Bayesian Classification on Parametric Class Probability Distributions

Michael Ferko
Dept. of Electrical and Computer Engineering
University of Louisville
Louisville, USA
Email: Mike.W.Ferko@gmail.com

***Abstract*—A Bayesian Classifier can discriminate or decide which category a random value belongs. This series of projects aims to implement multidimensional Bayesian classification by using parametric class probability distributions.**

Keywords—Bayesian Classifier, Pattern Recognition, Machine Intelligence, Probability & Statistics, Engineering

I. INTRODUCTION

The project is four parts: Part 1 deals with generating one-dimensional uniform random variables, as basis for generating generalized distribution. Part 2 deals with generating generalized one-dimensional random variables. Part 3 deals with Bayesian simulation in one-dimensional class distribution. Part 4 deals with multidimensional Bayesian class distributions.

II. PART 1: GENERATION OF STANDARD UNIFORM RANDOM VARIABLE

A linear congenital generator (LCG) algorithm is derived from the universal hash function where a set of random numbers from a dictionary is uniformly and universally reproducible via utilizing the law of large numbers. The law of large numbers of states that an observed sample average from a large sample will be close to the true population average and that it will get closer the larger the sample. So, utilizing very large values number of samples i.e. 20k or 10ksamples can reproduce the same results with the same seed same increment and same multiplier. What this means is that random sequences can be generated from a deterministic iterative scheme such as the LCG shown in equation (2). Figures 1 and 2 show the mapping configuration of universal keys to a hash table for crypto security and the building blocks of a universal hash function recursion equation built for reproducing hash tables while only knowing the seed, increment and multiplier. The recursive formula for a Linear Congenital Generator (LCG) is:

$$Z_i = aZ_{i-1} + c \tag{1}$$

The multiplicative LCG is:

$$Z_i = (aZ_{i-1}) mod m \tag{2}$$

Where m is the modulus, a is the multiplier and c is the increment.

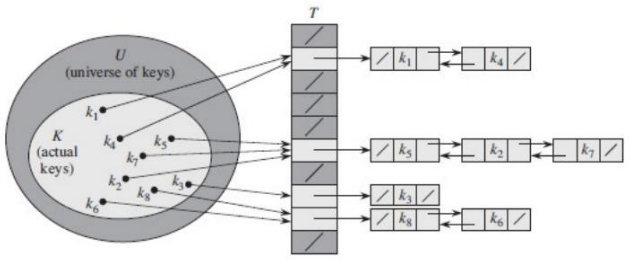


Figure 11.3 Collision resolution by chaining. Each hash-table slot $T[j]$ contains a linked list of all the keys whose hash value is j. For example, $h(k_1) = h(k_4)$ and $h(k_5) = h(k_7) = h(k_2)$. The linked list can be either singly or doubly linked; we show it as doubly linked because deletion is faster that way.

Figure 1: Mapping from Universal Sample Space to Hash Table T with keys $k_i = x_i.key$

Designing a universal class of hash functions

It is quite easy to design a universal class of hash functions, as a little number theory will help us prove. You may wish to consult Chapter 31 first if you are unfamiliar with number theory.

We begin by choosing a prime number p large enough so that every possible key k is in the range 0 to $p-1$, inclusive. Let $\mathbb{Z}_p$ denote the set $\{0, 1, \ldots, p-1\}$, and let $\mathbb{Z}_p^*$ denote the set $\{1, 2, \ldots, p-1\}$. Since p is prime, we can solve equations modulo p with the methods given in Chapter 31. Because we assume that the size of the universe of keys is greater than the number of slots in the hash table, we have $p > m$.

We now define the hash function h_{ab} for any $a \in \mathbb{Z}_p^*$ and any $b \in \mathbb{Z}_p$ using a linear transformation followed by reductions modulo p and then modulo m:

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m . \tag{11.3}$$

For example, with $p = 17$ and $m = 6$, we have $h_{3,4}(8) = 5$. The family of all such hash functions is

$$\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\} . \tag{11.4}$$

Each hash function h_{ab} maps $\mathbb{Z}_p$ to $\mathbb{Z}_m$. This class of hash functions has the nice property that the size m of the output range is arbitrary—not necessarily prime—a feature which we shall use in Section 11.5. Since we have $p-1$ choices for a and p choices for b, the collection $\mathcal{H}_{pm}$ contains $p(p-1)$ hash functions.

Theorem 11.5
The class $\mathcal{H}_{pm}$ of hash functions defined by equations (11.3) and (11.4) is universal.

Proof Consider two distinct keys k and l from $\mathbb{Z}_p$, so that $k \neq l$. For a given hash function h_{ab} we let

$r = (ak + b) \bmod p$,
$s = (al + b) \bmod p$.

We first note that $r \neq s$. Why? Observe that

$r - s \equiv a(k - l) \pmod p$. ⟶ $Z_i = (aZ_{i-1}) mod m$

Figure 2: Designing the Universal Hash Function and Producing the Multiplicative LCG in Equation 2

II.a. GENERATE THREE STREAMS OF RANDOM NUMBERS WITH 20,000 SAMPLES

```
stream_1 =  [[0.71391474 0.26896618 0.96597507 ... 0.21872684 0.95513748 0.50136883]]
stream_2 =  [[0.98712207 0.66079865 0.31836022 ... 0.20709011 0.79465411 0.17992192]]
stream_3 =  [[0.46317354 0.09581516 0.58970319 ... 0.34300703 0.52677591 0.88788533]]
```

Figure 3: Three Generated Streams printed

II.b. VISUALIZE THE GENERATED SAMPLES. HINT: YOU CAN USE A SCATTER PLOT

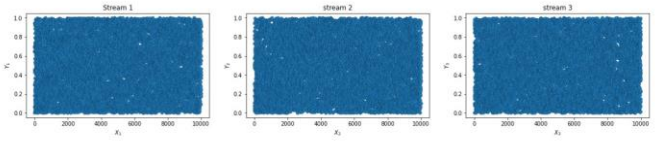


Figure 4: Scatter Plot of 3 randomly generated sample streams

II.c. COMPUTE THE HISTOGRAM OF THE THREE STREAMS, THEN NORMALIZE THEM TO BECOME A PROBABILITY DENSITY FUNCTION (PDF)

```
Hist(10 bins in [3.880573250159003e-05,0.9999501900881669], with sum 20000.0, 0 empty bins, and 0 non-finite values)
Hist(10 bins in [4.8278696363257545e-05,0.9999847295161585], with sum 20000.0, 0 empty bins, and 0 non-finite values)
Hist(10 bins in [4.6795215028483380e-06,0.9999911429818706], with sum 20000.0, 0 empty bins, and 0 non-finite values)
Hist(10 bins in [3.880573250159003e-05,0.9999501900881669], with sum 10.000886234977632, 0 empty bins, and 0 non-finite values)
Hist(10 bins in [4.8278696363257545e-05,0.9999847295161585], with sum 10.000635532189596, 0 empty bins, and 0 non-finite values)
Hist(10 bins in [4.6795215028483380e-06,0.9999911429818706], with sum 10.000135367228728, 0 empty bins, and 0 non-finite values)
```

Figure 5: Printed Normalized histogram of 3 randomly generated streams

II.d. VISUALIZE THE PDF 'S OF THE THREE STREAMS

The samples are uniformly distributed and represent a uniform distribution. 20,000 samples of randomly generated numbers have distributed themselves as the standard uniform distribution. As shown in Figure 6:

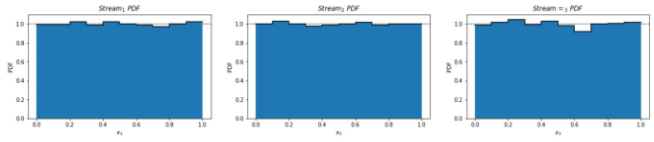


Figure 6: Plot of Normalized histogram of 3 randomly generated streams

III. PART 2: GENERALIZED ONE-DIMENSIONAL RANDOM NUMBER GENERATION

The probability density function (pdf) fX(x) and the Cumulative Distribution Function (CDF) FX(x) (i.e. the probability mass function (pmf) or the Probability Distribution Function (big PDF)) for the uniform random variable X (big X) between a lower bound a and an upper bound b X U[a, b] is given by:

$$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \le x \le b, \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

and

$$F_X(x) = \begin{cases} 0 & \text{if } x < a, \\ \frac{x-a}{b-a} & \text{if } a \le x \le b, \\ 1 & \text{if } b > x \end{cases} \tag{4}$$

Where x (small x) is the random values that the random variable X can take on the real number line Rd (d is the number of dimensions on the real number line where the simplest case for a one-dimensional function of one random variable without functional mapping between domains of random variables).

The uniform distributions stream 1 (first dimension), stream 2 (2nd dimension) and stream 3 (3rd dimension) are probability density functions fX(x) of the random samples generated by the LCG.

By mapping the standard uniform distribution to other distributions, we can generate any other distribution by inversely mapping the generated uniform samples to the intended domain of the distribution we desire.

III.a. GENERATE THREE STREAMS OF RANDOM NUMBERS WITH 20,000 SAMPLES

This was the approach taken in the first part tasks.

III.b. USE THE MAPPING APPROACH IN THIS CHAPTER TO GENERATE THE 10,000 SAMPLES FROM THE FOLLOWING DISTRIBUTIONS:

We can generate a standard Gaussian (normal) distribution X ˜N(0, 1), from which to generate X ˜N(−1, 4) using the affine transformation Y = σX + μ with the desired mean μ and standard deviation σ.

III.b.1. X ~N(0, 1):

To do this exercise the data was split from 20,000 samples generated from stream 1 into two sub sample streams of u1 and u2 for the two uniform distributions formed from 10,000 samples. The Box-Muller transformation is special in that it utilizes the properties of the standard uniform distribution being equal to 1 to be the radius of the unit circle of length 1. The transformation also uses the fact that the other uniform distribution can create the circumference of the unit circle i.e. $\theta = 2 * \pi * r$, if r = 1 the unit circle has completed 1 full rotation about the center axis. Taking these facts into consideration, the Box-Muller transformation takes the polar coordinates to Cartesian (rectangular) coordinates transformation i.e. the Leonhard Euler expansion $rej2\pi f = r[\cos(2\pi f) + j\sin(2\pi f)] = x + jy$. Since Euler's number e is actually just a result of the binomial expansion for approximating n! (i.e. $n! = \sqrt{2\pi n} n e n$) and by Stirling's formula we achieve a Gaussian distribution, we can say that the inverse transform of the polar to Cartesian coordinates will achieve a normal distribution in a Cartesian coordinate frame. This is what the Box-Muller transformation achieves (i.e. Normal distribution 1 N1 = x = $r\cos(2\pi f)$ and Normal distribution 2 N2 = y = $r\cos(2\pi f)$. Where r and f are the scattered data points forming a uniform distribution of 10,000 samples each. After transformation, the data is reinserted into one Normal1 distribution where the even indexed entries come from the first 2,500 samples of the transformed N1 and the odd indexed entries come from the first 2,500 samples of the transformed N2. This is shown in the following plots. The 1

notation was kept to remember that this data is generated from The 1 dimensional array of Stream1.

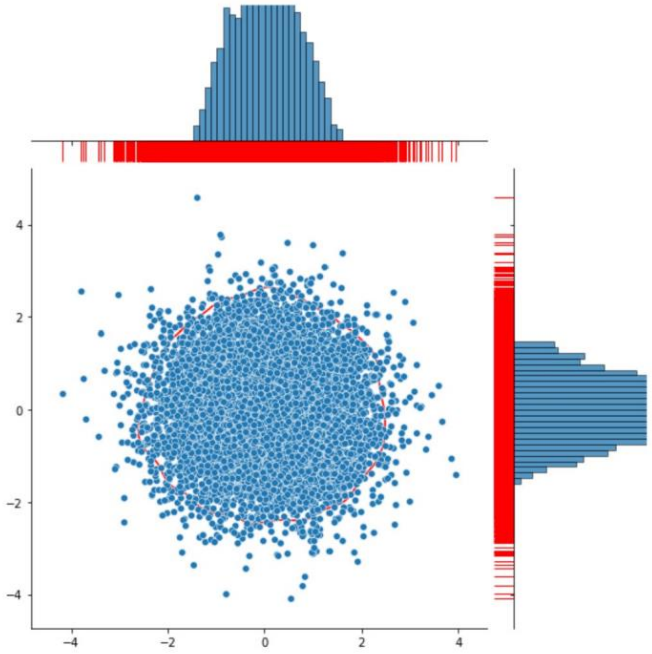


Figure 7: Seaborn plot of Box-Muller Transform pairs

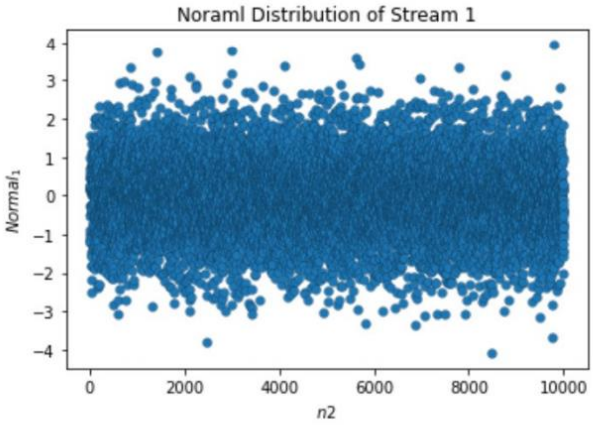


Figure 8: Scatter Plot of N_1 N_2 spliced data to form $Normal_1$

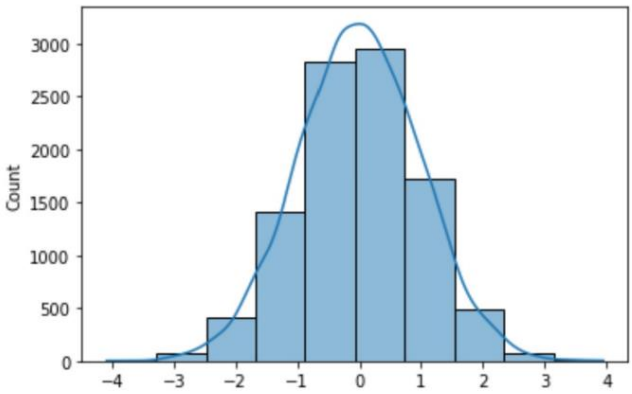


Figure 9: Histogram Plot of N_1 N_2 spliced data to form $Normal_1$ with 10 bins

Returning to this transformation became apparent that the intended transformation generation of the gaussian was the inverse transformation. And so, another block of code was written to create the inverse transform generation. Where the inverse transform from the standard uniform to the standard normal distribution could be computed using the following equation:

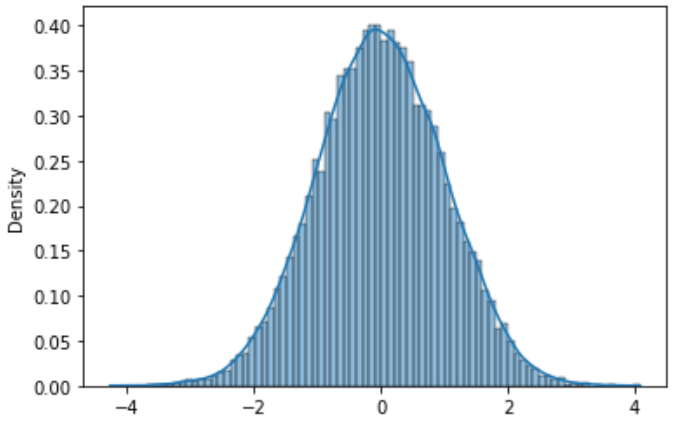
$$Z = \mu + \sqrt{2}\,\sigma\, erfc^{-1}(2U - 1) \qquad (5)$$

Implemented by:

```
# Gaussian Generation using error function compliment approximation
# Equation implimentation from John D. Cook's blog

# imports
import numpy as np
import math
from scipy import special
import seaborn as sns

# Linear Congruential Generator (LCG) utilizing Universal Hash functi
# Algorithm from Monte Carlo Simulations
def rng(m=2**32, a=1103515245, c=12345):
    rng.current = (a*rng.current + c) % m # z_i = (a*z_i-1 + c) mod(m
    return rng.current/m

# setting the seed of the LCG
rng.current = 1

# generate Uniform(0, 1)
a = 0; b = 1; N = 20000 # start, stop, number of samples
U = np.array([a+(b-a)*rng() for i in range(N)])
mean = 0; standard_deviation = 1
Z = mean + 2**0.5 * standard_deviation * special.erfinv(U * 2 - 1)

# plot Gaussian
sns.histplot(Z, stat = "density", kde = True)
```

Code Implementation of inverse error function transformation

The above code inverse transform was fully integrated and used to generate all the Normal_1 distribution generated in these projects. This includes everything up until the Box Muller approach was asked for in project 4.



Plot of Generate Gaussian from Inverse Transform

From this figure we see the inverse mapping method has produced a gaussian standard normal distribution from the standard uniform distribution as intended from the inverse mapping method of equation 5.

III.b.1. X ~N(-1, 4):

The affine transformation Y = σX +μ is used to transform any standard normal distribution to any other normal distribution of a desired mean μ and desired standard deviation σ. The desired normal distribution is as shown:

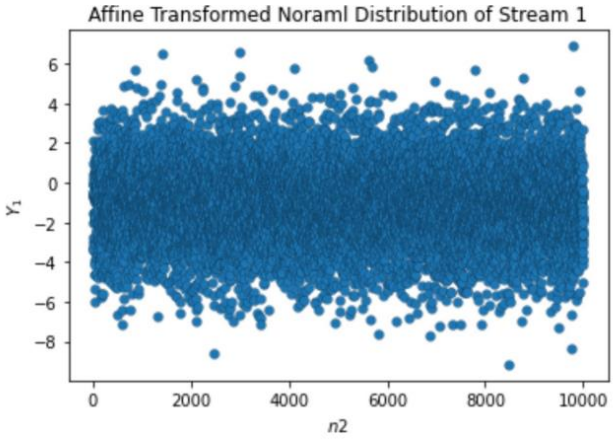


Figure 10: Scatter plot of Y_1

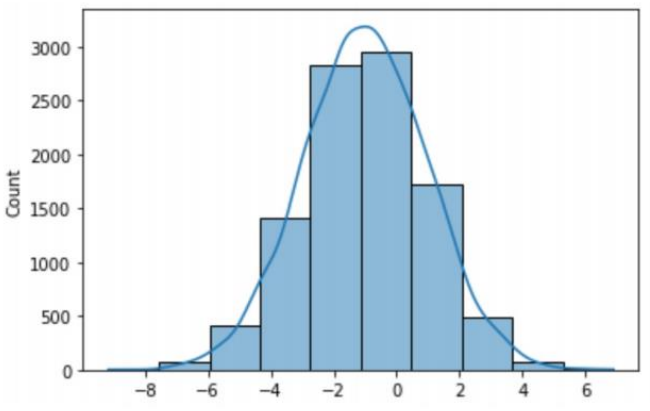


Figure 11: Seaborn Histogram plot of Y_1 with 10 bins

III.c. Use the mapping approach to generate the 10,000 samples from an exponential distribution:

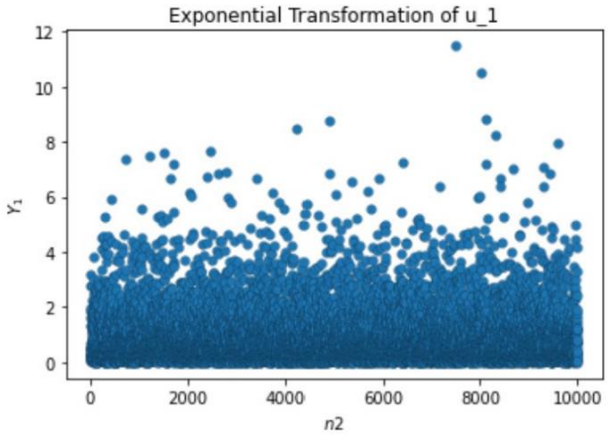


Figure 12: Scatter plot of Y_1 esponential

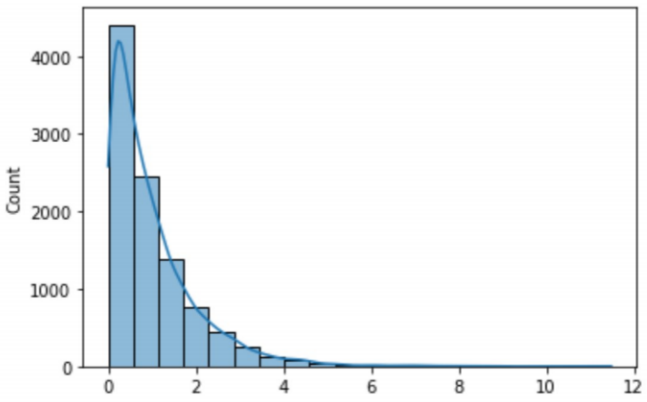


Figure 13: Seaborn Histogram plot of Y_1 exponential with 20 bins

III.d. USE THE MAPPING TO GENERATE THE 10,000 SAMPLES FROM A POISSON DISTRIBUTION WITH PARAMETER 1

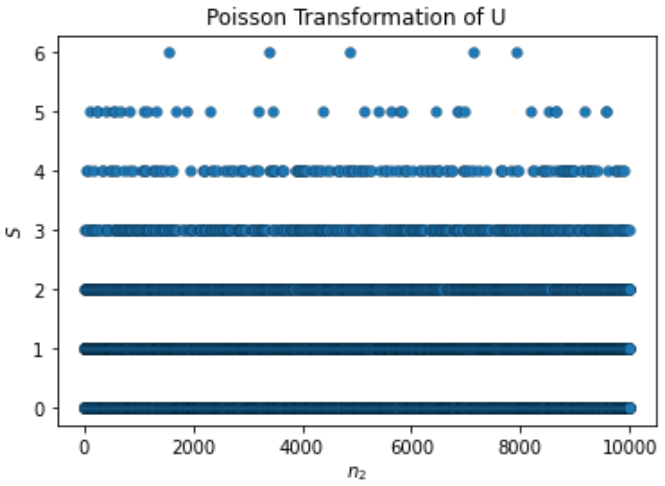


Figure 14: Scatterplot of Poisson Distribution

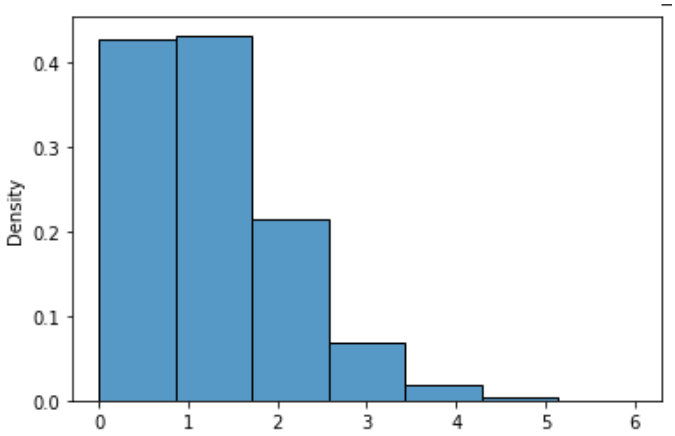


Figure 15: Histogram of Poisson Distribution

III.e. TEST THE QUALITY OF THE REALIZATIONS IN PARTS 1-4 USING THE CHI-SQUARE TEST AND THE RUN TEST

Pearson Chi-Squared Goodness-of-Fit Test

The chi-square goodness-of-fit test determines if a data sample comes from a specified probability distribution, with parameters estimated from the data.

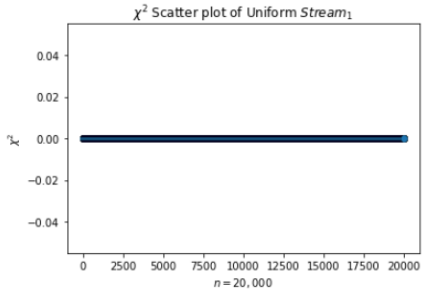
The test groups the data into bins, calculating the observed and expected counts for those bins, and computing the chi-square test statistic

$$\chi^2 = \sum_{i=1}^{N} \frac{(O_i - E_i)^2}{E_i}$$

where Oi are the observed counts and Ei are the expected counts based on the hypothesized distribution. The test statistic has an approximate chi-square distribution when the counts are sufficiently large.

Figure 16: Pearson's Chi Squared Test Procedure

The Pearson's chi squared goodness of fit test as described in Figure 16 determines if the observed sample of the distribution generated fits that of the expected value of that distribution. So, in Figure 17 we see the chi squared statistic is zero to represent the expected values all fit the observed values. And we get an associated P-value back from the embedded chi squared test provided from NumPy and create an if statement where if the p-value is less than an alpha of 0.5 to represent a confidence interval of 95% we accept the null hypothesis that the distribution properly fits the expected values and in turn the distribution is a good fit. The runs test for determining if a set of samples is generated in a random manner was a bit complicated in that we had to create a function to determine a comparison of the data to a standard normal distribution and then get a p-value from the fitted Z standard normal fitted data. That p-value was then, again, passed through a null hypothesis and alternative hypothesis if statement where the null hypothesis was chosen only when the p-value was less than an alpha of 0.5 to represent a confidence interval of 95%. To state it simply the standard uniform U(0,1), standard Normal N(0,1), Affine Transformed Normal N(-1,4), exponential distribution and poison distribution all tests accepted the two Null hypotheses where all generated distributions were a good fit of the generated samples and all generated distributions were confirmed to have been generated in a random manner. The results of these Tests are as shown in the following Figures:




```
Chi Squared Hypothesis Test

H0: (null hypothesis) The data are comming from a uniform distribution.

H1: (alternative hypothesis) The data are not comming from a uniform distribution.

conclusion: Accept the Null Hypothesis

Run Test Hypothesis Testing for Randomness

H0: (null hypothesis) the sequence was produced in a random manner.

H1: (alternative hypothesis) the sequence was not produced in a random manner.

conclusion: Accept the Null Hypothesis
```

Figure 17: Results of Chi-Squared and Runs Test on Standard Uniform Distribution

```
-1.007543709217046 1.9894523293213129
-39278.93889249027 1.0

H0: (null hypothesis) The data are comming from a normal distribution.

H1: (alternative hypothesis) The data are not comming from a normal distribution.

Accept the Null Hypothesis

Run Test Hypothesis Testing for Randomness

H0: (null hypothesis) the sequence was produced in a random manner.

H1: (alternative hypothesis) the sequence was not produced in a random manner.

conclusion: Accept the Null Hypothesis
```

Figure 18: Results of Chi-Squared and Runs Test on Standard Normal Distribution

```
H0: (null hypothesis) The data are comming from a exponential distribution.

H1: (alternative hypothesis) The data are not comming from a exponential distribution.

Accept the Null Hypothesis

Run Test Hypothesis Testing for Randomness

H0: (null hypothesis) the sequence was produced in a random manner.

H1: (alternative hypothesis) the sequence was not produced in a random manner.

conclusion: Accept the Null Hypothesis
```

Figure 19: Results of Chi-Squared and Runs Test on Exponential Distribution

```
H0: (null hypothesis) The data are comming from a poisson distribution.

H1: (alternative hypothesis) The data are not comming from a poisson distribution.

Accept the Null Hypothesis

Run Test Hypothesis Testing for Randomness

H0: (null hypothesis) the sequence was produced in a random manner.

H1: (alternative hypothesis) the sequence was not produced in a random manner.

conclusion: Accept the Null Hypothesis
```

Figure 19: Results of Chi-Squared and Runs Test on Poisson Distribution

IV. PART 3: BAYESIAN CLASSIFICATION IN ONE-DIMENSION

The problem under consideration is this: given a measurement $\boldsymbol{x}$ (can be multidimensional) that can belong to any of several possibilities (classes) Ci ,i∈[1,N], each class/possibility can take place with a certain probability $P(Ci)$. How to assign $\boldsymbol{x}$ to a particular class Ci?

IV.a. PART 1: BINARY CLASSIFICATION ON UNIFORM DISTRIBUTION

Given a two-class discriminant function with $fX(x|ci)$, where i = 1,2. Let $P(c1) = 3P(c2)$. Calculate the decision boundary and the probability of error for the following cases:

i) $c = -a$,$d = a$.

ii) $c = a/2$,$d = b/2$

iii) $c = 0$,$d = b$

Chose: a =1, b =2 to plot all cases at once

$$P(C_1) = 3P(C_2)$$

- $P(C_1) + P(C_2) = 1$
- $3P(C_2) + P(C_2) = 1$
- $4P(C_2) = 1$
- $P(C_2) = \frac{1}{4}$
- $P(C_1) + P(C_2) = 1$
- $P(C_1) + \frac{1}{4} = 1$
- $P(C_1) = 1 - \frac{1}{4} = \frac{3}{4}$

$$h_i(x) = f_X(x|C_i) * P(C_i)$$

- $h_1(x) = f_X(x|C_1) * P(C_1)$
- $h_1(x) = \frac{3}{4} f_X(x|C_1)$
- $h_2(x) = f_X(x|C_2) * P(C_2)$
- $h_2(x) = \frac{1}{4} f_X(x|C_2)$

Decision Boundary:

$$h(x) = h_1(x) - h_2(x)$$

- $h(x) = \frac{3}{4} f_X(x|C_1) - \frac{1}{4} f_X(x|C_2)$

Figure 20: Total Probability Calculations & Decision Boundary Identification

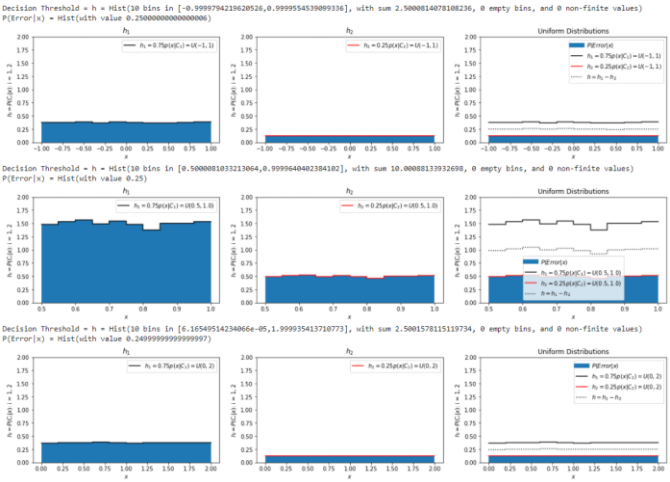
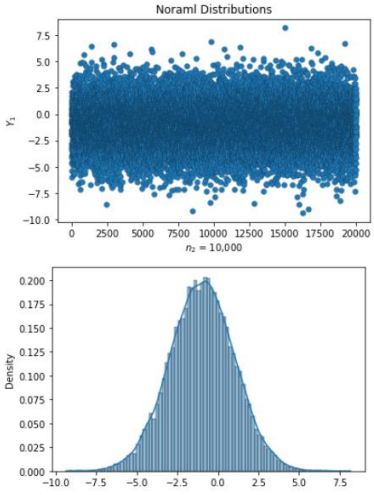


Figure 21: Classification of Two Uniformly Distributions where P(Error) = 0.25

From Figure 21

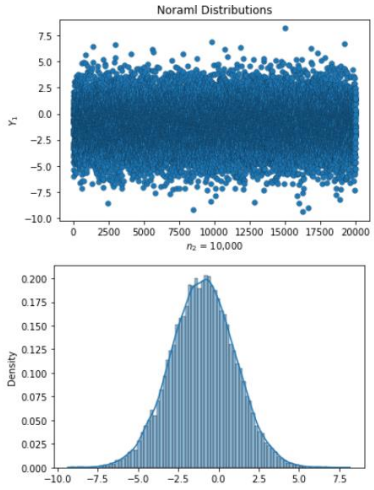
IV.b. PART 2: BINARY CLASSIFICATION ON NORMAL DISTRIBUTION

Given a two-class discriminant function with $fX(x|ci)$, where $fX(x|c1)$~$N(-1,4)$ and $fX(x|c2)$~$N(0,2)$. Let $P(c1) = 3P(c2)$. Calculate the decision boundary and probability of error. Simulate the above problem by generating 20,000 samples from each class and calculate the decision boundary and the probability of error, knowing the ground truth.




```
sample mean: X_bar = -0.9813781693935629 , sample standard deviation: s = 1.9977389276092063
mean and variance should be:
 population mean: Mean = -1 , population standard deviation: StdDev = 2.0
```

Figure 22: Generation of N(-1,4)




```
sample mean: X_bar = 0.013167622699918849 , sample standard deviation: s = 1.4126147427528113
mean and variance should be:
 population mean: Mean = 0 , population standard deviation: StdDev = 1.4142135623730951
```

Figure 23: Generation of N(0,2)

Decision Boundary:

$$h(x) = h_1(x) - h_2(x)$$

- $h(x) = \frac{3}{4} f_X(x|C_1) - \frac{1}{4} f_X(x|C_2)$

$$g(x) = g_1(x) - g_2(x)$$

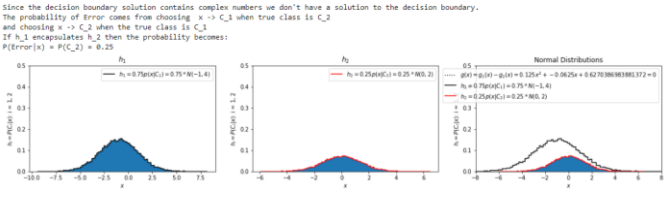


Figure 24: Decision Boundary Identification

From Figure 24, since the decision boundary solution contains complex numbers, we do not have a solution to the decision boundary. However, since class 1 encapsulates class 2 we can say the probability of error will be the probability of choosing class 2. So, the total probability of error is the area under the gaussian curve for class 2 times the prior probability associated with class 2 which is $P(Error) = P(\epsilon) = \int f_X(x|C_2)P(C_2)dx = P(C_2) = \left(\frac{1}{4}\right) = 0.25$. In Figure 24 we can see the right most graph shows the probability of error in blue under the red plot of h_2.

```
# Need to find A, B, C and wio coefficients for decision boundary function g(x) = g1(x) - g2(x) = 0
alph = P_C_1/P_C_2 # prior probabilities ratio
A = (1/2)*((1/desired_variance_2) - (1/desired_variance_1))
B = ((desired_mean_1/(desired_variance_1**2)) - (desired_mean_2/(desired_variance_2**2)))
C = (-1/2)*(desired_mean_1/desired_standard_deviation_1)**2 +
  (1/2)*(desired_mean_2/desired_standard_deviation_2)**2
  - math.log(desired_standard_deviation_1/desired_standard_deviation_2) + math.log(alph)
g = A*n**2 + B*n + C


# - ((x+1)^2)/8 + ln(3/8) = -(x^2)/4-lnV2
# -> Simplifying: x^2-x+0.628=0
# The solution is: x=(1±j1.229)/2; complex

# Simply ⟦P(C_1)f⟧ _X (x|C_1 ) eclipses ⟦P(C_2)f⟧ _X (x|C_2 )
# and as such x→C_1 always; which means the probability of
# error will be: ∫_(-∞)^∞ ⟦ ⟦P(C_2)f⟧ _X (x|C_2 )dx = P(C_2 ).⟧
```

Figure 25: Decision Boundary Does Not Exist because of complex roots & P(E) = P(C_2) = 0.25 because C_1 encapsulate C_2

IV.c. PART 3: BINARY CLASSIFICATION USING MIXED DISTRIBUTIONS

Consider two-class classifications from a Gaussian and Uniform distributions, where the class densities are $fX(x|c1)\sim N(-1,4)$ and $fX(x|c2)\sim U(-1,1)$ and the class densities are $P(c1) = 2P(c2)$.

a) Generate 20,000 samples from each class.

b) Calculate the probability of error in the classification.

c) Repeat (b) using theoretical computation of the decision boundary and the probability of error. Compare the results with (b).

$$P(C_1) = 2P(C_2)$$

- $P(C_1) + P(C_2) = 1$
- $2P(C_2) + P(C_2) = 1$
- $3P(C_2) = 1$
- $P(C_2) = \frac{1}{3}$
- $P(C_1) + P(C_2) = 1$
- $P(C_1) + \frac{1}{3} = 1$
- $P(C_1) = 1 - \frac{1}{3} = \frac{2}{3}$

$$h_i(x) = f_X(x|C_i) * P(C_i)$$

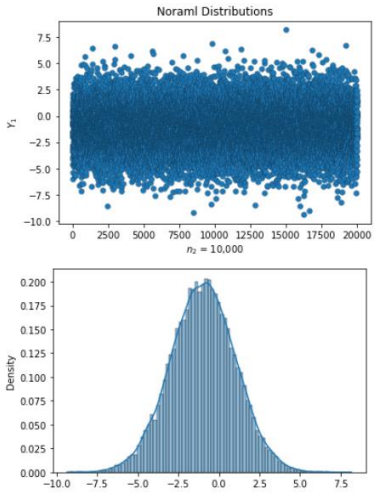
- $h_1(x) = f_X(x|C_1) * P(C_1)$
- $h_1(x) = \frac{2}{3} f_X(x|C_1)$
- $h_2(x) = f_X(x|C_2) * P(C_2)$
- $h_2(x) = \frac{1}{3} f_X(x|C_2)$

Decision Boundary:

$$h(x) = h_1(x) - h_2(x)$$

- $h(x) = \frac{2}{3} f_X(x|C_1) - \frac{1}{3} f_X(x|C_2)$

Figure 26: Total Probability Calculations & Decision Boundary Identification




```
sample mean: X_bar = -0.9813781693935629 , sample standard deviation: s = 1.9977389276092063
mean and variance should be:
 population mean: Mean = -1 , population standard deviation: StdDev = 2.0
```

Figure 27: Generation of N(-1,4)

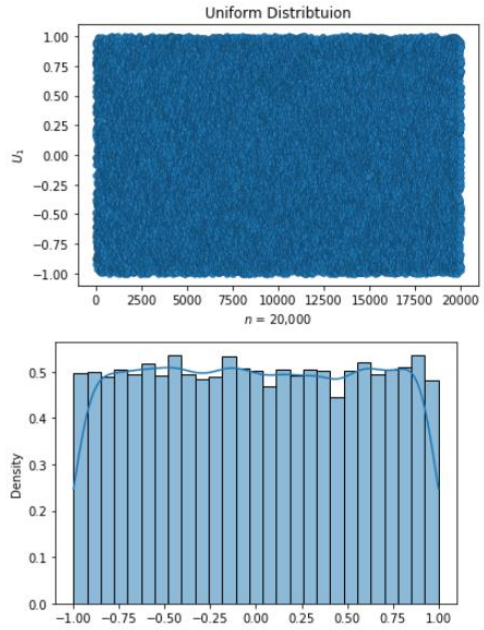


Figure 28: Generation of U(-1,1)

Decision Boundary:

$$h(x) = h_1(x) - h_2(x)$$

- $h(x) = \frac{2}{3} f_X(x|C_1) - \frac{1}{3} f_X(x|C_2)$

$$g(x) = g_1(x) - g_2(x)$$

Figure 29: Identification of Decision Boundary

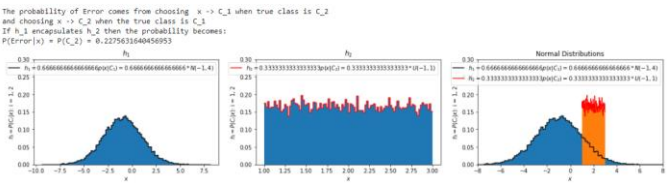


Figure 30: Overlap of Generated Mixed Distributions

class 1: $h_1 = (2/3) * N(-1, 4)$

class 2: $h_2 = (1/3) * U(-1, 1)$

From the plot combined plot we see the probability of error arises by choosing $x- > C_1$ when the true class is C_2

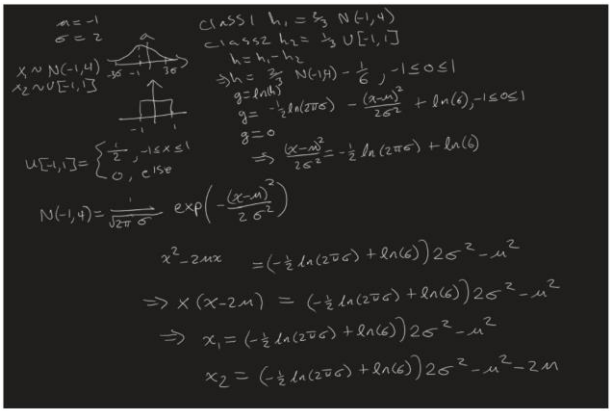
Figure 31: Discriminant Function Identification of Misclassification

From Figure 30 and 31 we see the misclassification is by choosing class 1 (i.e., h_1 = (1/3) * N(-1,4)) when the true class is class 2 (i.e., h_2 = (2/3) * U(-1,1)). This occurs because class 2 has the highest probability for the duration of the uniform distribution between -1 and 1. So the Probability of error will be the area under h_1 = (1/3) * N(-1,4) between x= -1 and 1. We can see in Figure 30, the probability of error will be the area highlighted in orange under the black plot line of the gaussian between x= -1 and 1.

$$P(Error) = P(\epsilon) = \int f_X(x|C_1)P(C_1)dx = P(C_1)N(-1,4) = \left(\frac{2}{3}\right)\int_{-1}^{1} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} = \left(\frac{2}{3}\right)\int_{-1}^{1} \frac{1}{\sqrt{2\pi}(2)} e^{-\frac{1}{2}\frac{(x-(-1))^2}{(2)^2}} = 0.22756.$$

The following calculations are for the decision boundary on the mixed distributions, but the points were outside the duration of the uniform (-1,1) distribution. So the computed Probability of Error below should be the same as the calculated Probability of Error just found above.

Figure 32: Probability of Error Misclassification



From the picture presented we can see the two theoretical boundary thresholds x_1 and x_2

Figure 33: Theoretical Derivation of Decision Boundary

```
x_1 = 1.2099787659472767
x_2 = - 1.2099787659472767

P(Error|x) = 0.2275631640456953
```

Figure 34: Calculated Decision Boundaries

The probability of error computed in figure 33 and 34 turned out to be the same as the calculated probability of error: $\int f_X(x|C_1)P(C_1)dx = N(-1,4)P(C_1) = 0.22756.$

V. Part 4: Multidimensional Simulation and Classification

This project will simulate Gaussian random variables in multiple dimensions. First, we will generate a random variable following the standard normal distribution$X\sim N(0,1)$, using the Box-Muller approach. In order to generate a generalized normal, $Y\sim N(\mu,\sigma 2)$, we use the transformation $Y=\sigma X+\mu$ with the desired mean μ and standard deviation σ. Appendix at contains important results in Linear Algebra and their applications to generate realizations from multinormal random Gaussian distribution. The Whitening approach enables us to simulate multinormal distribution in two steps: i) generate a standard Gaussian $N(0,I)$, where I is an identity matrix, then generate $N(0,\Sigma)$, with a covariance matrix Σ We can then generate $N(\mu,\Sigma)$ by adding the mean vector μ.

V.a. 4.1. SIMULATION OF RANDOM VARIABLES IN MULTIDIMENSIONAL SPACE

V.a.1. EXPERIMENT ONE

In this experiment, we generate a Gaussian distribution of zero mean and unit variance using the Box-Muller approach.

Task 1: Generate a generalization from standard Gaussian using the Box-Muller algorithm.

- Step 1: Generate streams of standard uniform $U(0,1)$ using the random number generator in the computer.
- Step 2: Using the Box-Muller approach we generate a standard Gaussian $N(0,1$) from the streams in Step 1, using the transformation:

$Y1=\sqrt{-2\ln X1}\cos 2\pi X2$ and $Y2=\sqrt{-2\ln X1}\sin 2\pi X2$

where $X1,X2\sim U(0,1)$ will result in $Y1,Y2\sim N(0,1)$.

Note: We can generate two streams ($X1$ and $X2$) of $U(0,1)$ to compute the Gaussian stream ($Y1$ and $Y2$) concatenated, or as one stream, where the first values would $X1$ and the second value would $X2$. We can also generate a stream of length $2n$ and consider $X1=\{x1i\},i\in[1,n]$ and $X2=\{xnj\},j\in[n,2n]$. Likewise, we can consider $X1=\{x1i\},i\in[1,2n]$ is even and $X2=\{x1j\},j\in[1,2n]$ is odd.

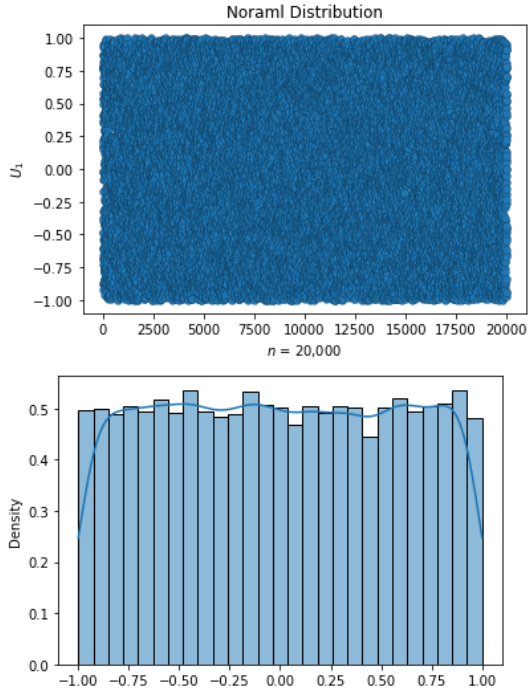
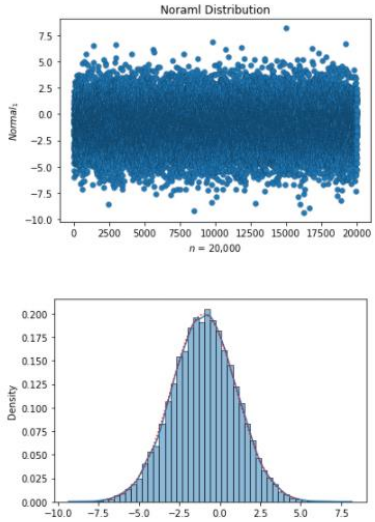


Figure 35: Generation of U(0,1)




```
sample mean: X_bar = -0.9813781693935629 , sample standard deviation: s = 1.9977389276092063
mean and variance should be:
 population mean: Mean = -1 , population standard deviation: StdDev = 2
```

Figure 36: Generation of N(0,1) from Box Muller Method as Shown in Figure 7

The samples are normally distributed with a mean of zero and unit variance. The histograms are identically similar. So similar in fact, that the dotted red line is hardly visible in Figure 36 since the generated gaussian Kernel Density Estimation (KDE) line follows the same plot as the realization plot. However, it should be clear that there are two plotted lines in Figure 36 that are generated differently. One was a randomly generated sample (the KDE) and the other was a realized form from the library of SciPy statistics. Figure 37 also shows the realization from SciPy statistics.

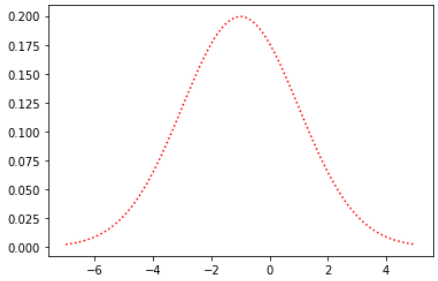


Figure 37: Generated Realization of N(-1,4)

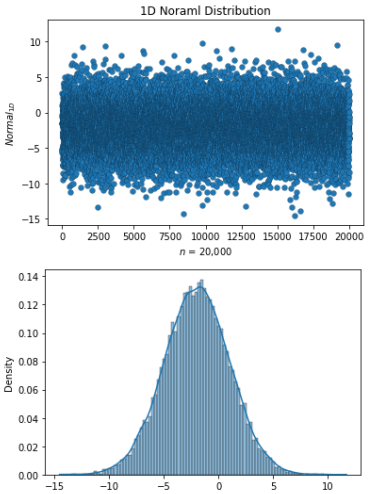
V.a.2. EXPERIMENT TWO

Using the random number generator (refer to experiment one) you can generate a uniform stream of 20,000 samples that follows U(0,1). Using the Box-Muller approach (refer to experiment two), you can convert the generated uniform stream to follow a standard normal N(0,1) . Now we can use the method of eigen values and eigen vectors described in Appendix in Chapter 4 to generate samples from N(μ, Σ). Then you can estimate the mean and covariance matrix of the resulted stream using the built-in Matlab function *mean* and *cov*. This is to validate the quality of the estimate. Of course, parameter estimation is an important subject that has been well-studied in probability theory.

Task 3: use the streams generated in task 2 to do the following:

(a) Convert the 1D normal variate to a normal distribution $N(-2,3)$; i.e., $\mu = -2$ and $\sigma = 3$.

(b) Convert the 2D normal variate to a normal distribution with mean vector $M = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ and Covariance matrix $\sum = \begin{bmatrix} 4 & 4 \\ 4 & 9 \end{bmatrix}$.




```
sample mean: X_bar = -1.9720672540903443 , sample standard deviation: s = 2.9966083914138095
mean and variance should be:
 population mean: Mean = -2 , population standard deviation: StdDev = 3
```

Figure 38: Generation of 1 Dimensional N(-2,9)

```
sigma =

[ 4 , 4 ]
[ 4 , 9 ]

M =

[ 1 ]
[ 2 ]

Normal_2D =

 [-1.1130540566435017, -1.216131017447495, 0.14058689585119702,
 [-1.1130540566435017, -1.216131017447495, 0.14058689585119702,
```

Figure 39: Normal 2-Dimensional Distribution Before Multivariate Transformation

```
result =

[-8.90443245 -9.72904814  1.12469517 ... -2.6440743  -2.3944245
  1.24345369]
[-14.46970274 -15.80970323   1.82762965 ...  -4.29662074  -3.89093982
   2.02061225]

 X =

 [[ -7.90443245  -8.72904814   2.12469517 ...  -1.6440743   -1.3944245
    2.24345369]
 [-12.46970274 -13.80970323   3.82762965 ...  -2.29662074  -1.89093982
    4.02061225]]
```

Figure 40: Results of Matrix Multiplication & then Matrix addition utilizing Matrix Multiplication Algorithm

(c) Convert the 3D normal variate to a normal distribution with mean vector $M = \begin{bmatrix} 5 \\ -5 \\ 6 \end{bmatrix}$ and Covariance matrix $\sum = \begin{bmatrix} 5 & 2 & -1 \\ 2 & 5 & 0 \\ -1 & 0 & 4 \end{bmatrix}$.

(d) For (a)(b) and (c) estimate the mean and covariance matrix and compare the estimated values with the groundtruth ones, comment.

```
sigma =

[ 5 , 2 , -1 ]
[ 2 , 5 , 0 ]
[ -1 , 0 , 4 ]

M =

[ 5 ]
[ -5 ]
[ 6 ]

Normal_3D =

 [-1.1130540566435017, -1.216131017447495, 0.14058689585119702, -1.0211424953186188,
 [-1.1130540566435017, -1.216131017447495, 0.14058689585119702, -1.0211424953186188,
 [-1.1130540566435017, -1.216131017447495, 0.14058689585119702, -1.0211424953186188,
```

Figure 41: Normal 3-Dimensional Distribution Before Multivariate Transformation

```
X =

[[ -1.67832434  -2.2967861    5.84352138 ...   3.01694428   3.20418162
   5.93259027]
[-12.7913784  -13.51291712  -4.01589173 ...  -7.31356501  -7.09512144
  -3.91197802]
[  2.66083783   2.35160695   6.42176069 ...   5.00847214   5.10209081
   6.46629513]]
```

Figure 42: Multivariate Affine Transformation of 3-Dimensional Gaussian Distribution

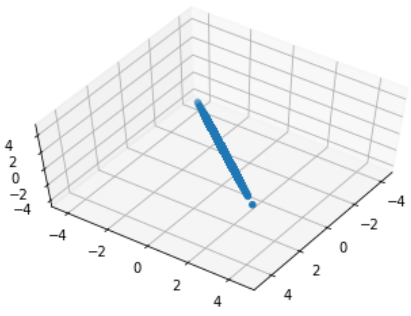


Figure 43: 3-D plot of 3-D Gaussian seems to be a 4-D shape when changing view of 3-D object.

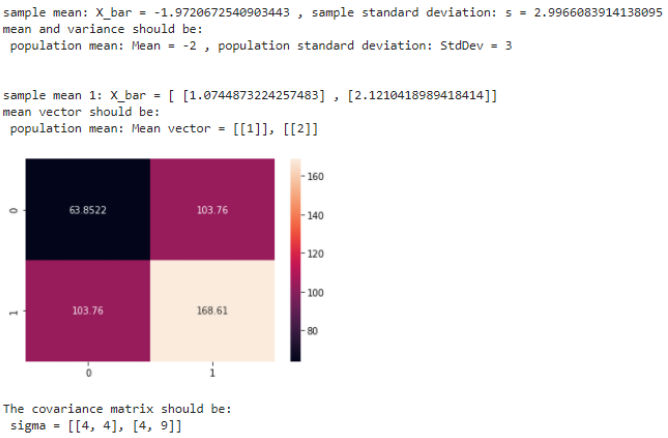


Figure 44: Mean Vector and Covariance Matrix Samples from Generated 1-D and 2-D Gaussian Distributions

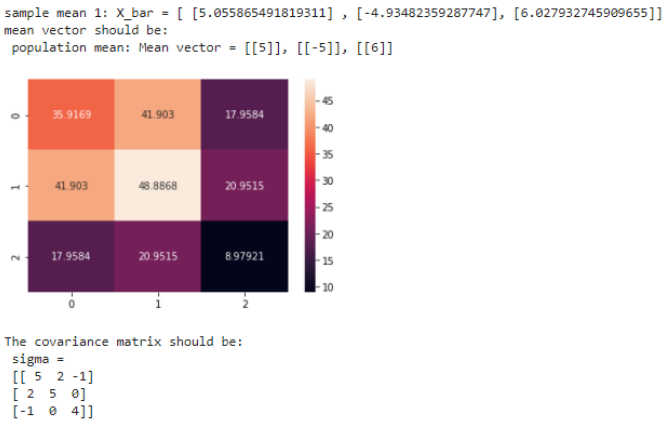


Figure 45: Mean Vector and Covariance Matrix samples from Generated 3-D Gaussian Distributions

Something was seriously wrong here. So, after using the built in NumPy generation of the normal distributions the affine transformation worked. While working through this issue another covariance decomposition was discovered called the Cholesky decomposition. The Cholesky decomposition works well for this case because the covariance matrix is a Hermitian positive-definite matrix. The covariance is decomposed into a lower triangular matrix and its' conjugate transpose (i.e. $\Sigma = LL^H = LL^T$, H is the Hermitian transpose or conjugate transpose and L is the lower triangular matrix).

We should mention that there are two functioning decompositions of the covariance matrix that will allow for the affine transformation. The first is the eigen value decomposition. The second and much more simple method is the Cholesky decomposition. This decomposition can separate the covariance matrix into a lower left triangular off diagonal and its Hermitian conjugate (i.e. complex conjugate) or the upper right triangular off diagonal and its Hermitian conjugate (i.e. $\Sigma = LL^H = R^H R$)

$$\Sigma = LL^H = \begin{bmatrix} 0 & 0 & 0 \\ 1+j & 0 & 0 \\ 2+j & 3+j & 0 \end{bmatrix} \begin{bmatrix} 0 & 3-j & 2-j \\ 0 & 0 & 1-j \\ 0 & 0 & 0 \end{bmatrix} \quad (6)$$

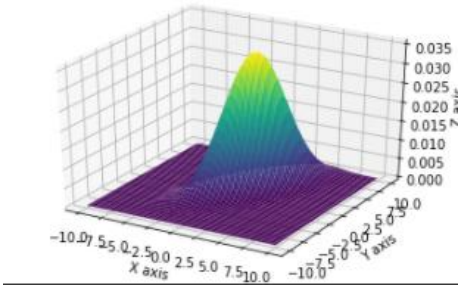
$$\Sigma = R^H R = \begin{bmatrix} 0 & 0 & 0 \\ 1+j & 0 & 0 \\ 2+j & 3+j & 0 \end{bmatrix} \begin{bmatrix} 0 & 3-j & 2-j \\ 0 & 0 & 1-j \\ 0 & 0 & 0 \end{bmatrix} \quad (7)$$

The Hermitian conjugate is both a Transpose of a matrix and the complex conjugate of the elements of that matrix. The Cholesky decomposition of the right diagonal and left diagonal are equivalent. Every Hermitian positive-definite matrix has a unique Cholesky decomposition (i.e., every real-value symmetric positive-definite matrix has a unique Cholesky decomposition) [Golub, 3].

After the matrix decomposition we can use the affine transformation as normal instead of finding an eigen value decomposition to do column by column matrix multiplication of the normal gaussian distribution streams. The results of the Cholesky decomposition are as follows:

```python
# (b) Convert the 2D normal variate to a normal distribution with mean vector

sigma_1 = np.array([[4, 4],[4, 9]]) # covariance matrix
print('sigma = \n')
print('[', sigma_1[0][0], ',', sigma_1[0][1], ']')
print('[', sigma_1[1][0], ',', sigma_1[1][1], ']')

M_1 = np.array([[1],[2]]) # mean vector M
print('\nM = \n')
print('[', M_1[0][0], ']')
print('[', M_1[1][0], ']')

mean = np.matrix(M_1)
covariance = np.matrix(sigma_1)

# Create L
L = np.linalg.cholesky(covariance)

d = 2 # Number of random variables
n = 20000  # Samples to draw
X = np.random.normal(size=(d, n))

# Apply the transformation
Y = L.dot(X) + mean

print("\n M sample= \n\n", np.mean(Y, axis = 1))

print("\n Σ sample= \n\n", np.cov(Y))

# Plot the samples and the distribution
fig, ax = plt.subplots(figsize=(6, 4.5))
# Plot bivariate distribution
x1, x2, p = generate_surface(mean, covariance, d)
con = ax.contourf(x1, x2, p, 100, cmap='rainbow')
# Plot samples
ax.plot(Y[0,:], Y[1,:], 'ro', alpha=.6,
        markeredgecolor='k', markeredgewidth=0.5)
ax.set_xlabel('$y_1$', fontsize=13)
ax.set_ylabel('$y_2$', fontsize=13)
ax.axis([-15, 15, -9, 21])
ax.set_aspect('equal')
ax.set_title('Samples from bivariate normal distribution')
cbar = plt.colorbar(con)
cbar.ax.set_ylabel('density: $p(y_1, y_2)$', fontsize=13)
plt.savefig('sampling_from_multivariate_normal_distribution')
plt.show()
```

```
sigma =

[ 4 , 4 ]
[ 4 , 9 ]

M =

[ 1 ]
[ 2 ]

 M sample=

 [[0.99962018]
 [2.01238012]]

 Σ sample=

 [[4.09335513 4.11027735]
 [4.11027735 9.08403528]]
```



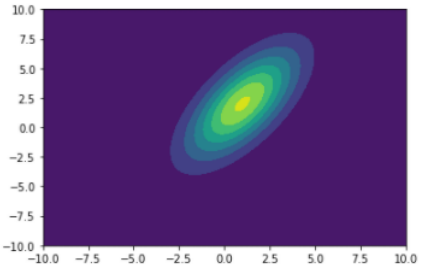


Figure 46: Cholesky Decomposition for 2D Affine Transformation

```python
# (c) Convert the 3D normal variate to a normal distribution with mean vector

sigma = np.array([[5, 2, -1],[2, 5, 0], [-1, 0, 4]]) # covariance matrix
print('sigma = \n')
print('[', sigma[0][0], ',', sigma[0][1], ',', sigma[0][2], ']')
print('[', sigma[1][0], ',', sigma[1][1], ',', sigma[1][2], ']')
print('[', sigma[2][0], ',', sigma[2][1], ',', sigma[2][2], ']')

M = np.array([[5],[-5], [6]]) # mean vector M
print('\nM = \n')
print('[', M[0][0], ']')
print('[', M[1][0], ']')
print('[', M[2][0], ']')

Normal_3D = np.array(Normal_3D)

mean = np.matrix(M)
covariance = np.matrix(sigma)

# Create L
L = np.linalg.cholesky(covariance)

# something was off with the generation of the normal distribution
# but by using the numpy generation the chelosky and eigenvalue decomposition for
# the affine transformation of the bivariate and multivartiate distributions worked

d = 3 # Number of random variables
n = 20000  # Samples to draw
X = np.random.normal(size=(d, n))

# Apply the transformation
Y = L.dot(X) + mean

print("\n M sample= \n\n", np.mean(Y, axis = 1))

print("\n Σ sample= \n\n", np.cov(Y))
```

```
sigma =

[ 5 , 2 , -1 ]
[ 2 , 5 , 0 ]
[ -1 , 0 , 4 ]

M =

[ 5 ]
[ -5 ]
[ 6 ]
[[ 2.23606798  0.          0.        ]
 [ 0.89442719  2.04939015  0.        ]
 [-0.4472136   0.19518001  1.93956303]]

 M sample=

 [[ 5.03697374]
 [-4.98436861]
 [ 6.01576588]]

 Σ sample=

 [[ 4.93153651  1.99597272 -1.00416127]
 [ 1.99597272  4.99356215 -0.01792522]
 [-1.00416127 -0.01792522  3.99465769]]
```

Figure 47: Cholesky Decomposition for 3D Affine Transformation

Now we should compare the Cholesky decomposition to the eigen value decomposition as in the following Figures:

```
# Calculate the eigenvalues and right eigenvectors of A.
[D, V] = np.linalg.eig(sigma); D = np.array(D*np.identity(3)) # eigen values, eigen vector
V = np.array(V)
# print(V)
# print(D)

# Verify that the results satisfy A*V = V*D.
# print(np.matmul(sigma,V) - np.matmul(V,D))
# e-16 accuracy is close enough

Y = np.array([np.matmul(np.matmul(V,np.sqrt(D)), np.reshape(j, (3, 1))) for j in X.T]).T
Yn = np.squeeze(Y);
Ym = Yn + mb.repmat(M, 1, 20000)

print("\n M sample= \n\n", np.mean(Ym, axis = 1))

print("\n Σ sample= \n\n", np.cov(Ym))
```

```
M sample=

[ 4.99469082 -5.02832144  5.97987898]

Σ sample=

[[ 5.02423079  2.07119758 -0.97377204]
 [ 2.07119758  5.08259958 -0.01310223]
 [-0.97377204 -0.01310223  3.93501104]]
```

Figure 48: Eigen Value Decomposition for 3D Affine Transformation

It seems that the results of the eigen value decomposition and the Cholesky decomposition combined with the affine domain transformations produce similar results. Now we can move onto the plotting of the transformed distributions and class discriminant functions for the gaussian distributions.

By setting up bivariate Mean vectors and covariance matrices we can visualize the behavior of the 2D individual contours. And by doing so, we can visualize the level setting procedure that the decomposition forms into elliptical structures. We will first show the contours before affine transformation and compare the contours after transformation.

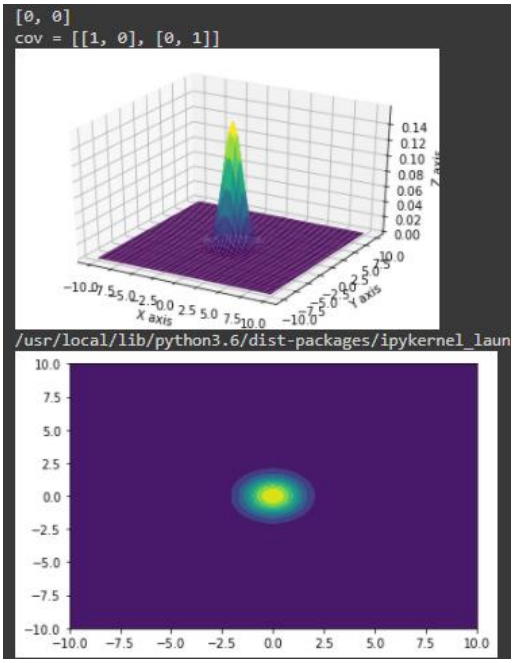


Figure 49: 2D Contour plot of standard Gaussian components

From Figure 49 only one contour needs to be plotted because the contours between dimensions 1-2, 1-3 and 2-3 will all be a standard normal gaussian.

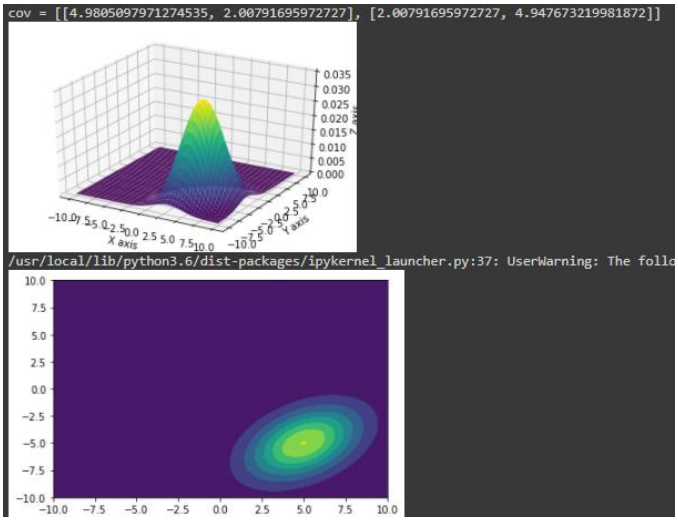


Figure 50: 2D Contour plot of 3D Affine Transformation for 1st and 2nd Dimension

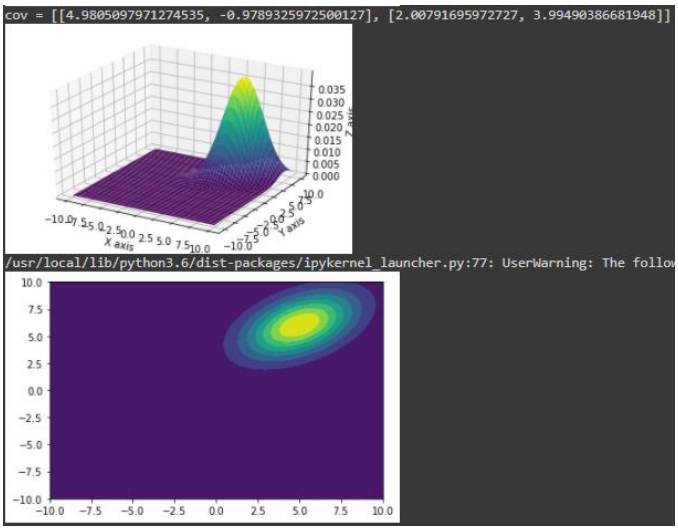


Figure 51: 2D Contour plot of 3D Affine Transformation for 1st and 3rd Dimension

In each of these contour graphs the sampled mean and variance are presented at the top of the graph just before the 3D bell curve and the contours.

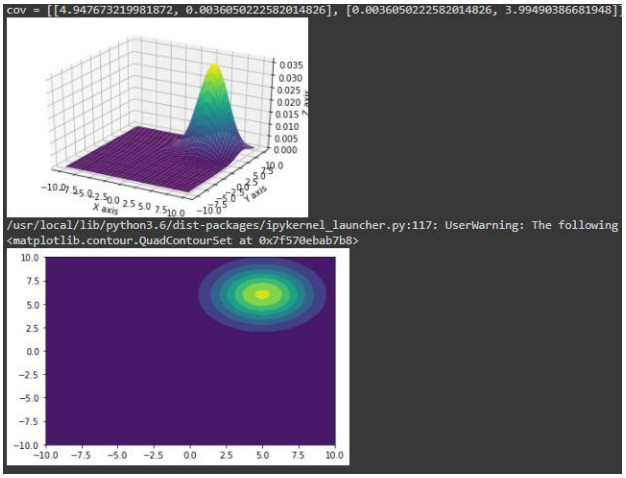


Figure 52: 2D Contour plot of 3D Affine Transformation for 2nd and 3rd Dimension

Next, if we plot all the contours on the same plot, we can see the three distinct locations:

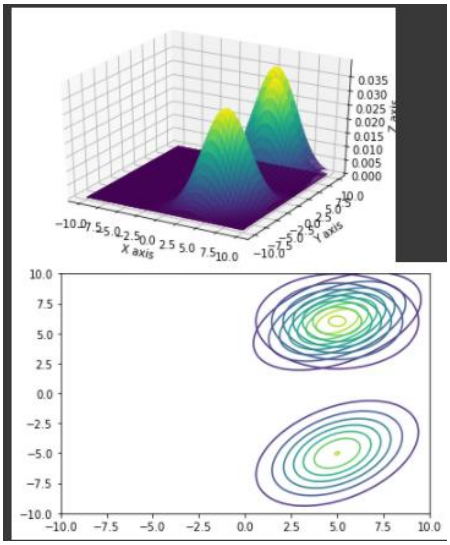


Figure 53: Combined plot of all Three Affine Transformed Dimensions

We can see right away from Figure 53 we create three distinct locations of the transformed gaussians. The contours of the 3-dimensional gaussian distributions form 2d ellipses. The equation of an ellipse is:

$$\left(\frac{x}{\sigma_x}\right)^2 + \left(\frac{y}{\sigma_y}\right)^2 = s \tag{8}$$

Where s is the scale of the ellipse. The scale can be set to s = 5.991 to represent a 95% confidence interval (i.e., P(s<5.991) = 1 - 0.05 = 0.95). Therefore, we can set the major axis length (x-axis) to $2\sigma_x\sqrt{5.991}$ and the minor axis length to $2\sigma_y\sqrt{5.991}$.

We can also say the major and minor axis length can be set based on the eigen value decomposition of the covariance matrix. Where the major axis length would be $2\sqrt{5.991\lambda_1}$ and the minor axis length would be $2\sqrt{5.991\lambda_2}$. To calculate the orientation of the ellipse, we just calculate the angle of the largest eigenvector towards the major axis (x-axis) [Spruyt, 6].

$$\alpha = \tan^{-1}\frac{\boldsymbol{v}_1(y)}{\boldsymbol{v}_2(x)} \tag{9}$$

V.b. 4.2. BINARY CLASSIFICATION ON MULTIDIMENSIONAL NORMAL DISTRIBUTIONS

Consider the problem of classification of data from normal distributions with mean and covariance defined as follows:

$$\boldsymbol{\mu}_i = \begin{bmatrix}\mu_{i1}\\ \mu_{i2}\\ \vdots\\ \mu_{iN}\end{bmatrix}, \quad \sum_i = \begin{bmatrix}\sigma_{11} & \cdots & \sigma_{1N}\\ \vdots & \ddots & \vdots\\ \sigma_{N1} & \cdots & \sigma_{NN}\end{bmatrix} \tag{P4.1}$$

Where C is the number of classes and N is the dimension of the random vector $\boldsymbol{X}$ such that:

$$\mu_{ij} = E(X_{ij}), i \in [1, C], j \in [1, N] \tag{P4.2}$$

$$\sigma_{mn} = E((X_m - \mu_m)(X_n - \mu_n)), i \in [1, C], m, n \in [1, N] \tag{P4.3}$$

The generalized Gaussian density function is defined as follows:

$$f_{\boldsymbol{X}}(\boldsymbol{x}|c_i) = \frac{1}{2\pi^{N/2}|\sum_i|^{1/2}} \exp -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^t \sum_i^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) \tag{P4.4}$$

The discriminant function is defined as:

$$h_i(x) = ln\, g_i(x) = ln\, f_{\boldsymbol{X}}(\boldsymbol{x}|c_i)P(C_i) = -\frac{N}{2} ln\, 2\pi - \frac{1}{2}\ln|\sum_i| - \frac{1}{2}(x - \mu_i)^t \sum_i^{-1}(x - \mu_i) \tag{P4.5}$$

Similar to the one-dimensional case, we can define three distinct cases based on the covariance matrix $\sum_i$. In particular,

- Case 1: $\sum_i = \begin{bmatrix}\sigma_{11} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \sigma_{11}\end{bmatrix} = \sigma I$, where I is the identity matrix.

In this case, discriminant function will have the following form:

$$h_i(x) = w_i{}^t x + w_{i0} \tag{P4.6}$$

Where the parameters in Eq. P4.6 has the following definitions:

$$w_i = \frac{1}{\sigma^2}\boldsymbol{\mu}_i \tag{P4.7a}$$

$$w_{i0} = -\frac{1}{2\sigma^2}\boldsymbol{\mu}_i{}^t\boldsymbol{\mu}_i + ln\, P(C_i) \tag{P4.7b}$$

- Case 2: equal covariance matrix; i.e., $\sum_i = \begin{bmatrix}\sigma_{11} & \cdots & \sigma_{1N}\\ \vdots & \ddots & \vdots\\ \sigma_{N1} & \cdots & \sigma_{NN}\end{bmatrix} = \sum$

In this case, the discriminant function will also have the following form, as in Eq. 4.6

$$h_i(x) = w_i{}^t x + w_{i0} \tag{P4.6}$$

where

$$w_i = \sum{}^{-1}\boldsymbol{\mu}_i \tag{P4.8a}$$

$$w_{i0} = -\frac{1}{2\sigma^2}\boldsymbol{\mu}_i{}^t\sum{}^{-1}\boldsymbol{\mu}_i + ln\, P(C_i) \tag{P4.8b}$$

The hyperplane separating classes C_i and C_j has the following equation:

$$\boldsymbol{x_0} = \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j) - \frac{ln(P(C_i)/P(C_j))}{(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^t\sum^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)} \cdot (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) \tag{P4.9}$$

- Case 3: general covariance matrix; i.e., $\sum_i = \begin{bmatrix}\sigma_{11} & \cdots & \sigma_{1N}\\ \vdots & \ddots & \vdots\\ \sigma_{N1} & \cdots & \sigma_{NN}\end{bmatrix}$

The discriminant function will have a quadratic form:

$$h_i(x) = \boldsymbol{x}^t W_i \boldsymbol{x} + w_i{}^t \boldsymbol{x} + w_{i0} \tag{P4.10}$$

Where:

$$W_i = -\frac{1}{2}\sum_i^{-1} \tag{P4.11a}$$

$$w_i = \sum_i^{-1}\boldsymbol{\mu}_i \tag{P4.11b}$$

$$w_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i{}^t\sum_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2} ln\left|\sum_i^{-1}\right| + ln\, P(C_i) \tag{P4.11c}$$

Figure 54: Problem of Classification

Experiment:

a) Generate 20,000 sample points from three classes with normal distributions with me
vectors and covariance matrices as follows:

i) $\mu_1 = \begin{bmatrix}1\\-1\\0\end{bmatrix}; \mu_2 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}; \mu_3 = \begin{bmatrix}0\\-1\\0\end{bmatrix}$ and $\sum_i = 4\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$

ii) $\mu_1 = \begin{bmatrix}1\\-1\\0\end{bmatrix}; \mu_2 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}; \mu_3 = \begin{bmatrix}0\\-1\\0\end{bmatrix}, \sum_i = \begin{bmatrix}5&2&-1\\2&5&0\\-1&0&4\end{bmatrix}$

$P(c_1) = P(c_2) = 2P(c_3)$

iii) $\mu_1 = \begin{bmatrix}1\\-1\\0\end{bmatrix}; \mu_2 = \begin{bmatrix}-1\\1\\-1\end{bmatrix}; \mu_3 = \begin{bmatrix}0\\-1\\0\end{bmatrix}; \sum_1 = \begin{bmatrix}5&2&-1\\2&5&0\\-1&0&4\end{bmatrix} = \sum_2; \sum_3 = \begin{bmatrix}4&0&0\\0&4&0\\0&0&4\end{bmatrix}$

$P(c_1) = P(c_2) = 2P(c_3)$

In this experiment the Prior probabilities were calculated using the total probability theorem (i.e., Total Probability adds up to 1), and then the multivariate streams were generated and sampled for the mean and covariance matrix to confirm proper transformation of the gaussian multivariate. In the experiment statement above the subproblem a I, ii, and iii correspond the three cases of the covariance matrix for the multivariate gaussian as stated in chapter 2 of Pattern Recognition and Scene Analysis by Stork, Duda and Hart. The three cases are i) a constant diagonal covariance matrix with constant variance: $\Sigma_i = \sigma I$, ii) a constant covariance matrix: $\Sigma_i = \Sigma$, and a differing covariance matrix with very differing variances: $\Sigma_i =$ general covariance matrix. The sample mean vector and sample covariance matrix are presented to prove the successful generation of the affine transformed multivariate gaussian distributions:

$P(c1) = P(c2)=2P(c3)$

$P(c1) + P(c2) + P(c3) = 1$

$P(C_1) + P(C_1) + \frac{1}{2}P(C_1) = 1$

$\frac{5}{2}P(C_1) = 1$

$P(C_1) = \frac{2}{5}$

$P(C_2) = \frac{2}{5}$

$P(C_3) = \frac{1}{5}$

Figure 55: Total Probability Calculations

```
sigma =

[ 4 , 0 , 0 ]
[ 0 , 4 , 0 ]
[ 0 , 0 , 4 ]

M_1 =

[ 1 ]
[ -1 ]
[ 0 ]

M_2 =

[ -1 ]
[ 1 ]
[ -1 ]

M_3 =

[ 0 ]
[ -1 ]
[ 0 ]
```

Figure 56: Variable Assignments for case I: $\Sigma_i = \sigma I$

```
M_1 sample=

[[ 1.0080024 ]
[-0.98619823]
[ 0.00815026]]

M_2 sample=

[[-0.99103097]
[ 0.99936192]
[-1.02302562]]

M_3 sample=

[[ 0.02837669]
[-1.00897962]
[ 0.00126885]]

Σ sample=

[[ 3.97838352 -0.05571198  0.01133565]
[-0.05571198  3.98744567  0.01784311]
[ 0.01133565  0.01784311  4.0269393 ]]

Σ sample=

[[ 4.01154951  0.01829654  0.02936711]
[ 0.01829654  3.97382418 -0.02492525]
[ 0.02936711 -0.02492525  3.95896248]]

Σ sample=

[[3.94436596 0.04148854 0.01124126]
[0.04148854 3.93196313 0.02041342]
[0.01124126 0.02041342 3.95551477]]
```

Figure 57: 3-D Gaussian Generation of class 1,2 and 3 for case I: $\Sigma_i = \sigma I$

From here on, some of the work was done in Jupyter Notebook and some work was done in MATLAB. The reason for this is because the mesh grid functionality in matplot lib for python does not work the same way in MATLAB and should be developed to do the same. After searching for a solution to this plot for days without recourse we ultimately turn to the mesh grid functionality that MATLAB provides that is needed to plot these 4D objects as 3D contours.

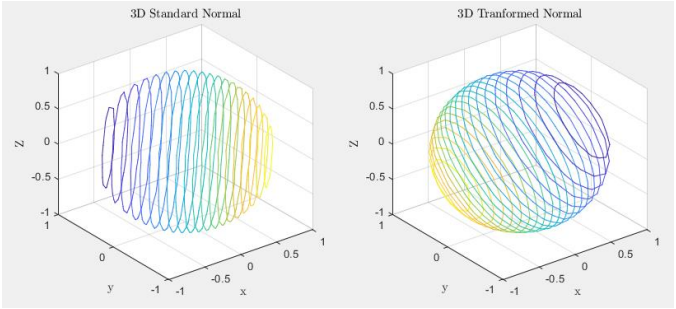


Figure 58: 3D contours of Multivariate Gaussian Distribution for case i: $\Sigma_i = \sigma I$

From Figure 57 we see the transformation and sampling for case I was successful! In Figure 58 we see the standard Normal Distribution plotted on the left and the affine transformation on the right. The affine transformation seems to have rotated the gaussian sphere. In the standard normal distribution the darker colors indicate outer regions of the gaussian bell shape as shown in the bivariate case with elliptical boundaries, and the

bright contours seem to be the higher value in the dimension. So, we can see that the higher value in the 4th dimension (i.e., the yellow) has rotated from the right in the standard normal to the bottom left in the affine transformed multivariate gaussian.

Next let us see case ii:

```
sigma =

[ 5 , 2 , -1 ]
[ 2 , 5 , 0 ]
[ -1 , 0 , 4 ]

M_1 =

[ 1 ]
[ -1 ]
[ 0 ]

M_2 =

[ -1 ]
[ 1 ]
[ -1 ]

M_3 =

[ 0 ]
[ -1 ]
[ 0 ]

Normal_3D =

 [[-1.11305406 -1.21613102  0.1405869  ... -0.33050929 -0.29930306
   0.15543171]]
 [[-1.11305406 -1.21613102  0.1405869  ... -0.33050929 -0.29930306
   0.15543171]]
 [[-1.11305406 -1.21613102  0.1405869  ... -0.33050929 -0.29930306
   0.15543171]]
```

Figure 59: Variable Assignments for case II: $\Sigma_i = \Sigma$

```
M_1 sample=

[[ 0.99922495]
[-0.99983865]
[-0.00612168]]

M_2 sample=

[[-0.98368184]
[ 1.00473868]
[-1.00083194]]

M_3 sample=

[[-0.01430241]
[-1.00390779]
[-0.00904387]]

Σ sample=

[[ 4.96478602e+00  1.99312315e+00 -9.85295472e-01]
[ 1.99312315e+00  5.09624910e+00  2.64727023e-03]
[-9.85295472e-01  2.64727023e-03  4.04040944e+00]]

Σ sample=

[[ 5.05926016  2.02970124 -0.97463365]
[ 2.02970124  4.98961194 -0.04643479]
[-0.97463365 -0.04643479  4.00452133]]

Σ sample=

[[ 4.96053615  2.01526473 -1.03112396]
[ 2.01526473  4.96784231 -0.0067038 ]
[-1.03112396 -0.0067038   4.00255879]]
```

Figure 60: 3-D Gaussian Generation of class 1,2 and 3 for case II: $\Sigma_i = \Sigma$

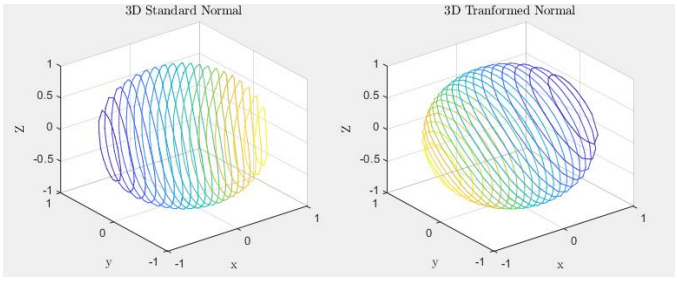


Figure 61: 3D contours of Multivariate Gaussian Distribution for case ii: $\Sigma_i = \Sigma$

From Figure 60 we see the transformation and sampling for case II was successful! In Figure 61 we see less rotation, but seems very similar to case i.

Next let us see case iii:

```
sigma_1 = sigma_2 = sigma =

[ 5 , 2 , -1 ]
[ 2 , 5 , 0 ]
[ -1 , 0 , 4 ]

sigma_3 =

[ 4 , 0 , 0 ]
[ 0 , 4 , 0 ]
[ 0 , 0 , 4 ]

M_1 =

[ 1 ]
[ -1 ]
[ 0 ]

M_2 =

[ -1 ]
[ 1 ]
[ -1 ]

M_3 =

[ 0 ]
[ -1 ]
[ 0 ]

Normal_3D =

 [[-1.11305406 -1.21613102  0.1405869  ... -0.33050929 -0.29930306
   0.15543171]]
 [[-1.11305406 -1.21613102  0.1405869  ... -0.33050929 -0.29930306
   0.15543171]]
 [[-1.11305406 -1.21613102  0.1405869  ... -0.33050929 -0.29930306
   0.15543171]]
```

Figure 62: Variable Assignments for case III: Σ_i = general covariance matrix

```
M_1 sample=

[[ 0.97946001]
[-1.02333789]
[-0.00277853]]

M_2 sample=

[[-1.01896216]
[ 0.98710227]
[-0.98993013]]

M_3 sample=

[[-0.02304597]
[-1.00763702]
[-0.02727356]]

Σ sample=

[[ 4.9674516   1.959861   -0.98297168]
[ 1.959861    4.9369761   0.01566   ]
[-0.98297168  0.01566     4.05034426]]

Σ sample=

[[ 5.04344640e+00  2.02826595e+00 -9.99722996e-01]
[ 2.02826595e+00  5.01988320e+00  1.23619186e-03]
[-9.99722996e-01  1.23619186e-03  4.02610301e+00]]

Σ sample=

[[ 3.99444457e+00 -3.89580465e-02 -1.67579175e-03]
[-3.89580465e-02  3.97234005e+00 -2.57047634e-02]
[-1.67579175e-03 -2.57047634e-02  4.00160254e+00]]
```

Figure 63: 3-D Gaussian Generation of class 1,2 and 3 for case III: Σ_i = general covariance matrix

From Figure 61 we see the transformation and sampling for case III was successful!

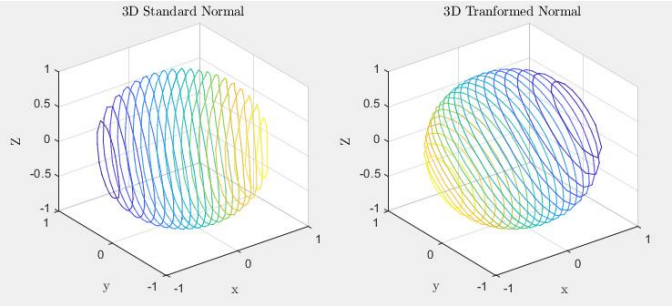


Figure 64: 3D contours of Multivariate Gaussian Distribution for case ii: Σ_i = general covariance matrix

In fact, all the affine transformations look very similar in the degree of rotation. But the degree of rotation is slightly different.

Now that the transformations for all three cases of the covariance matrix are complete, we can find the lossless boundary decision (i.e., g(x)). And compare the 3D contour plots of the 4-Dimensional gaussians created. The following boundary decisions are presented along with plots and discussion of the behavior of the decision boundary fir the three cases of the covariance matrix.

$$\mathbf{w}^t(\mathbf{x}-\mathbf{x_0})=0$$

$$\text{where } \mathbf{w}=\mu_i-\mu_j, \text{ and } \mathbf{x}_0=\frac{1}{2}(\mu_i+\mu_j)-\frac{\sigma^2}{\|\mu_i-\mu_j\|^2}\ln\frac{P(\omega_i)}{P(\omega_j)}(\mu_i-\mu_j)$$

Figure 65: Multivariate Boundary Decision Equations for case I: $\Sigma_i = \sigma I$

```
x_0_12 =
 [[ 1. ]
 [-1. ]
 [ 0.5]]

x_0_13 =
 [[-2.27258872]
 [ 0.        ]
 [ 0.        ]]

x_0_23 =
 [[-0.03790188]
 [ 0.07580376]
 [-0.03790188]]

w_12 =
 [[ 1. ]
 [-1. ]
 [ 0.5]]

w_13 =
 [[-2.27258872]
 [ 0.        ]
 [ 0.        ]]

w_23 =
 [[-0.03790188]
 [ 0.07580376]
 [-0.03790188]]
```

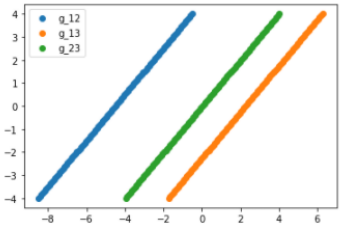


Figure 66: Linear Boundary Decision for case I: $\Sigma_i = \sigma I$

As shown in Figure 57 the x_0 shifts away from the more likely category where the more likely categories are classes C_1 and C_2.

$$\mathbf{w}^t(\mathbf{x}-\mathbf{x_0})=0$$

$$\text{where } \mathbf{w}=\Sigma^{-1}(\mu_i-\mu_j) \text{ and } \mathbf{x}_0=\frac{1}{2}(\mu_i+\mu_j)-\frac{ln[P(\omega_i)/P(\omega_j)]}{(\mu_i-\mu_j)^t\Sigma^{-1}(\mu_i-\mu_j)}(\mu_i-\mu_j)$$

Figure 67: Multivariate Boundary Decision Equations for case II: $\Sigma_i = \Sigma$

```
x_0_12 =
 [[ 0. ]
 [ 0. ]
 [-0.5]]

x_0_13 =
 [[-2.23793136]
 [-1.        ]
 [ 0.        ]]

x_0_23 =
 [[-0.17210403]
 [-0.65579194]
 [-0.17210403]]

w_12 =
 [[ 0.7721519 ]
 [-0.70886076]
 [ 0.44303797]]

w_13 =
 [[ 0.25316456]
 [-0.10126582]
 [ 0.06329114]]

w_23 =
 [[-0.51898734]
 [ 0.60759494]
 [-0.37974684]]
```

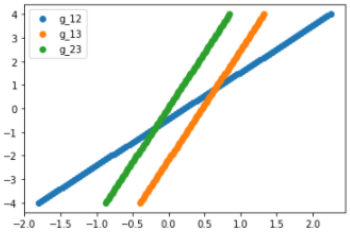


Figure 68: Linear Boundary Decision for case II: $\Sigma_i = \Sigma$

As shown in Figure 68 the x_0 shifts away from the more likely category where the more likely categories are classes C_1 and C_2. And the slope rotates the decision boundary clockwise to match the lossless ratio (i.e., ln(P_C_1or2/P_C_3)).

$$g_i(\mathbf{x}) = \mathbf{x}^t \mathbf{W}_i \mathbf{x} + \mathbf{w}_i \mathbf{x} + w_{i0}$$
(quadratic discriminant)

where $\mathbf{W}_i = -\frac{1}{2}\Sigma_i^{-1}$, $\mathbf{w}_i = \Sigma_i^{-1}\mu_i$, and $w_{i0} = -\frac{1}{2}\mu_i^t\Sigma^{-1}\mu_i - \frac{1}{2}ln|\Sigma_i| + \ln P(\omega_i)$

Figure 69: Multivariate Boundary Decision Equations for case III: Σ_i = general covariance matrix

When computing the quadratic decision boundary, it became apparent that the first term of the decision boundary became a matrix of size (sample size, sample size) where the sample size here was 20,000 samples. So, the quadratic term became a matrix of 20k-by-20k samples. To solve the quadratic discriminant functions for the 3 classes, need to be set equal and reduced to solve for the discriminant difference between the classes (i.e., g_1 = g_2, g_1 = g_3, g_2 = g_3).

```
w_10 =
 [[-3.44911592]]

w_20 =
 [[-3.67063491]]

w_30 =
 [[-3.81387945]]

w_1 =
 [[ 0.35443038]
 [-0.34177215]
 [ 0.08860759]]

w_2 =
 [[-0.41772152]
 [ 0.36708861]
 [-0.35443038]]

w_3 =
 [[ 0.  ]
 [-0.25]
 [ 0.  ]]

W_1 =
 [[-0.12658228  0.05063291 -0.03164557]
 [ 0.05063291 -0.12025316  0.01265823]
 [-0.03164557  0.01265823 -0.13291139]]

W_2 =
 [[-0.12658228  0.05063291 -0.03164557]
 [ 0.05063291 -0.12025316  0.01265823]
 [-0.03164557  0.01265823 -0.13291139]]

W_3 =
 [[-0.125 -0.    -0.   ]
 [-0.    -0.125 -0.   ]
 [-0.    -0.    -0.125]]
(20000, 20000)
```

```
print((X.T.dot(W_1.dot(X))).shape)
# g_1 = X.T.dot(W_1.dot(X)) + w_1*X + w_10
```

Figure 69: Quadratic Boundary Decision for case III: Σ_i = general covariance matrix

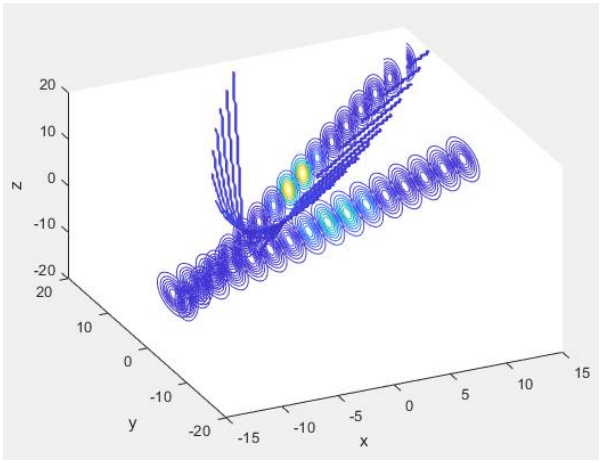


Figure 70: Quadratic Decision Boundary in MATLAB for Multidimensional Gaussian between classes 1 and 2

The Multidimensional classification shown in Figure 70 shows the case for general covariance matrices. Where the decision boundary is quadratic and the contour slices in high dimensionality shows a lower value along the 4th dimension between the higher valued 4th dimension values concentrated at the yellow centers of classes 1 and 2. From Figure 70 it becomes apparent that the elliptical pattern of the contours in higher dimensions seems to skew the contours perpendicularly to the quadratic decision boundary in opposite directions (i.e., if we draw a line along the angle created by the eigen vectors for each class seems to be at an angle of 90 degrees from the decision boundary.

Conclusion

The aim of this series of projects was to implement multidimensional Bayesian classification by using parametric class probability distributions. By finding the linear boundaries for cases I ($\Sigma_i = \sigma I$) and case II ($\Sigma_i = \Sigma$) we have achieved boundary lines for three multivariate gaussian distributed classes. The intersection of these lines will created decision planes where a value x can be assigned to the hyper-object gaussian class with a probability of error in choosing one class will be the multidimensional integration over the regions of the classes to which x was not chosen (i.e. choose x->C_1 will yield a probability of error over the regions of class C_2 and C_3 that are in the region divided by the planes belonging to C_1).

VI. References

[1] Farag, Introduction to Probability Theory with Engineering Applications, Cognella Academic Publishing, 2021.

[2] Farag, Biomedical Image Analysis Statistical and Variational Approaches.

[3] Golub, Gene Howard, and Van Loan Charles F. *Matrix Computations.* The Johns Hopkins University Press, 2013.

[4] Reuven Rubinstein and Dirk P. Kroese Simulation and the Monte Carlo Method, 3rd Edition, Wiley, New Jersey, 2017

[5] R. O. Duda, P. E. Hart and D. G. Stork: Pattern Classification, 2nd Edition, Wiley, 2000.

[6] Spruyt, Vincent. "How to Draw an Error Ellipse Representing the Covariance Matrix?" *Computer Vision for Dummies*, 22 Mar. 2015, www.visiondummy.com/2014/04/draw-error-ellipse-representing-covariance-matrix/.

[7] Zurada, Jacek M. *Introduction to Artificial Neural Systems*. West Publishing Company, 1992