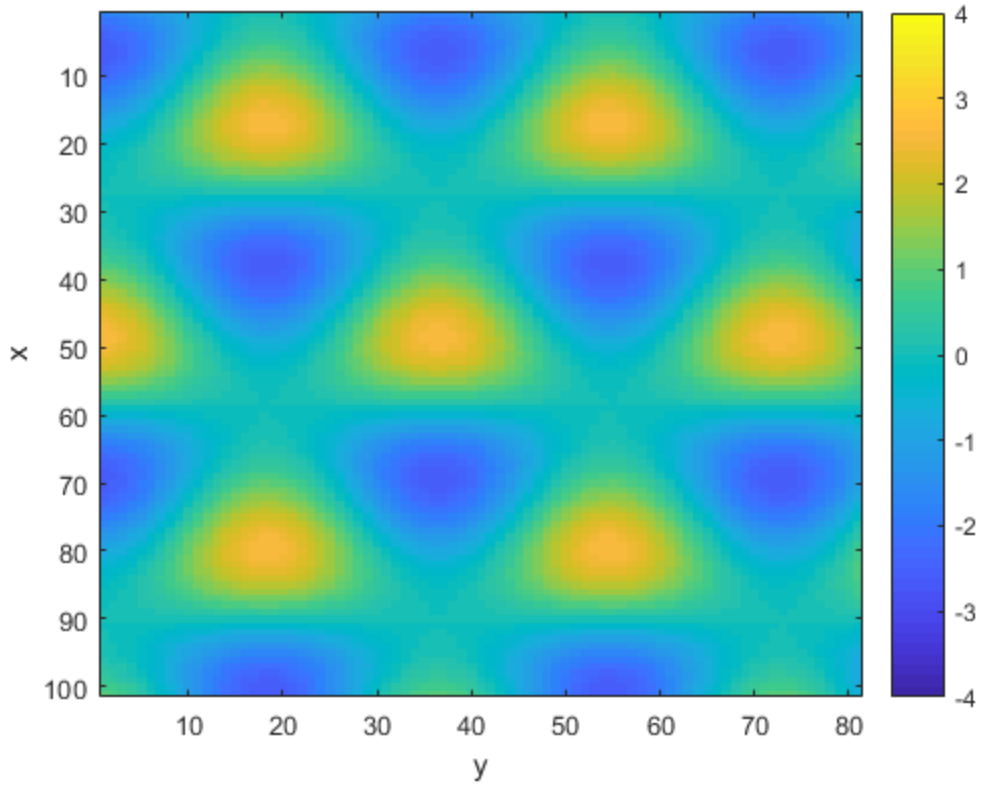
Table of Contents

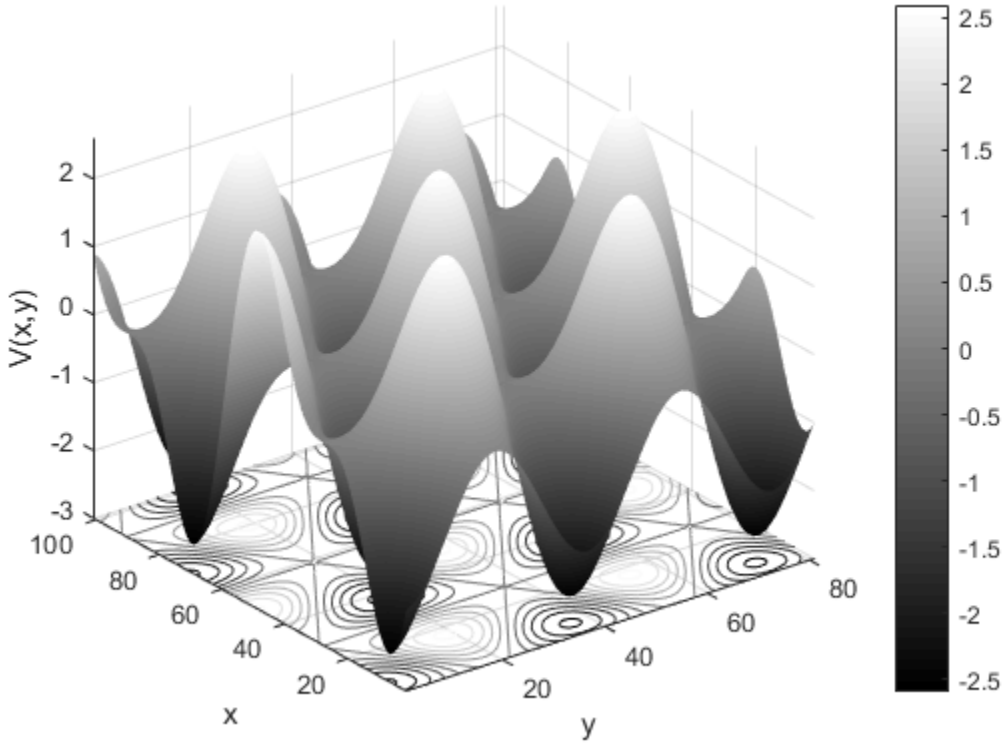

```
% INSTRUCTIONS:
% 1) save this file into the directory that holds your work
% 2) make sure it runs without error
% 3) publish this file to PDF.
% 4) There should then be a PDF file in the subdirectory 'html'
%    that you can upload to TurnItIn.
%    PLEASE DOUBLE-CHECK it has everything you need!!

% prepare to run...
clear all  % clear out all variables
close all  % close any open figures
```

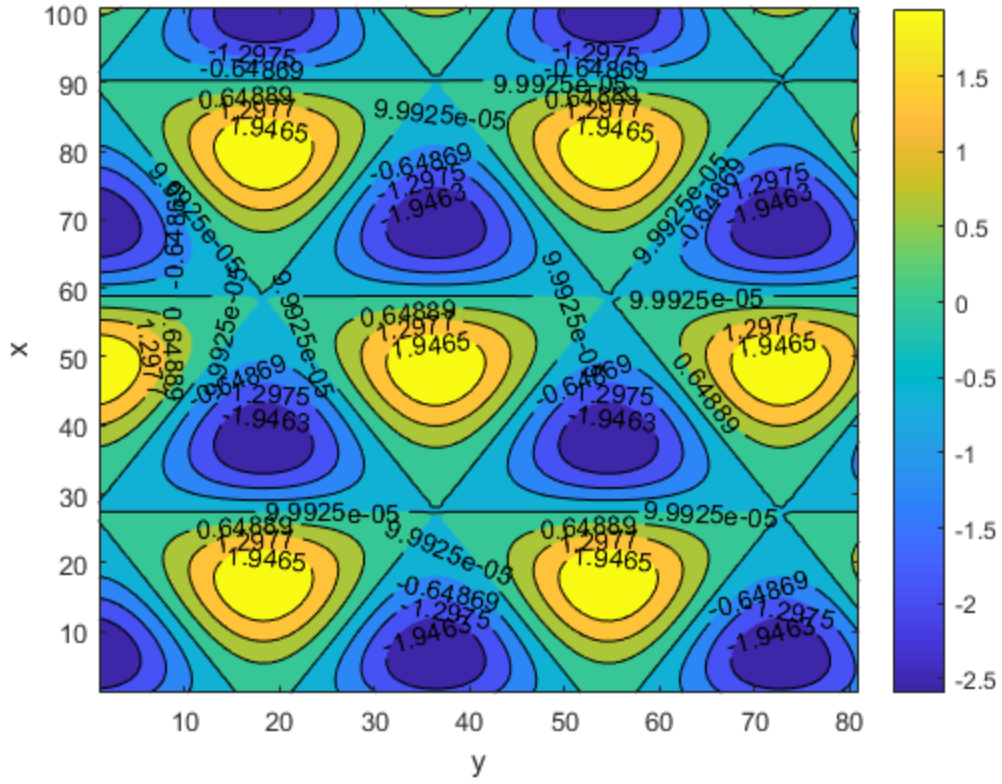
problem 1

```
% add in some horizontal lines to break up code
type('PrettyPicture')

fprintf('\n----------------------------------\n')
type('PlotPrettyPicture')

fprintf('\n----------------------------------\n')
PlotPrettyPicture


function vMatrix = PrettyPicture(a,b,c,k)
%{
takes in a,b,c, and k(numbers that change the graph values)
and creates a matrix of equation outputs using x,y,a,b,c,k to
create a function.

Alyssa Rose  HW6  3-18-2018
%}
%creates x and y values
x = linspace(-10,10,101);
y = linspace(-8,8,81);
vMatrix = zeros(101,81);
%creates a matrix with output values
for n=1:101
    for m=1:81
        vMatrix(n,m) = cos(a + k*(sqrt(3)*y(m) + x(n))/2)+ cos(b +
 k*(sqrt(3)*y(m) - x(n))/2)+ cos(c + k*x(n));
```

```
    end
end

---------------------------------

%{
uses PrettyPicture function to create 3 graphs

Alyssa Rose  HW6   3-14-18
%}
z = PrettyPicture(pi/2, 0, 0, 1);
x = linspace(-10,10,101);
y = linspace(-8,8,81);

figure(1)
imagesc(z);
colorbar
caxis([-4,4]); %sets colorbar axis
xlabel('y')
ylabel('x')

figure(2)
surfc(z)
colormap(gray)
colorbar
xlabel('y')
ylabel('x')
zlabel('V(x,y)')
shading INTERP %removes crosshatching

figure(3)
[C,h] = contourf(z,7); %creates 7 levels
clabel(C,h) %labels the 7 levels
xlabel('y')
ylabel('x')
colorbar

---------------------------------
```


x
y
10
20
30
40
50
60
70
80
90
100
4
3
2
1
0
-1
-2
-3
-4

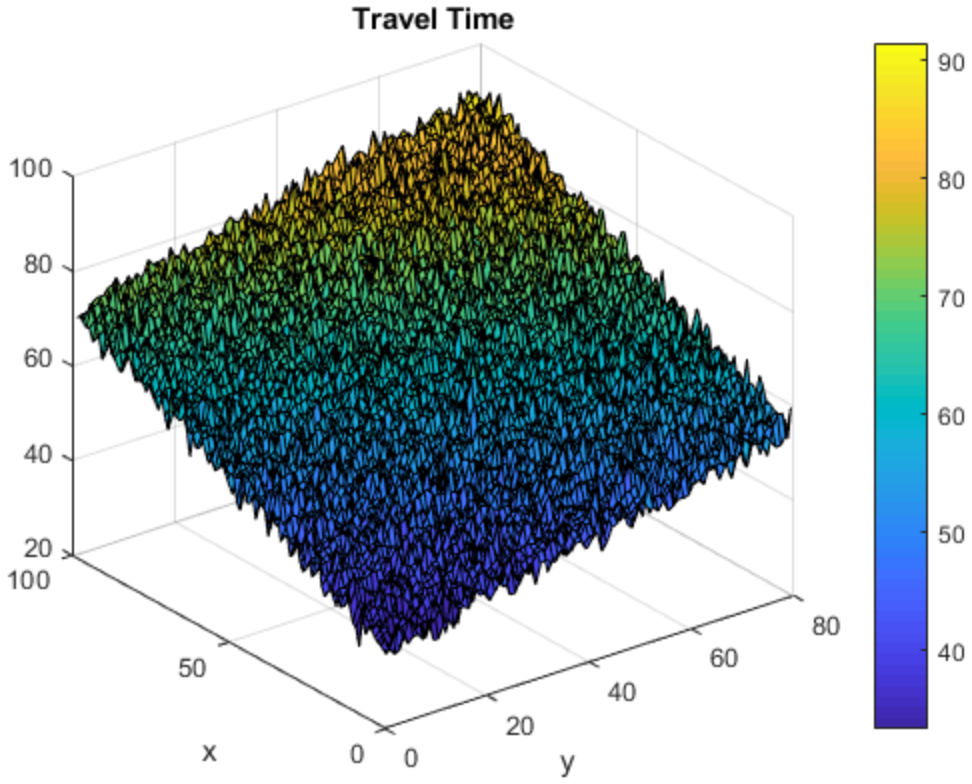

V(x,y)
x
y
2
1
0
-1
-2
-3
100
80
60
40
20
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5



problem 2

```
type('SimScanner')

fprintf('\n----------------------------------\n')
type('PlotScanData')

fprintf('\n----------------------------------\n')
PlotScanData


function scanData = SimScanner(nxr,nxc,dx,sigma)
%{
Takes in nxr/nxc (which are # of elements in rows and columns),
dx (spacing between points) and sigma(multiplier for the noise term)
and returns the travel time data for an ultrasonic range finder

Alyssa Rose HW 6   3-13-2018
%}

%spedd of sound in air
c= 340;
%preallocation
scanData = zeros(nxr,nxc);
dist= zeros(nxr,nxc);
%creates values for rows and columns
```

```
xR = linspace(dx, nxr*dx, nxr);
xC = linspace(dx, nxc*dx, nxc);
%loops over each value to calc distance and travel time
for m = 1:nxr
    for k=1:nxc
        n = sigma*randn(1);
        d(m,k) = 6 + 1.2*xR(m) + 0.8*xC(k);
        scanData(m,k) = 2000*d(m,k)/c + n;
    end
end

-----------------------------------

%{
plots SimScanner data

Alyssa Rose HW6  3-14-2018
%}
z = SimScanner(100,80,0.05,2);
surf(z);
ylabel('x')
xlabel('y')
colorbar
title('Travel Time')

-----------------------------------
```



problem 3

```
type('HomemadeMatrixMult')
```

```
function matrixC = HomemadeMatrixMult(matrixA, matrixB)
%{
takes in 2 matrices and outputes their product

Alyssa Rose  HW6  3-14-2018
%}
%finds size of matrices
[rowA, colA] = size(matrixA);
[rowB, colB] = size(matrixB);
%checks if matrices can be multiplied
if colA~=rowB
    matrixC = NaN;
    return
end
matrixC= [];
%finds the products of the matrices
for m = 1:colB
    matrixC=[matrixC matrixA* matrixB(:,m)];
end
```

problem 4

```
type('NormalizedDotProduct')
```

```
function dp = NormalizedDotProduct(v1,v2)
%{
takes in 2 vectors and finds their dot products

Alyssa Rose  HW6  3-14-2018
%}
%checks if vectors are same length
if length(v1)~=length(v2)
    dp = NaN;
    return
end
%find size of the vector
[rv1,cv1] = size(v1);
[rv2,cv2] = size(v2);
%converts vector to column vector
if cv1~=1
    v1 = v1';
end

if cv2~=1
    v2 = v2';
end
```

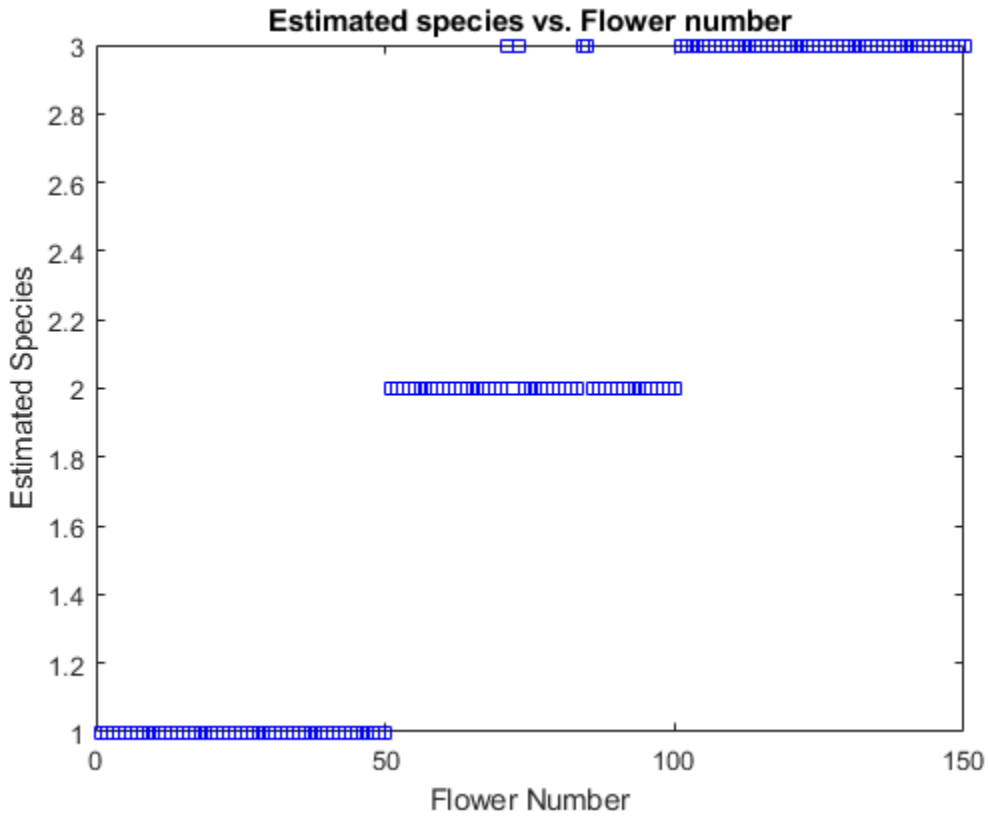
```
%finds unit vector and computes dot product
unitV1 = v1./(sqrt(sum(v1.^2)));
unitV2 = v2./(sqrt(sum(v2.^2)));
dp = sum(unitV1.*unitV2);
```

problem 5

```
type('IrisClassifier')

fprintf('\n----------------------------------\n')
IrisClassifier(true);  % makes the plots


function [accuracy, speciesAvgDP] = IrisClassifier(isPlotted)
%{
take in boolean (true, false) and outputs dot products
of all species and how close estimation is to actual.

Estimates the species of an iris flower based on
how similar the flower and average species are (by
determining how close the dot products are to 1)

Alyssa Rose HW 6    3-13-18
%}
load irisData.mat
%finds how similar species are to each other
speciesAvgDP(1) =
 NormalizedDotProduct(averageSetosa,averageVersicolor);
speciesAvgDP(2) =
 NormalizedDotProduct(averageSetosa,averageVirginica);
speciesAvgDP(3) =
 NormalizedDotProduct(averageVersicolor,averageVirginica);
speciesAvgDP
%{
I think that averageVersicolor and averageVirginica are
most difficult to distinguish since their dot product is
extremely close to 1, meaning that they are nearly identical.
%}
estVec = [];
dotVec= [];
%finds estimate for species of flowers
for m=1:150
    dotVec(1) = NormalizedDotProduct(irisFeatures(:,m),
 averageSetosa);
    dotVec(2)= NormalizedDotProduct(irisFeatures(:,m),
 averageVersicolor);
    dotVec(3) = NormalizedDotProduct(irisFeatures(:,m),
 averageVirginica);
    [M,I] = max(dotVec);
    %creates vector of estimation species
    estVec(m) = [I];
end
%finds accuracy of vector estimation and true species
```

```
accuracy = 100*(numel(find(estVec==trueSpecies))/numel(estVec));
fprintf('Percent accuracy: %2.1f', accuracy)
%plots graph if input is 'true'
if isPlotted == true
    figure (4)
    plot(estVec,'sb');
    xlabel('Flower Number')
    ylabel('Estimated Species')
    title('Estimated species vs. Flower number')
end
return
```

```
----------------------------------

speciesAvgDP =

    0.9248    0.8884    0.9957

Percent accuracy: 97.3
```



problem 6

```
type('ImageCombo')

fprintf('\n----------------------------------\n')
type('FaceMorph')
```

```
function imgOut = ImageCombo(img1,img2,alphaPercent)
%{
takes in two images and a number to calc the weighted
average of the images

Alyssa Rose HW 6   3-13-2018
%}
if alphaPercent<0
    alphaPercent = 0;
elseif alphaPercent >100
    alphaPercent =100;
end
%reads in 2 images and converts to double
img1 = double(img1);
img2 = double(img2);
%calcs weighted average and converts back to uint8
imgOut = uint8((((alphaPercent)/100)*img1) + (((100-
alphaPercent)/100)*img2));
return
end
```

```
---------------------------------
```

```
%{
reads in two images and fades them in and out based on weighted
 average

Alyssa Rose HW 6   3-13-2018
%}
k = imread('thing1.jpg');
m = imread('thing2.jpg');

for alpha = 0:100
    g = uint8(ImageCombo(k, m, alpha));
    imagesc(g);
    pause(0.01);
end
```

Published with MATLAB® R2017b