 & -2+12+40 & -1+15+48 \end{bmatrix}$$

$$\begin{bmatrix} 29 & 50 & 60 \end{bmatrix}$$

1. $\mathbf{bb}$ Cannot be commuted
2. $\mathbf{b}^T\mathbf{b}$

$$\begin{bmatrix} -1 & 3 & 8 \end{bmatrix} \begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 + 9 + 64 \end{bmatrix}$$

$$\begin{bmatrix} 74 \end{bmatrix}$$

1. $\mathbf{b}^T\mathbf{b}$

$$\begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix} \begin{bmatrix} -1 & 3 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -3 & -8 \\ -3 & 9 & 24 \\ -8 & 24 & 64 \end{bmatrix}$$

2. $\mathbf{b} + \mathbf{c}^T$ Cannot be commuted
3. $\mathbf{b}^T\mathbf{b}^T$ Cannot be commuted
4. $\mathbf{A}^{-1}\mathbf{b}$

$$\mathbf{A}^{-1} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 1 & 0 & 0 \\ 2 & 4 & 5 & 0 & 1 & 0 \\ 3 & 5 & 6 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 0 & -1 & -2 & 1 & 0 \\ 0 & -1 & -3 & -3 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 3 & 3 & 0 & -1 \\ 0 & 0 & 1 & 2 & -1 & 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 2 & 0 & -5 & 3 & 0 \\ 0 & 1 & 0 & -3 & 3 & -1 \\ 0 & 0 & 1 & 2 & -1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 1 & -3 & 2 \\ 0 & 1 & 0 & -3 & 3 & -1 \\ 0 & 0 & 1 & 2 & -1 & 0 \end{bmatrix}$$

$$\mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} 1 & -3 & 2 \\ -3 & 3 & -1 \\ 2 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix}$$

$$\begin{bmatrix} -1 - 9 + 16 \\ 3 + 9 - 8 \\ -2 - 3 + 0 \end{bmatrix}$$

$$\begin{bmatrix} 6 \\ 4 \\ -5 \end{bmatrix}$$

1. $\mathbf{A} \circ \mathbf{A}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix} \circ \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 & 9 \\ 4 & 16 & 25 \\ 9 & 25 & 36 \end{bmatrix}$$

2. $\mathbf{b} \circ \mathbf{c}$

$$\begin{bmatrix} -1 \\ 3 \\ 8 \end{bmatrix} \circ \begin{bmatrix} 4 \\ -3 \\ 6 \end{bmatrix}$$

$$\begin{bmatrix} -4 \\ -9 \\ 48 \end{bmatrix}$$

In [34]:
```
import numpy as np

A = np.matrix('1,2,3; 2,4,5; 3,5,6')
b = np.matrix('-1; 3; 8')
c = np.matrix('4; -3; 6')
I = np.matrix('1,0,0; 0,1,0; 0,0,1')

# 1 AA
print('1 :', np.matmul(A,A))

# 2 AA^T
print('2 :', np.matmul(A,A.transpose()))

# 3 Ab
print('3 :', np.matmul(A,b))

# 4 Ab^T
print('4 : Cannot be commuted')

# 5 bA
print('5 : Cannot be commuted')

# 6 b^TA
print('6 :', np.matmul(b.transpose(),A))

# 7 bb
print('7 : Cannot be commuted')

# 8 b^Tb
print('8 :', np.matmul(b.transpose(), b))

# 9 bb^T
print('9 :', np.matmul(b, b.transpose()))

# 10 b + c^T
print('10 : Cannot be communted')

# 11 b^Tb^T
print('11 : Cannot be communted')

# 12 A^1b
print('12 :', np.matmul(A.I, b))

# 13 A*A
print('13: ', np.array(A)*np.array(A))

# 13 b*c
print('13: ', np.array(b)*np.array(c))
```

```
1 : [[14 25 31]
 [25 45 56]
 [31 56 70]]
2 : [[14 25 31]
 [25 45 56]
 [31 56 70]]
3 : [[29]
 [50]
 [60]]
4 : Cannot be commuted
5 : Cannot be commuted
6 : [[29 50 60]]
7 : Cannot be commuted
8 : [[74]]
9 : [[ 1 -3 -8]
 [-3  9 24]
 [-8 24 64]]
10 : Cannot be communted
11 : Cannot be communted
12 : [[ 6.]
 [ 4.]
 [-5.]]
13:  [[ 1  4  9]
 [ 4 16 25]
 [ 9 25 36]]
13:  [[-4]
 [-9]
 [48]]
```

6

Eigenvectors and eigenvalues. Eigenvectors and eigenvalues are useful for some machine learning algorithms, but the concepts take time to solidly grasp. For an intuitive review of these concepts, explore this interactive website at Setosa.io (http://setosa.io/ev/eigenvectors-and-eigenvalues/). Also, the series of linear algebra videos by Grant Sanderson of 3Brown1Blue are excellent and can be viewed on youtube here (https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab).

1. Calculate the eigenvalues and corresponding eigenvectors of matrix $\mathbf{A}$ above, from the last question.
2. Choose one of the eigenvector/eigenvalue pairs, $\mathbf{v}$ and λ, and show that $\mathbf{Av} = \lambda\mathbf{v}$. Also show that this relationship extends to higher orders: $\mathbf{AAv} = \lambda^2\mathbf{v}$
3. Show that the eigenvectors are orthogonal to one another (e.g. their inner product is zero). This is true for real, symmetric matrices.

In [42]:
```
import numpy as np

# Define A
#A = np.matrix('1,2,3; 2,4,5; 3,5,6')
A = np.array([[1,2,3], [2,4,5], [3,5,6]])

# Find the eigenvalues and corresponding eigenvectors
eigVals, eigVecs = np.linalg.eig(A)

# Select first eigenpair
lamd = eigVals[0]
v = eigVecs[:,0]

# Show that Av = λv
side1 = np.matmul(A,v)
side2 = np.multiply(lamd,v)
print('Av = λv : \n', side1, ' = ', side2, '\nTrue\n' )

# Show that AAv = λ^2v
side1 = np.matmul(A,np.matmul(A,v))
side2 = lamd**2*v
print('AAv = λ^2v : \n', side1, ' = ', side2, '\nTrue\n')

# Show that eigenvectors are orthogonal
v1 = eigVecs[:,0]
v2 = eigVecs[:,1]
v3 = eigVecs[:,2]
print('Inner product of v1 and v2: ', np.round(v1.dot(v2),10))
print('Inner product of v2 and v3: ', np.round(v2.dot(v3),10))
print('Inner product of v1 and v3: ', np.round(v1.dot(v3),10))
```

```
Av = λv :
 [-3.72093206 -6.70488789 -8.36085845]  =  [-3.72093206 -6.70488789 -8.
36085845]
True

AAv = λ^2v :
 [-42.2132832  -76.06570795 -94.85238636]  =  [-42.2132832  -76.0657079
5 -94.85238636]
True

Inner product of v1 and v2:  -0.0
Inner product of v2 and v3:  -0.0
Inner product of v1 and v3:  -0.0
```

Numerical Programming

7

Speed comparison between vectorized and non-vectorized code. Begin by creating an array of 10 million random numbers using the numpy random.randn module. Compute the sum of the squares first in a for loop, then using Numpy's `dot` module. Time how long it takes to compute each and report the results and report the output. How many times faster is the vectorized code than the for loop approach?

*Note: all code should be well commented, properly formatted, and your answers should be output using the `print()` function as follows (where the # represents your answers, to a reasonable precision):

```
Time [sec] (non-vectorized): ######

Time [sec] (vectorized):     ######

The vectorized code is ##### times faster than the vectorized code
```

In [35]:
```
import numpy as np
import time

# Generate the random samples
n = 100000000
r = np.random.randn(n)


# Compute the sum of squares the non-vectorized way (using a for loop)
# Start time
start_time = time.time()

# Calcuate the mean of the observations
mu = np.mean(r)

# Find sum of squares
summ = 0
for i in r:
    summ = summ + (i-mu)**2
    pass

# End the time counter and calcuate total time
end_time = time.time()
time_nonvec = end_time-start_time



# Compute the sum of squares the vectorized way (using numpy)
start_time = time.time()
summ_vec = np.dot(r,r - mu)
end_time = time.time()
time_vec = end_time-start_time


# Print the results
print("Time [sec] (non-vectorized): ", round(time_nonvec,2))
print("Time [sec] (vectorized): ", round(time_vec,2))
print("The vectorized code is, ", round(time_nonvec/time_vec,2), "times
 faster than the vectorized code")
```

```
Time [sec] (non-vectorized):  70.98
Time [sec] (vectorized):  1.35
The vectorized code is,  52.68 times faster than the vectorized code
```

8

One popular Agile development framework is Scrum (a paradigm recommended for data science projects). It emphasizes the continual evolution of code for projects, becoming progressively better, but starting with a quickly developed minimum viable product. This often means that code written early on is not optimized, and that's a good thing - it's best to get it to work first before optimizing. Imagine that you wrote the following code during a sprint towards getting an end-to-end system working. Vectorize the following code and show the difference in speed between the current implementation and a vectorized version.

The function below computes the function $f(x, y) = x^2 - 2y^2$ and determines whether this quantity is above or below a given threshold, `thresh=0`. This is done for $x, y \in \{-4, 4\}$, over a 2,000-by-2,000 grid covering that domain.

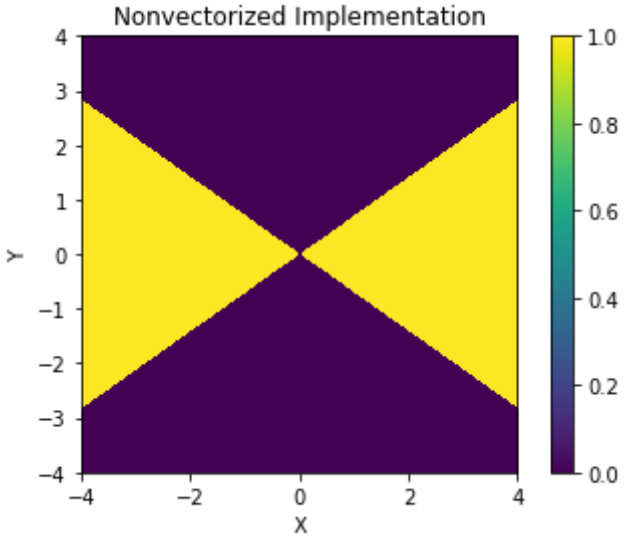
(a) Vectorize this code and demonstrate (as in the last exercise) the speed increase through vectorization and (b) plot the resulting data - both the function $f(x, y)$ and the thresholded output - using `imshow` (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.imshow.html?highlight=matplotlib%20pyplot%20imshow#matplotlib.pyplot.imshow) from `matplotlib`.

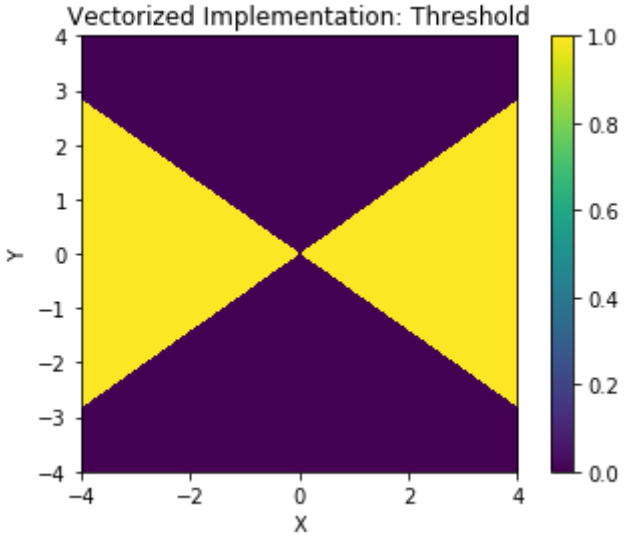
Hint: look at the `numpy` `meshgrid` (https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.meshgrid.html) documentation

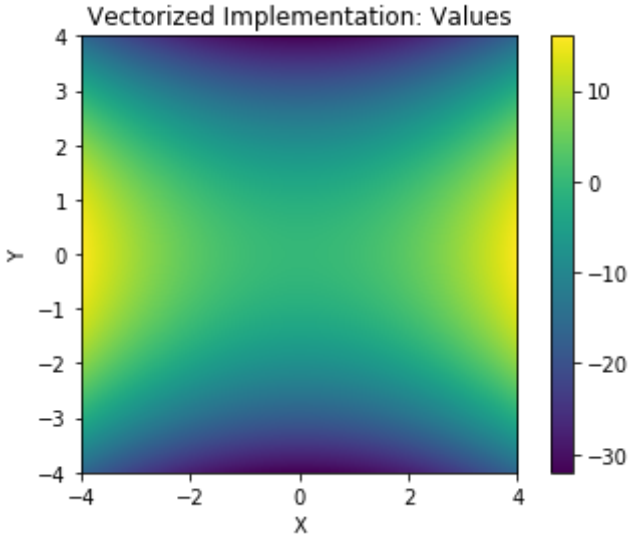
In [34]:
```
import numpy as np
import time
import matplotlib.pyplot as plt

# Initialize variables for this exerise
# Initialize scale and threshold
x = np.linspace(-4,4, num = 2000)
y = np.linspace(-4,4, num = 2000)
thresh = 0

# Nonvectorized implementation
start_time = time.time()

# Set up grid
X,Y = np.meshgrid(x,y)
nonVec = np.zeros((2000,2000))

# Define function
def f(x, y):
    func = x**2 - 2*y**2
    if func > thresh:
        return 1
    else:
        return 0
    pass

#Iterate over non-vectorized grid
for i in range(2000):
    for j in range(2000):
        nonVec[i,j] = f(X[i,j],Y[i,j])
        pass
    pass
end_time = time.time()
time_nonvec = end_time-start_time


# Vectorized implementation - threshold
start_time = time.time()
X, Y = np.meshgrid(x, y)
vec_values = X**2 - 2* Y**2
vec = (vec_values > thresh)*1
end_time = time.time()
time_vec = end_time-start_time


# Print the time for each and the speed increase
print("Time [sec] (non-vectorized): ", round(time_nonvec,2))
print("Time [sec] (vectorized): ", round(time_vec,2))
print("The vectorized code is, ", round(time_nonvec/time_vec,2), "times
 faster than the vectorized code")


# Plot the result
print()

extents = (-4, 4, -4, 4)
```

```
plt.imshow(nonVec, extent=extents)
plt.title('Nonvectorized Implementation')
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar()
plt.show()

plt.imshow(vec, extent=extents)
plt.title('Vectorized Implementation: Threshold')
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar()
plt.show()

plt.imshow(vec_values, extent=extents)
plt.title('Vectorized Implementation: Values')
plt.xlabel('X')
plt.ylabel('Y')
plt.colorbar()
plt.show()
```

```
Time [sec] (non-vectorized):  9.36
Time [sec] (vectorized):  0.07
The vectorized code is,  129.1 times faster than the vectorized code
```







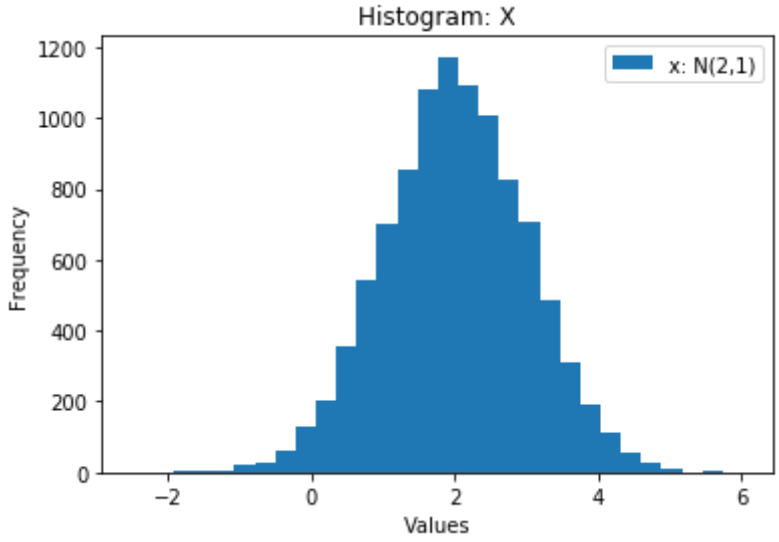
9

This exercise will walk through some basic numerical programming exercises.

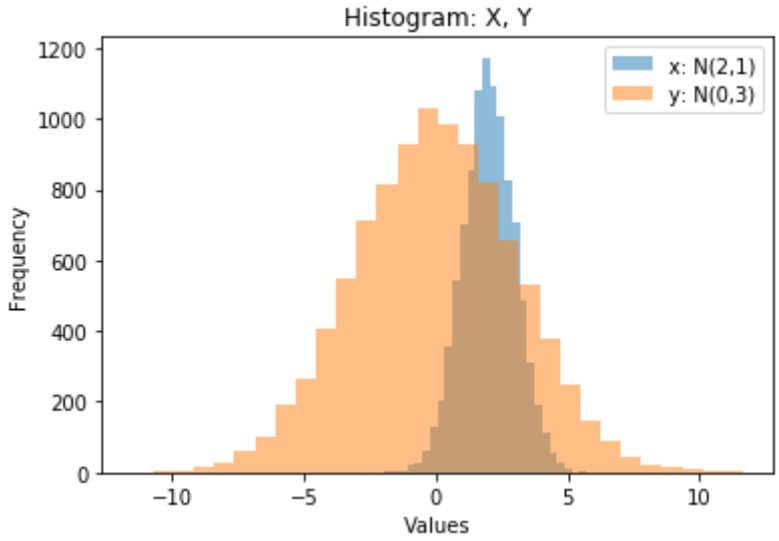
1. Synthesize $n = 10^4$ normally distributed data points with mean $\mu = 2$ and a standard deviation of $\sigma = 1$. Call these observations from a random variable X, and call the vector of observations that you generate, $\mathbf{x}$.
2. Calculate the mean and standard deviation of $\mathbf{x}$ to validate (1) and provide the result to a precision of four significant figures.
3. Plot a histogram of the data in $\mathbf{x}$ with 30 bins
4. What is the 90th percentile of $\mathbf{x}$? The 90th percentile is the value below which 90% of observations can be found.
5. What is the 99th percentile of $\mathbf{x}$?
6. Now synthesize $n = 10^4$ normally distributed data points with mean $\mu = 0$ and a standard deviation of $\sigma = 3$. Call these observations from a random variable Y, and call the vector of observations that you generate, $\mathbf{y}$.
7. Plot the histogram of the data in $\mathbf{y}$ on a (new) plot with the histogram of $\mathbf{x}$, so that both histograms can be seen and compared.
8. Using the observations from $\mathbf{x}$ and $\mathbf{y}$, estimate $E[XY]$

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

# 1
# Generate normally distributed data points with mean μ=2 and a standard
 deviation of σ=1
mu, sd = 2, 1 # mean and standard deviation
x = np.random.normal(mu, sd, 10000)

# 2
# Calculate the mean and standard deviation and provide the result to a
 precision of four significant figures
print('Observed x mean: ', '%s' % float('%.4g' % np.mean(x)))
print('Observed x standard deviation: ', '%s' % float('%.4g' % np.std(x
)))

# 3
# Plot histogram
plt.hist(x, 30, label = 'x: N(2,1)')
plt.legend(loc='upper right')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram: X')
plt.show()

# 4
# What is the 90th percentile of  x ?
print('x 90th percentile: ', '%s' % float('%.4g' % np.percentile(x, 90
)))

# 5
# What is the 90th percentile of  x ?
print('x 99th percentile: ', '%s' % float('%.4g' % np.percentile(x, 99
)))

# 6
# Generate normally distributed data points with mean μ=2 and a standard
 deviation of σ=1
mu, sd = 0, 3 # mean and standard deviation
y = np.random.normal(mu, sd, 10000)

# 7
# Plot the histogram of the data in y on a (new) plot with the histogram
 of x
plt.hist(x, 30, alpha = .5, label = 'x: N(2,1)')
plt.hist(y, 30, alpha = .5, label = 'y: N(0,3)')
plt.legend(loc='upper right')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram: X, Y')
plt.show()

# 8
print('E(XY) = E(X) + E(Y) = ',
      '%s' % float('%.4g' % np.mean(x)), ' * ', '%s' % float('%.4g' % np
```

```
.mean(y)),
      ' = ', '%s' % float('%.4g' % (np.mean(x) * np.mean(y))))
Observed x mean:  2.013
Observed x standard deviation:  0.9959
```




```
x 90th percentile:  3.286
x 99th percentile:  4.297
```




```
E(XY) = E(X) + E(Y) =  2.013  *  -0.0006118  =  -0.001232
```

10

Estimate the integral of the function $f(x)$ on the interval $0 \leq x < 2.5$ assuming we only know the following points from f:

Table 1. Dataset containing n=5 observations

x_i	**0.0**	**0.5**	**1.0**	**1.5**	**2.0**
y_i	6	7	8	4	1

In [142]:
```
import numpy as np

#Initialize variables
t = .5 # step
x = np.arange(0,2.5,t)
y = np.array((6,7,8,4,1))

# Calculate right hand sum and left hand sum
rhs = 0
for i in range(1,len(y)):
    rhs = rhs + y[i]*t
    #print(y[i]*t)
    pass

lhs = 0
for i in range(0,len(y)-1):
    lhs = lhs + y[i]*t
    #print(y[i]*t)
    pass

# Average right and left hand sum to estimate integral
estimate = (rhs + lhs)/2
print('Estimated integral = ', estimate)
```

```
Estimated integral =  11.25
```

Version Control via Git

11

Complete the Atlassian Git tutorial (https://www.atlassian.com/git/tutorials/what-is-version-control), specifically the following sections. Try each concept that's presented. For this tutorial, instead of using BitBucket, use Github. Create a github account here if you don't already have one: https://github.com/ (https://github.com/)

1. What is version control (https://www.atlassian.com/git/tutorials/what-is-version-control)
2. What is Git (https://www.atlassian.com/git/tutorials/what-is-git)
3. Install Git (https://www.atlassian.com/git/tutorials/install-git)
4. Setting up a repository (https://www.atlassian.com/git/tutorials/install-git)
5. Saving changes (https://www.atlassian.com/git/tutorials/saving-changes)
6. Inspecting a repository (https://www.atlassian.com/git/tutorials/inspecting-a-repository)
7. Undoing changes (https://www.atlassian.com/git/tutorials/undoing-changes)
8. Rewriting history (https://www.atlassian.com/git/tutorials/rewriting-history)
9. Syncing (https://www.atlassian.com/git/tutorials/syncing)
10. Making a pull request (https://www.atlassian.com/git/tutorials/making-a-pull-request)
11. Using branches (https://www.atlassian.com/git/tutorials/using-branches)
12. Comparing workflows (https://www.atlassian.com/git/tutorials/comparing-workflows)

For your answer, affirm that you either completed the tutorial or have previous experience with all of the concepts above. Do this by typing your name below and selecting the situation that applies from the two options in brackets.

I, Anna Berman, affirm that I have completed the above tutorial.

12

Using Github to create a static HTML website:

1. Create a branch in your `machine-learning-course` repo called "gh-pages" and checkout that branch (this will provide an example of how to create a simple static website using Github Pages (https://pages.github.com/))
2. Create a file called "index.html" with the contents "Hello World" and add, commit, and push it to that branch.
3. Submit the following: (a) a link to your github repository and (b) a link to your new "Hello World" website. The latter should be at the address https://[USERNAME].github.io/ECE590-assignment0 (https://[USERNAME].github.io/ECE590-assignment0) (where [USERNAME] is your github username).

a) Link to Github account: https://github.com/aberman6/machine-learning-course (https://github.com/aberman6/machine-learning-course)

b) Link to new 'Hello World' website: https://aberman6.github.io/machine-learning-course/ (https://aberman6.github.io/machine-learning-course/)

Exploratory Data Analysis

13

Here you'll bring together some of the individual skills that you demonstrated above and create a Jupyter notebook based blog post on data analysis.

1. Find a dataset that interests you and relates to a question or problem that you find intriguing
2. Using a Jupyter notebook, describe the dataset, the source of the data, and the reason the dataset was of interest.
3. Check the data and see if they need to be cleaned: are there missing values? Are there clearly erroneous values? Do two tables need to be merged together? Clean the data so it can be visualized.
4. Plot the data, demonstrating interesting features that you discover. Are there any relationships between variables that were surprising or patterns that emerged? Please exercise creativity and curiosity in your plots.
5. What insights are you able to take away from exploring the data? Is there a reason why analyzing the dataset you chose is particularly interesting or important? Summarize this as if your target audience was the readership of a major news organization - boil down your findings in a way that is accessible, but still accurate.
6. Create a public repository on your github account titled "machine-learning-course". In it, create a readme file that contains the heading "ECE590: Introductory Machine Learning for Data Science". Add, commit, and push that Jupyter notebook to the master branch. Provide the link to the that post here.

You can find my analysis here (https://github.com/aberman6/machine-learning-course/blob/master/Assignment_1_Titanic.ipynb). If the previous link has a problem loading, try looking here (https://nbviewer.jupyter.org/github/aberman6/machine-learning-course/blob/master/Assignment_1_Titanic.ipynb).