Name:
Course: CAP 4601
Semester: Summer 2013
Assignment: Assignment 04
Date: 19 JUN 2013

Complete the following written problems:

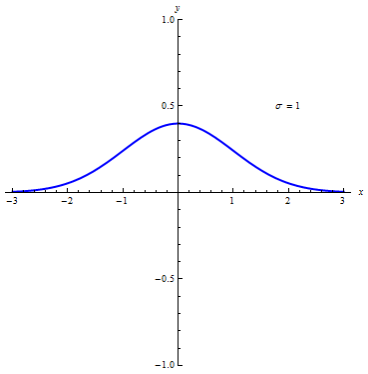
1. The Probability Density Function of the Normal Distribution (50 Points).

The Normal Distribution has the following Probability Density Function (a.k.a. the "Gaussian"):

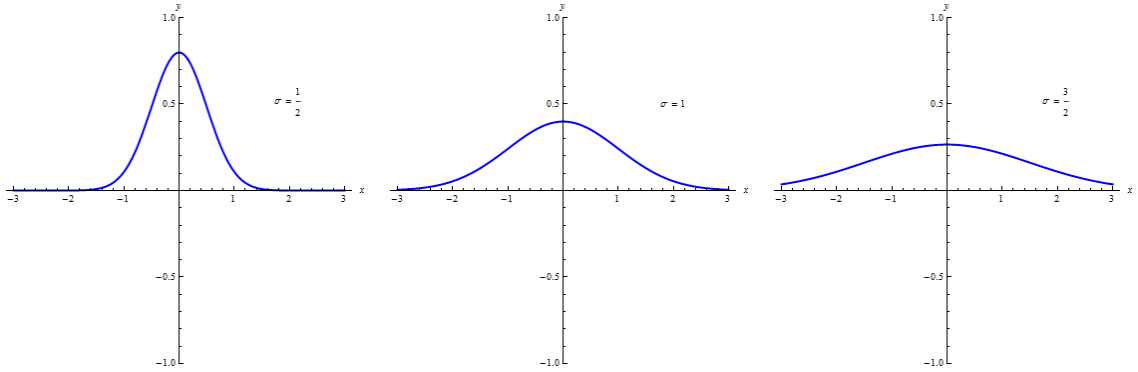
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where $\mu \in \mathbb{R}$ is the mean, $\sigma \in \mathbb{R}$ is the standard deviation, and $\sigma > 0$.

If the mean μ and the standard deviation σ are such that $\mu = 0$ and $\sigma = 1$, then we have the following bell-shaped curve:



Note: The mean μ translates this curve left or right; while the standard deviation σ makes this curve narrower or wider. For instance, the following plots show how this bell-shaped curve changes as σ changes. The plot on the left is for $\sigma = \frac{1}{2}$, the plot in the center is for $\sigma = 1$, and the plot on the right is for $\sigma = \frac{3}{2}$:



Therefore, as σ decreases, the bell-shaped curve shoots up (i.e. gets narrower). Similarly, as σ increases, the bell-shaped curve flattens out (i.e. gets wider).

Moreover, this function has the following property:

$$\int_{-\infty}^{\infty} f(x)\,dx = \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx$$

$$= \frac{1}{2} erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\Bigg|_{-\infty}^{\infty}$$

$$= \frac{1}{2} erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\Bigg|^{\infty} - \frac{1}{2} erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\Bigg|^{-\infty}$$

$$= \left(\frac{1}{2}\right) - \left(-\frac{1}{2}\right)$$

$$= 1$$

where erf is the Error function, $erf(x) = \int \frac{2}{\sqrt{\pi}} e^{-x^2}\,dx$, and $\frac{d}{dx}\left[erf(x)\right] = \frac{2}{\sqrt{\pi}} e^{-x^2}$.

Therefore, regardless of what the value is for μ and σ, the "area under the curve" for the entire function is always 1 for this function. This is a desirable property that we can take advantage of in Probability.

a. Given the function for the Normal distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Derive the x values that production inflection points in the function above. In other words, using Calculus and Algebra, find the x values that make $f''(x) = 0$ for any value of μ and σ.

b. C++11 added the Error function $erf(\ )$ shown above to the header `<cmath>` as the function `erf()`. The following block of code shows how to use this Error function:

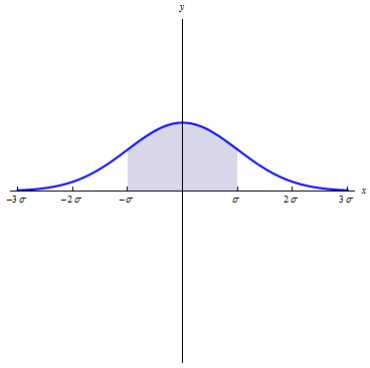
```
#include <iostream>
#include <cmath>

int main () {
        std::cout << erf( 1.0 ) << '\n'; // 0.842701
}
```

Use the indefinite integration above to calculate the exact value of the following definite integrals where $\mu = 0$. Exact value means keeping the square roots and reducing down to one erf function. Note: $erf(-a) = -erf(a)$. Also, use the C++11 `erf()` function to calculate the decimal values of those same definite integrals:

Definite Integral	Exact Value	Decimal Value
$\int_{-\sigma}^{\sigma} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$	$erf\left(\frac{\sqrt{2}}{2}\right)$	0.682689
$\int_{-2\sigma}^{2\sigma} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$		
$\int_{-3\sigma}^{3\sigma} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx$		

c. For these bell-shaped curves, what is the percentage of the "area under the curve" that is within one standard deviation from the mean? In other words, if we fixed $\mu = 0$, what is the blue shaded area in this plot as a percentage of the overall "area under the curve":



d. For these bell-shaped curves, how many standard deviations σ from the mean covers approximately 99.73% of the "area under the curve"?

2. Expectation Maximization using a Gaussian Mixture Model (50 Points).

OpenCV can perform Expectation Maximization using the `cv::EM` class. Here is an excerpt of the OpenCV code needed to perform Expectation Maximization to pull out two "Gaussians" (i.e. the multivariate version of the Normal distribution we covered above in Written Problem 1):

```
int max_number_of_iterations = 1024;
double threshold = 0.000001;
cv::TermCriteria termination_criteria(
        cv::TermCriteria::COUNT + cv::TermCriteria::EPS,
        max_number_of_iterations,
        threshold
);
int number_of_clusters = 2;
cv::EM em(
        number_of_clusters,
        cv::EM::COV_MAT_GENERIC,
        termination_criteria
);
em.train( dataset );
auto means = em.getMat( "means" );
std::cout << "means:\n" << means << "\n\n";
auto covs = em.getMatVector( "covs" );
std::cout << "covs:\n" << covs << "\n\n";
```

The OpenCV class `cv::TermCriteria` is used to set up the termination criteria (i.e. when to stop the EM algorithm). This is same as the termination criteria that we have used to stop gradient descent in previous assignments. Similarly, we are stopping here after either 1024 iterations of EM or when the relative change in the likelihood logarithm is under the threshold 0.000001.

The OpenCV enumeration value `cv::EM::COV_MAT_GENERIC` ensures that we will receive "Gaussians" that are both scaled and rotated (both of these, vice just one or the other). In other words, this enumeration value ensures that we will receive a full Covariance Matrix for each cluster it will try to fit. A covariance matrix functions in much the same way as the standard deviation functioned in Written Problem 1 above.

In the code excerpt above, we are requesting that "Gaussians" be fitted to two clusters of data; therefore, after we train on a dataset, we should receive two means and two covariance matrices.

These multivariate "Gaussians" (i.e. the Probability Density Function of the Multivariate Normal Distribution) have a similar function to that of the bell-shaped curve we saw in Written Problem 1 above:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\mathbf{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

where T in the exponent means matrix or vector transpose, $|\ |$ means the determinant, n is the input dimension such that $\mathbf{x} \in \mathbb{R}^n$, the mean $\boldsymbol{\mu} \in \mathbb{R}^n$, and the covariance matrix $\mathbf{\Sigma}$ is an $n \times n$

matrix such that $\boldsymbol{\Sigma} = (\mathbf{RS})(\mathbf{RS})^T$ where $\mathbf{R}$ is a $n \times n$ parametric rotation matrix and $\mathbf{S}$ is a $n \times n$ parametric scaling matrix. As in Written Problem 1, the "area under the curve" (or "volume under the curve") is 1. Note: Here $\boldsymbol{\Sigma}$ is a variable, not the summation symbol. Since we are dealing with the multivariate case here, we use the uppercase Greek letter S (uppercase sigma): $\boldsymbol{\Sigma}$. When we were dealing with the univariate case in Written Problem 1, we used the lowercase Greek letter s (lowercase sigma): σ.

For instance, if we have $\mathbf{x} = (x, y)$, a mean $\boldsymbol{\mu} = (4, -2)$, a parametric rotation matrix that $\mathbf{R}$ rotates the "Gaussian" by $\theta = 30°$, and a parametric scaling matrix $\mathbf{S}$ that scales the "Gaussian" by 4 in the x direction and 1 in the y direction prior to the rotation, then we would have the following preliminary calculations:

$$\mathbf{S} = \begin{pmatrix} 4 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} \cos(\theta) & \cos(\theta + 90°) \\ \sin(\theta) & \sin(\theta + 90°) \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} = \begin{pmatrix} \cos(30°) & -\sin(30°) \\ \sin(30°) & \cos(30°) \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{3}}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$$

$$\boldsymbol{\Sigma} = (\mathbf{RS})(\mathbf{RS})^T = \mathbf{RSS}^T\mathbf{R}^T = \underbrace{\mathbf{R}(\mathbf{SS})\mathbf{R}^T}_{\text{Singular Value Decomposition}} = \begin{pmatrix} \frac{49}{4} & \frac{15\sqrt{3}}{4} \\ \frac{15\sqrt{3}}{4} & \frac{19}{4} \end{pmatrix}$$

$$|\boldsymbol{\Sigma}| = \left| \begin{pmatrix} \frac{49}{4} & \frac{15\sqrt{3}}{4} \\ \frac{15\sqrt{3}}{4} & \frac{19}{4} \end{pmatrix} \right| = 16$$

$$|\boldsymbol{\Sigma}|^{\frac{1}{2}} = (16)^{\frac{1}{2}} = 4$$

$$\boldsymbol{\Sigma}^{-1} = \left(\mathbf{RSS}^T\mathbf{R}^T\right)^{-1} = \left(\mathbf{R}^T\right)^{-1}\left(\mathbf{S}^T\right)^{-1}\mathbf{S}^{-1}\mathbf{R}^{-1} = \underbrace{\mathbf{R}\left(\mathbf{S}^{-1}\mathbf{S}^{-1}\right)\mathbf{R}^T}_{\text{Singualr Value Decomposition}} = \begin{pmatrix} \frac{19}{64} & -\frac{15\sqrt{3}}{64} \\ -\frac{15\sqrt{3}}{64} & \frac{49}{64} \end{pmatrix}$$

$$\mathbf{x} - \boldsymbol{\mu} = \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} 4 \\ -2 \end{pmatrix} = \begin{pmatrix} x - 4 \\ y + 2 \end{pmatrix}$$

Then, we would have the following intermediate calculation:

$$(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}) = \begin{pmatrix} x-4 \\ y+2 \end{pmatrix}^T \begin{pmatrix} \frac{19}{64} & -\frac{15\sqrt{3}}{64} \\ -\frac{15\sqrt{3}}{64} & \frac{49}{64} \end{pmatrix} \begin{pmatrix} x-4 \\ y+2 \end{pmatrix}$$

$$= \begin{pmatrix} x-4 & y+2 \end{pmatrix} \begin{pmatrix} \frac{19}{64} & -\frac{15\sqrt{3}}{64} \\ -\frac{15\sqrt{3}}{64} & \frac{49}{64} \end{pmatrix} \begin{pmatrix} x-4 \\ y+2 \end{pmatrix}$$

$$= \frac{19x^2}{64} - \frac{15}{32}\sqrt{3}xy - \frac{15\sqrt{3}x}{16} - \frac{19x}{8} + \frac{49y^2}{64} + \frac{15\sqrt{3}y}{8} + \frac{49y}{16} + \frac{15\sqrt{3}}{4} + \frac{125}{16}$$

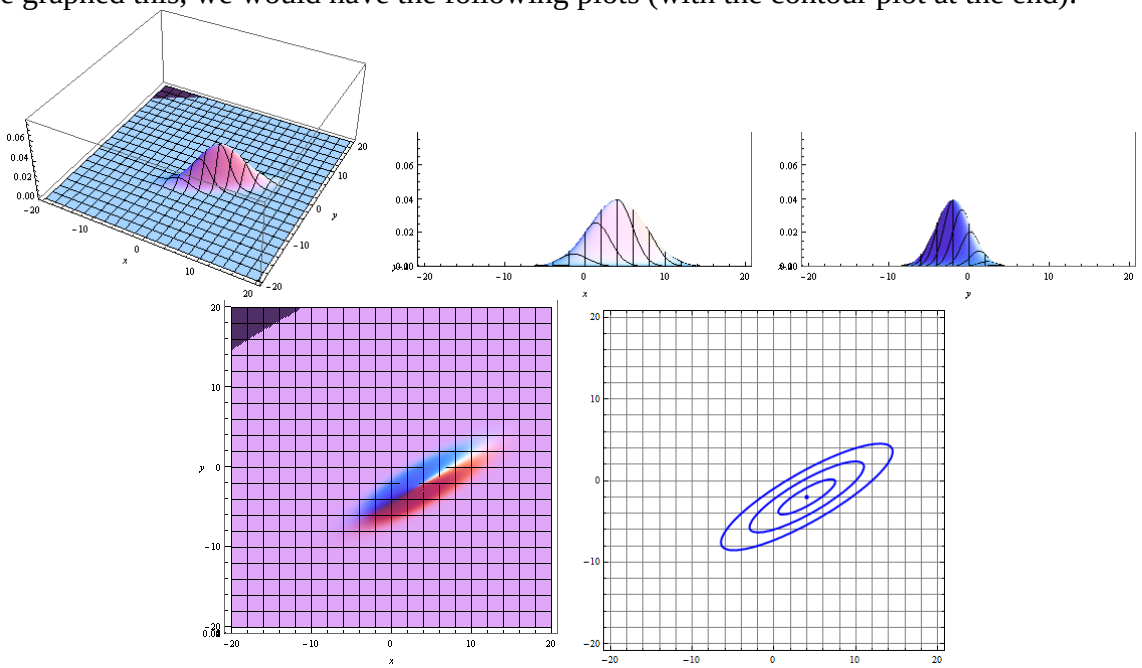
And finally, we would have:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu})}$$

$$f(x,y) = \frac{1}{(2\pi)^{\frac{2}{2}} (4)} e^{-\frac{1}{2}\left(\frac{19x^2}{64} - \frac{15}{32}\sqrt{3}xy - \frac{15\sqrt{3}x}{16} - \frac{19x}{8} + \frac{49y^2}{64} + \frac{15\sqrt{3}y}{8} + \frac{49y}{16} + \frac{15\sqrt{3}}{4} + \frac{125}{16}\right)}$$

$$= \frac{1}{8\pi} e^{-\frac{19x^2}{128} + \frac{15}{64}\sqrt{3}xy + \frac{15\sqrt{3}x}{32} + \frac{19x}{16} - \frac{49y^2}{128} - \frac{15\sqrt{3}y}{16} - \frac{49y}{32} - \frac{15\sqrt{3}}{8} - \frac{125}{32}}$$

If we graphed this, we would have the following plots (with the contour plot at the end):

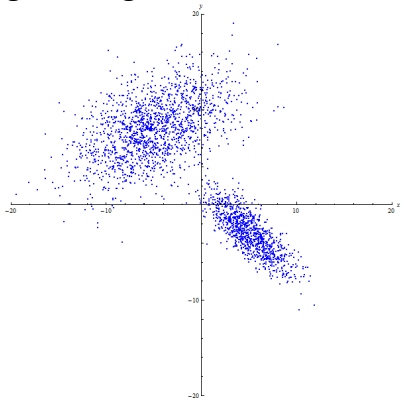


Note: The ellipses in the contour plot above are the first, second, and third "standard deviations" from the mean point at the center.

Note: Since we're dealing with ellipses, a $30°$ rotation for a scaled ellipse will look the same if the ellipse id rotated $180°$ in the opposite direction. In other words, a $30°$ rotation will look the same as a $-150°$ rotation.

That's a lot of math! Thank goodness the OpenCV library does most of this for us.

The data from Assignment 1, Programming Problem 5 was the following:



This data is not separated into classes; however, it does appear that there are two classes of points in this dataset – a class for each cluster of data. These clusters also appear to be generated from a distribution similar to the Multivariate Normal Distribution we just learned about. Why? Because these clusters are dense at their center, that density gradually falls off as we move away from their center, and the clusters are elliptical. Their center could be described by a mean (i.e. their translation). Their elliptical nature could be described by a covariance matrix (i.e. their scaling and rotation).

We need the EM algorithm to return the mean and covariance matrix of each of the clusters above so that we can separate those clusters into classes.

Once the EM algorithm returns the mean and covariance matrix of each cluster, the following OpenCV code extracts the elliptical information from each cluster (Note: `i` is the index of each cluster [`0` or `1`]):

```
// Singular Value Decomposition
cv::Mat_<double> U, W, Vt;
cv::SVDecomp( covs[ i ], W, U, Vt );

cv::Mat_<double> center = means.row( i );
cv::Mat_<double> semi_major_axis_direction = Vt.row( 0 );
double semi_major_axis_angle_in_radians = atan2(
                semi_major_axis_direction( 1 ),
                semi_major_axis_direction( 0 )
);
double semi_major_axis_angle_in_degrees = (
        semi_major_axis_angle_in_radians * degrees_per_radian
);
double semi_major_axis_magnitude = sqrt(  W( 0 ) );
```

```
cv::Mat_<double> semi_minor_axis_direction = Vt.row( 1 );
double semi_minor_axis_angle_in_radians = atan2(
        semi_minor_axis_direction( 1 ),
        semi_minor_axis_direction( 0 )
);
double semi_minor_axis_angle_in_degrees = (
        semi_minor_axis_angle_in_radians * degrees_per_radian
);
double semi_minor_axis_magnitude = sqrt( W( 1 ) );
```

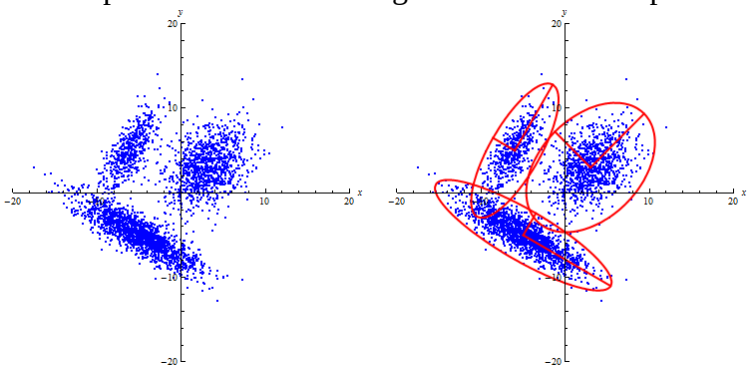
Compile and run the following code from em.zip:
– main.cpp: The file containing the OpenCV calls for the EM algorithm.
– data.cpp: The dataset from Programming Problem 5 of Assignment 01.
– makefile: The makefile for linprog4.cs.fsu.edu.

Using information about Cluster 0 from the output of the code above, the Probability Density Function of the Bivariate Normal Distribution that generated that first cluster of data (Cluster 0) is approximately:

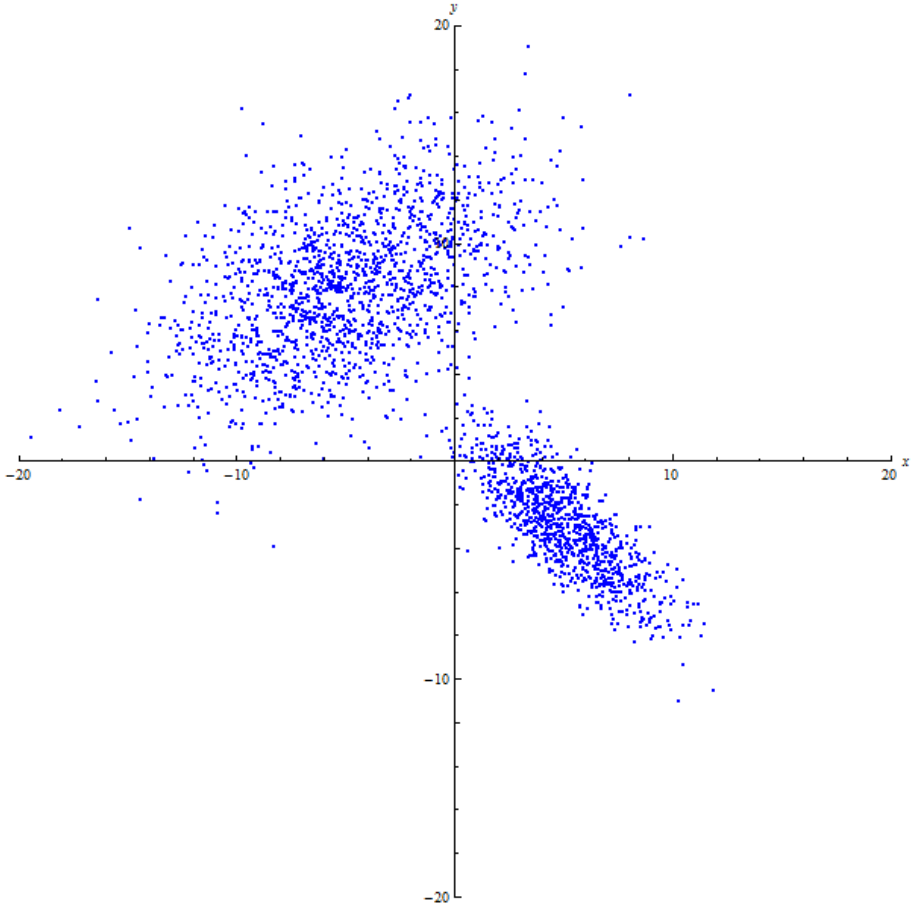
$$0.0140821e^{-0.0385986x^2+0.0492529xy-0.77767x-0.0664186y^2+1.30577y-7.14869}$$

a. Using information about Cluster 1 from the output of the code above, calculate the approximate Probability Density Function of the Bivariate Normal Distribution that generated that second cluster of data (Cluster 1).

b. Using the information about Ellipse 0 and Ellipse 1 from the output of the code above, draw the ellipse for each cluster that should contain 99.73% of the data. Draw these ellipses on the large picture below. Ensure that you include the line for semi-major axis and the semi-minor axis. Ensure that you pay close attention to the angle and the length of those semi-major and semi-minor axes. For example, given the initial plot of points on the left, you would produce the final plot of points with ellipses and axes on the right based on the output of the code above:



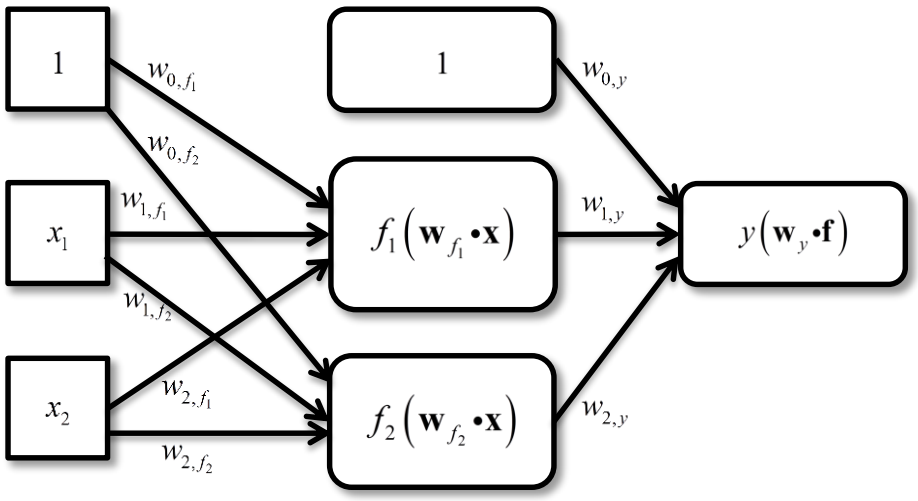
Draw the ellipses and axes on the following picture based on the output from the code above:



Keep in mind that this dataset did not contain class labels. This was an unlabeled dataset. We told the EM algorithm to find two clusters of points and it found two … without supervision … and the two clusters that the EM algorithm found happened to exactly match our expectations. Yay, Unsupervised Learning!

3. Neural Networks (100 Points).

Given the following Neural Network:



where $\mathbf{w}_{f_1}\bullet\mathbf{x}=w_{0,f_1}\cdot 1+w_{1,f_1}\cdot x_1+w_{2,f_1}\cdot x_2$, $\mathbf{w}_{f_2}\bullet\mathbf{x}=w_{0,f_2}\cdot 1+w_{1,f_2}\cdot x_1+w_{2,f_2}\cdot x_2$, $\mathbf{w}_y\bullet\mathbf{f}=w_{0,y}\cdot 1+w_{1,y}\cdot f_1+w_{2,y}\cdot f_2$, and activation functions f_1, f_2, and y are step functions defined as follows:

$$f_1\left(\mathbf{w}_{f_1}\bullet\mathbf{x}\right)=\begin{cases}\mathbf{w}_{f_1}\bullet\mathbf{x}\geq 0, & 1\\ \mathbf{w}_{f_1}\bullet\mathbf{x}<0, & 0\end{cases}$$

$$f_2\left(\mathbf{w}_{f_2}\bullet\mathbf{x}\right)=\begin{cases}\mathbf{w}_{f_2}\bullet\mathbf{x}\geq 0, & 1\\ \mathbf{w}_{f_2}\bullet\mathbf{x}<0, & 0\end{cases}$$

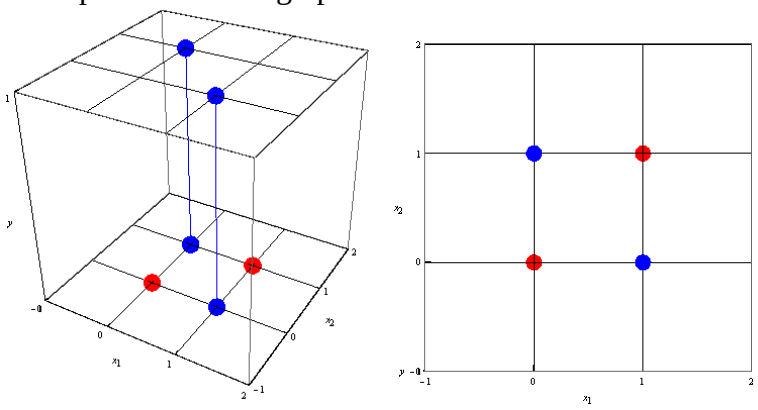
$$y\left(\mathbf{w}_y\bullet\mathbf{x}\right)=\begin{cases}\mathbf{w}_y\bullet\mathbf{x}\geq 0, & 1\\ \mathbf{w}_y\bullet\mathbf{x}<0, & 0\end{cases}$$

a. Place and orient the activation functions f_1, f_2, and y by hand and calculate the nine Neural Network parameters (i.e. w_{0,f_1}, w_{1,f_1}, w_{2,f_1}, w_{0,f_2}, w_{1,f_2}, w_{2,f_2}, $w_{0,y}$, $w_{1,y}$, and $w_{2,y}$) above needed to compute the XOR function. In other words, calculate the nine parameters needed to create this function:

x_1	x_2	y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

HINT: It will be very helpful to plot the (x_1, x_2) points, draw the lines $\mathbf{w}_{f_1} \bullet \mathbf{x} = 0$ and $\mathbf{w}_{f_2} \bullet \mathbf{x} = 0$ on that plot, and shade each positive region (i.e. where $f_1 = 1$ and $f_2 = 1$).
HINT: It will be very helpful to plot the (f_1, f_2) points, draw the line $\mathbf{w}_y \bullet \mathbf{f} = 0$ on that plot, and shade the positive region (i.e. where $y = 1$).

Note: The XOR function produces these graphs:



Remember that $\mathbf{w} \bullet \mathbf{x} = w_0 \cdot 1 + w_1 \cdot x_1 + w_2 \cdot x_2$ is just a plane that we can think of in terms of the normal vector equation that we saw in Assignment 01:

$$\begin{aligned}
\vec{\mathbf{n}} \bullet (\mathbf{x} - \mathbf{x}_0) &= (\cos(\theta), \sin(\theta)) \bullet ((x_1, x_2) - (x_{1,0}, x_{2,0})) \\
&= (\cos(\theta), \sin(\theta)) \bullet (x_1 - x_{1,0}, x_2 - x_{2,0}) \\
&= \cos(\theta)(x_1 - x_{1,0}) + \sin(\theta)(x_2 - x_{2,0}) \\
&= -\cos(\theta) x_{1,0} - \sin(\theta) x_{2,0} + \cos(\theta) x_1 + \sin(\theta) x_2 \\
&= (-\cos(\theta) x_{1,0} - \sin(\theta) x_{2,0}) 1 + (\cos(\theta)) x_1 + (\sin(\theta)) x_2
\end{aligned}$$

Therefore, $w_0 = -\cos(\theta) x_{1,0} - \sin(\theta) x_{2,0}$, $w_1 = \cos(\theta)$, and $w_2 = \sin(\theta)$. Note: We could also adjust the "slope" of that plane using the $\tan(\phi)$ as we saw in the Logistic Function problem from Assignment 03, but we won't need to do that here.

b. For the functions f_1, f_2, and y above, calculate the following tables:

x_1	x_2	f_1
0	0	
0	1	
1	0	
1	1	

x_1	x_2	f_2
0	0	
0	1	
1	0	
1	1	

f_1	f_2	y
		0
		1

Note: For that last table, you may add additional 0 or 1 values as needed for various combinations of f_1 and f_2.

Complete the following programming problems on `linprog4.cs.fsu.edu`:

Download the ZIP file containing the directory structure and files for these programming problems: `assignment 04.zip`

1. Cross Validation – Part 2 (200 Points):

Use either the "Cross Validation – Part 1" code you wrote for Assignment 03 or the following code:
– `main.cpp`: The file to be edited.
– `wdbc.data`: The Breast Cancer Wisconsin (Diagnostic) Data Set.
– `makefile`: The makefile for `linprog4.cs.fsu.edu`.

Do __not__ touch the Testing dataset that contains 5% of the malignant data and 5% of the benign data.

Program a k-fold Stratified Cross Validation:

Using just the Training dataset, divide the training dataset into class datasets (for this problem, a "malignant" dataset and a "benign" dataset).

(Optional) Shuffle each class dataset.

Divide each class dataset into k equal sets. For this programming problem, let $k = 10$; however, ensure that k is a variable that can change be easily changed.

THE START OF A FOLD OF CROSS VALIDATION.

Create a "Train" dataset and a "Validate" dataset. Copy 1 set from each class into the "Validate" dataset. Copy the remaining $k - 1$ sets from each class into the "Train" dataset.

(Required) Shuffle the "Train" dataset and then shuffle the "Validate" dataset that contains copies from each class dataset.

Choose a classifier from OpenCV's Machine Learning Library (MLL) that interests you. Train a classifier using the "Train" dataset. Use the classifier's `train()` method (if available). Validate that classifier's performance using the "Validate" dataset. Use the classifier's `predict()` method (if available). Use `std::cout` to report the parameters that were used for the classifier and the performance of the classifier. For this programming problem, the format of the output is not important.

For performance, use "Overall Accuracy":

$$\text{Overall Accuracy} = \frac{\text{the number of classes you predicted correctly}}{\text{the number of predictions you had to make}}$$

Store "Overall Accuracy" in a `std::vector` that is maintained throughout the life of the program.

THE END OF A FOLD OF CROSS VALIDATION.

From the k sets that each class dataset has been divided into, use a different set for your "Validate" set … the next set of the k sets in each class. Use the remaining sets to create your "Train" set. Perform another fold of Cross Validation.

Repeat this k times until every set of each class dataset has been used in the "Validate" set. In other words, do k-folds of this k-fold Stratified Cross Validation.

Example: If $k = 3$, then the "malignant" class would be broken up into 3 subsets … let's call them malignant_1, malignant_2, malignant_3. Similarly, we would have benign_1, benign_2, and benign_3.

For the 1st fold, our "Validate" set would contain the data from malignant_1 and benign_1; while the "Train" set would contain the data from the rest of the sets.

For the 2nd fold, our "Validate" set would contain the data from malignant_2 and benign_2; while the "Train" set would contain the data from the rest of the sets.

For the 3rd (and final) fold, our "Validate" set would contain the data from malignant_3 and benign_3; while the "Train" set would contain the data from the rest of the sets.