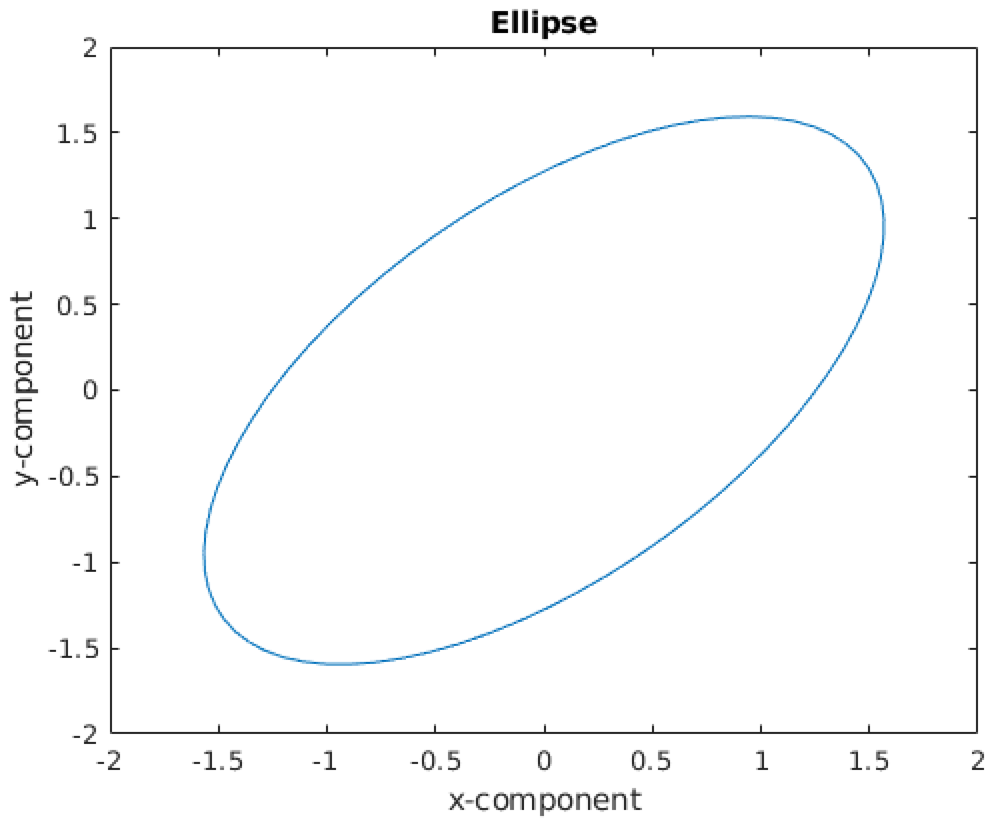
Table of Contents

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% Author = Ryan Senne
% Date = 4/9/2021
% Project 9
% The purpose of this project is to explore the theoretical nature of
 PCA
% and its relationships to ellipses. In this project I write a
 function to
% plot an ellipse, another function which plots a colored ellipse,
 plot a
% scatterplot of two random variables, plot its scaled eigenvectors,
 and
% write a function to plot confidence ellispes. As a bonus I will
 perform a
% principle component analysis on the Iris Dataset (please install
 matlab
% deep learning toolkit).
%
% As a quick refresher:
% covariance is the joint variability between two variables
% An eigenvector is the equation which satisfies Ax=lambdaX , where A
 is
% an m X n matrix, x is a vector and lambda is a scalar quantity. Thus
 an
% eigenvector is a column vector that when multiplied by A is
 transformed
% by a scalar quantity lambda, lambda is then by definition an
 eigenvalue.
% Eigenvectors in the context of a covariance matrix represent the
% direction of variability and an eigenvalue is its magnitude.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
```

question 1

```
% prepare workspace
clear all
```

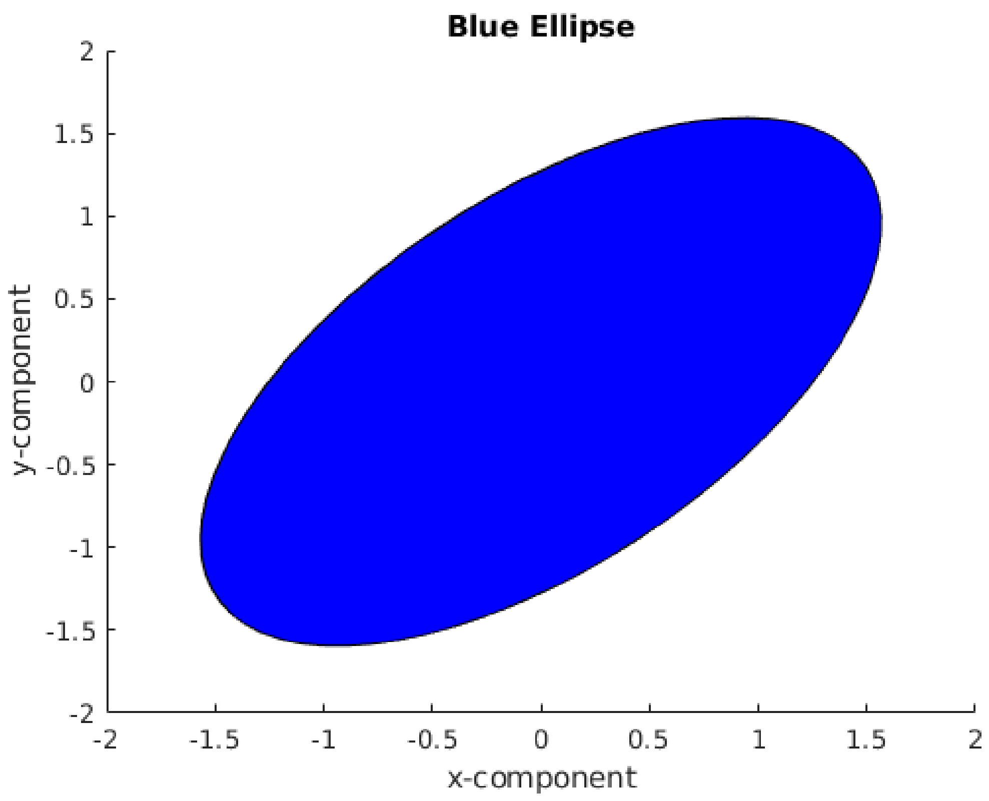
```
close all
% this section of code will plot an ellipse that has a major axis of 2
 and
% minor axis of 1, centered at the origin: (0,0). The ellipse has been
% rotated 0.8 radians
a = 2; % set major axis
b = 1; % set minor axis
x = 0; % set x-coordinate
y = 0; % set y-coordinate
theta = 0.8; % set rotation angle
ellipse(a,b,x,y,theta);
title('Ellipse');
xlabel('x-component')
ylabel('y-component')
```



question 2

```
% close plots
close all
% this section of code plots the same exact ellispe from the prior
 cell,
% BUT, it fills in the ellipse via the patch() fucntion. The ellipse
 will
% be blue.
color = 'b'; % set color of ellipse
colorEllipse(a,b,x,y,theta,color);
```

```
title('Blue Ellipse');
xlabel('x-component')
ylabel('y-component')
```



question 3

```
%close plots
close all
% this section of code will draw 75 ellipses with random major and
 minor
% axis values, x and y coordinates, rotation angles, and colors. This
 code
% essentially creates 75 random parameters for each input and plots
 them

% set random parameters
rng('default') % use default seed
% create normally distributed major axis values on the interval (-5,5)
randA = -5 + (5+5) * randn(1,75);
% create normally distributed minor axis values on the interval (-5,5)
randB = -5 + (5+5) * randn(1,75);
% create normally distributed x-coordinates values on the interval
 (-15,15)
randX = -15 + (15+15) * randn(1,75);
% create normally distributed y-coordinate values on the interval
 (-15,15)
randY = -15 + (15+15) * randn(1,75);
```

```
% create normally distributed rotation angle values on the interval
% (-2*pi, 2*pi)
randTheta = (-2*pi) + ((2*pi)+(2*pi)) * randn(1,75);

% we need random 'RGB' values so we can plot all kinds of colors
randomColorR = rand(1,75); % create random R values
randomColorG = rand(1,75); % create random G values
randomColorB = rand(1,75); % create random B values
% I realized after I wrote this that I could have just used linspace
 or
% something but, I already wrote the code--so here it stays

% this for loop will plot each of our 75 randomly designed ellipses
% it iterates through each randomly created values and uses the
% colorEllipse() function for plotting
for j=1:75
    a = randA(j); % random major axis values
    b = randB(j); % random minor axis values
    x = randX(j); % random x-coordinates
    y = randY(j); % random y-coordinates
    theta = randTheta(j); % random rotation angles
    color = [randomColorR(j) randomColorG(j)...
        randomColorB(j)]; % random RGB matrix
    colorEllipse(a,b,x,y,theta,color); % graphic design is my passion
end

% remove axes and title because its art
set(findobj(gcf, 'type','axes'), 'Visible','off')
```

question 4

```
%close plots
close all
% set random parameters
rng('default'); % use default seed
X = 3 + 2*randn(500,1); % x-values
Y = 2*X + 4*randn(500,1); % y-values
scatter(X,Y); % scatterplot of two variables
% make axes equal. DO NOT TURN THIS OFF; you need the axes to be equal
 so
% that the eigenvectors we will plot later look visually orthogonal
axis equal
hold on

% calculate centroid values (mean of each variable), from here our
% eigenvectors will originate from
x_0 = mean(X);
y_0 = mean(Y);

% calculate covariance matrix, eigenvectors/values
C = cov(X,Y); % calculate the covariance between our two variables
[eigVec, eigVal] = eig(C); % calculate eigs of covariance matrix

% new variables for eigenvectors for easy referencing
eig1 = eigVec(:,1);
```

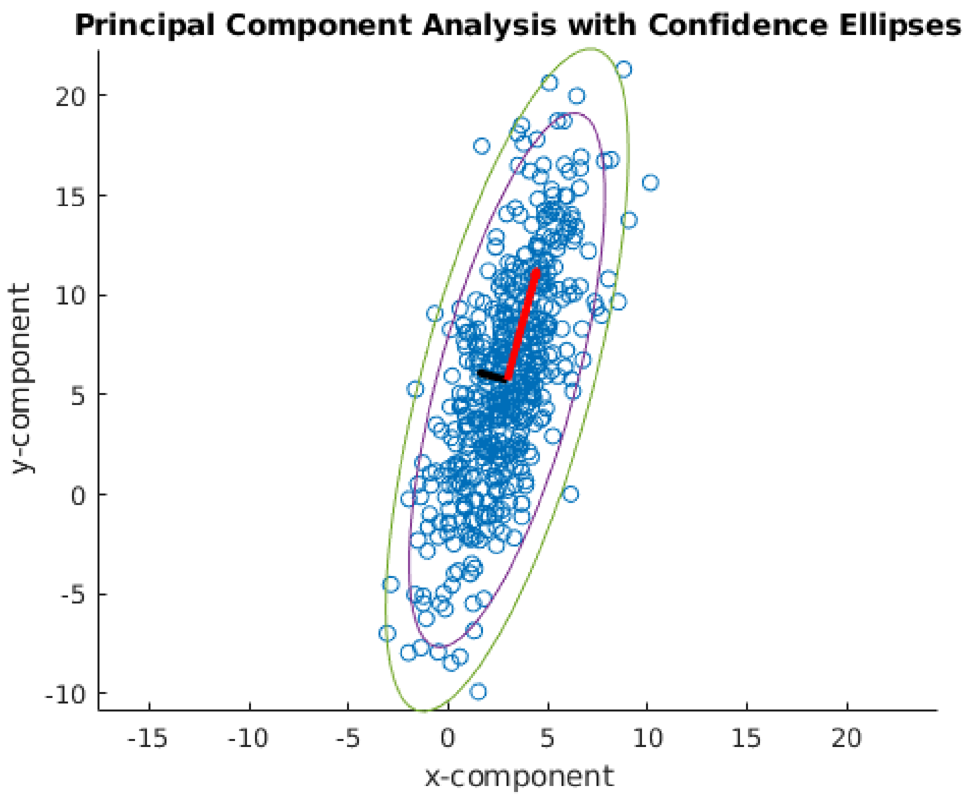
```
eig2 = eigVec(:,2);

% remove zeroes that matlab exports and scale eigenvalues
eigVal = sqrt(diag(eigVal));

% plot our eigenvectors
quiver(x_0,y_0,eig1(1),eig1(2),eigVal(1),'k','LineWidth',3);
quiver(x_0,y_0,eig2(1),eig2(2),eigVal(2),'r','LineWidth',3);

% plot 95% confidence ellipse
confidenceEllipse(X,Y,0.05);
% plot 99% confidence ellipse
confidenceEllipse(X,Y,0.01);

title('Principal Component Analysis with Confidence Ellipses')
xlabel('x-component')
ylabel('y-component')
hold off
```



bonus

```
close all
% in all honesty I just wanted to do more with PCA, I learned a lot
 from a
% theory perspective but I want to show a practical example of the
% technique. So, lets load one of the most famous datasets of all
 time, the
```

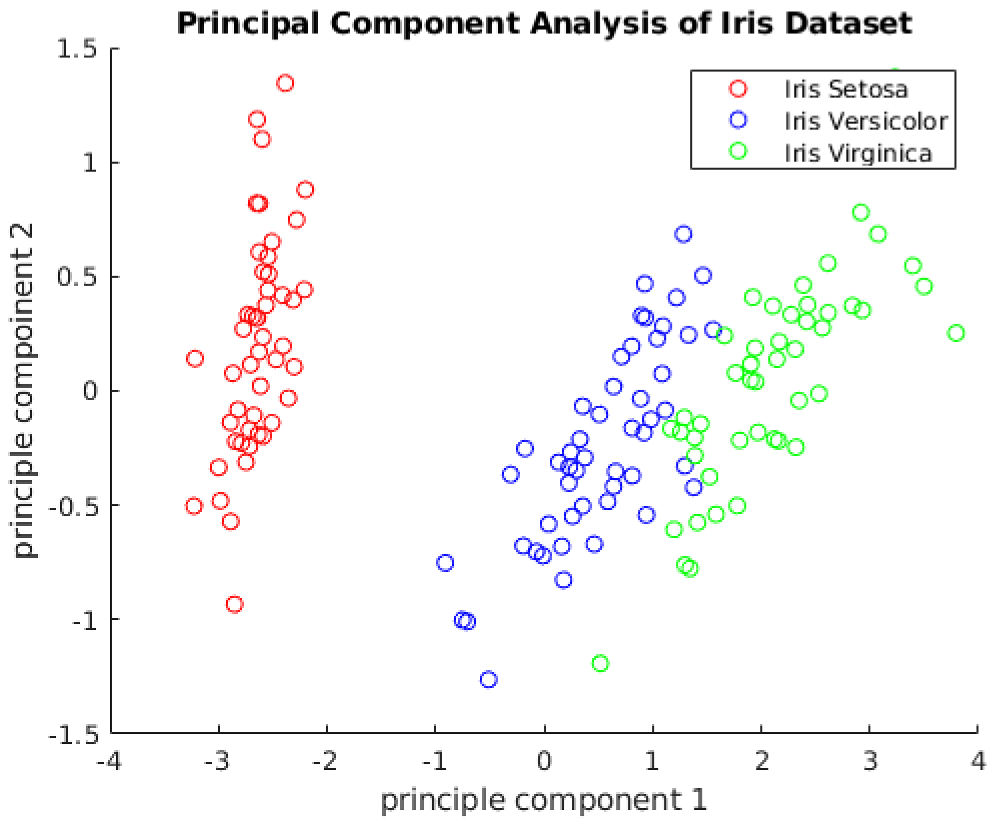
```
% iris dataset! ***YOU NEED THE DEEP LEARNING TOOLKIT FROM MATLAB FOR
% THIS***

load iris_dataset
% transpose matrix so the rows are the iris observations and the
 columns
% are the four variables
irisInputs = irisInputs';
% normalize by subtracting the mean from each variable, centers data
 on
% origin
irisInputsNorm = bsxfun(@minus,irisInputs,mean(irisInputs));
% calculate covariance matrix and eigenvalues/vectors
covarianceIris = cov(irisInputs);
[eigVec, eigVal] = eig(covarianceIris);

% sort eigenvalues by descending order of magniotude and create and
 index
% we can reference later
[eigenVal,ind] = sort(diag(eigVal),'descend');
% sort eigenvectors based on index we just created
sortedEigenVecs = eigVec(:,ind);
% grab the first two principal components
sortedEigs = sortedEigenVecs(:,[1:2]);
%multiply our eigenvectors by our original datset to transform the
 data
principleComponents = sortedEigs' * irisInputsNorm';

% the matlab datset doesn't tell you this but the first 50
 oberservations
% are of the Iris Setosa, the next 50 of Versicolor, and last 50 of
% virginica
irisSetosa = principleComponents(:,[1:50]);
irisVersicolor = principleComponents(:,[51:100]);
irisVirginica = principleComponents(:,[101:150]);

% plot all of our species on our new principal component space
hold on
scatter(irisSetosa(1,:), irisSetosa(2,:), 'r');
scatter(irisVersicolor(1,:), irisVersicolor(2,:), 'b');
scatter(irisVirginica(1,:), irisVirginica(2,:), 'g');
title('Principal Component Analysis of Iris Dataset')
xlabel('principle component 1')
ylabel('principle compoinent 2')
legend('Iris Setosa', 'Iris Versicolor', 'Iris Virginica')
```



Published with MATLAB® R2020b

```
function ellipseCoords = ellipse(a, b, x, y, theta, steps)
    % input:
    % this function returns coordinates for plotting an ellipse
    % parameter a is the major axis
    % parameter b is the minor axis
    % parameter x is the x-coordinate for the center
    % parameter y is the y-coordiante for the center
    % parameter theta is the angle in RADIANS, this function does NOT
    % take degrees as an input!!!
    % amount of steps used in drawing the ellipse, optional argument
    % where if no  input is supplied, 72 is default parameter
    %
    % output:
    % a {steps}x2 matrix consisting of the coordinates for plotting an
 ellipse

    % sets mininum and maximum amount of arguments, if steps not
 specified,
    % defaults to 72 steps
    narginchk(5,6);
    if nargin<6
        steps = 72;
    end

    % calculate sin and cos of our angle in radians
    sinbeta = sin(theta);
    cosbeta = cos(theta);

    % calculate the "eccentric anomaly" of our ellipse
    % create equally spaced points between 0 an 2pi using our steps
    alpha = linspace(0, 2*pi, steps)';
    sinalpha = sin(alpha);
    cosalpha = cos(alpha);

    % calculate rotated points of our ellipse using paramtric form
    X = x + (a * cosalpha * cosbeta - b * sinalpha * sinbeta);
    Y = y + (a * cosalpha * sinbeta + b * sinalpha * cosbeta);

    % matrix containing representative points of our ellipse
    ellipseCoords = [X, Y];

    %plot our ellipse
    plot(ellipseCoords(:,1),ellipseCoords(:,2))
end
```

Published with MATLAB® R2020b

```
function coloredEllipse = colorEllipse(a, b, x, y, theta, color,
 steps)
    % input:
    % this function returns coordinates for plotting an ellipse
    % parameter a is the major axis
    % parameter b is the minor axis
    % parameter x is the x-coordinate for the center
    % parameter y is the y-coordiante for the center
    % parameter theta is the angle in RADIANS, this function does NOT
    % take degrees as an input!!!
    % amount of steps used in drawing the ellipse, optional argument
    % where if no  input is supplied, 72 is default parameter
    %
    % output:
    % a stepsx2 matrix consisting of the coordinates for plotting an
 ellipse
    % a colored ellipse

    % sets mininum and maximum amount of arguments, if steps not
 specified,
    % defaults to 72 steps
    narginchk(6,7);
    if nargin<7
        steps = 72;
    end

    % calculate sin and cos of our angle in radians
    sinbeta = sin(theta);
    cosbeta = cos(theta);

    % calculate the "eccentric anomaly" of our ellipse
    % create equally spaced points between 0 an 2pi using our steps
    alpha = linspace(0, 2*pi, steps)';
    sinalpha = sin(alpha);
    cosalpha = cos(alpha);

    % calculate rotated points of our ellipse using paramtric form
    X = x + (a * cosalpha * cosbeta - b * sinalpha * sinbeta);
    Y = y + (a * cosalpha * sinbeta + b * sinalpha * cosbeta);

    % matrix containing representative points of our ellipse
    coloredEllipse = [X, Y];

    % plot our ellipse
    patch(coloredEllipse(:,1), coloredEllipse(:,2), color)
end
```

Published with MATLAB® R2020b

```
function CE = confidenceEllipse(x,y,alpha)
    % input
    % this function creates a confidence ellipse for a 2-D dataset
    % x: x-coordinates
    % y: y-coordinates
    % alpha: our alpha value i.e. if we want to be 95% confident, we
 select
    % an alpha of 0.05 as 1-0.95=0.05
    % output:
    % this will plot an ellipse using the ellipse function. Will also
 give
    % a matrix containing values neccesary to plot the ellipse

    % calculate the centroid of our data by finding the mean values of
 our
    % X and Y vairables
    meanX = mean(x);
    meanY = mean(y);

    % find a chi-squared value for our polar-ellispe equation to be
    % equivalent to
    % we have two standard normal variables, so we will always have
 k=2 dof
    k = 2; %dof
    p = 1 - alpha; %
    % we can use the noncentral probabiltiy density function to
 calculate
    % our chi-squared value, use 0 to assume our distribution is
 centered
    % at 0
    chi = ncx2inv(p, k, 0);

    % we need to calculate the covariance matrix to find its
 eigenvalues,
    % these will be used later
    covariance = cov(x,y);
    % calculate the eigenvectors and eigenvalues
    [eigVec, eigVal] = eig(covariance);
    % create variables for eigenvectors for easy referencing
    eig1 = eigVec(:,1);
    eig2 = eigVec(:,2);
    % this removes those pesky zeroes in the output matrix
    eigVal = diag(eigVal);

    % calculate the axis values. the axis of an error ellipse is equal
 to
    % the square root of the chivalue of an error ellipse multiplied
 by the
    % eigenvalue. Remember, the eigenvector shows the direction of
 variance
    % and the eigenvalue shows its magnitude!
    % calculate the major axis
    a = sqrt(chi*eigVal(1));
    % calcualte the minor axis
    b = sqrt(chi*eigVal(2));

    % we can calculate the rotation angle by using the eigenvector
    % associated with the largest eigenvalue i.e. the eigenvector
    % corresponding to the axis with most variance. We can use
    % arctan(Eig-ycomponent/Eig-xcomponent)
    if  eigVal(2)>eigVal(1)
        theta = atan(eig1(2)/eig1(1));
    else
        theta = atan(eig2(2)/eig2(1));
    end

    %plot the confidence ellipse
    ellipse(a,b,meanX,meanY,theta, 72);

    % gives output containing all info necessary, in case you need to
 plot
    % it again later
    CE = [ a,b,meanX,meanY,theta];
end
```

Published with MATLAB® R2020b