Introduction

This assignment requires you to write a MatLab program that performs an animation of 4 surface functions by warping the first surface into the second surface, then the second surface into the third surface and so one, until the fourth surface is warped into the first surface. The surface functions are:

\[
\begin{align*}
  f_1(x, y) & = 2x^2 + 2y^2 \\
  f_2(x, y) & = -2x^2 + 2y^2 \\
  f_3(x, y) & = -2x^2 - 2y^2 \quad \text{and} \\
  f_4(x, y) & = 2x^2 - 2y^2.
\end{align*}
\]

Figures 1a to 1b shows the mesh plots of these four surfaces. Surfaces 1 and 3 in Figures 1a and 1c are quadric surfaces (cup like) while surfaces 2 and 4 in Figures 1b and 1d are surfaces with saddle points.

Plotting the Functions

The various task in this assignment include:

1. First, you have to plot the 4 functions. Use \( x_{\text{min}} \) and \( y_{\text{min}} \) values of -2 and \( x_{\text{max}} \) and \( y_{\text{max}} \) values of +2, along with \( \Delta x = \Delta y = 0.05 \), to generate \( X \) and \( Y \) from \texttt{meshgrid}. The \( z \) values are computed by MatLab (but fix them at \( z_{\text{min}} = -4 \) and \( z_{\text{max}} = +4 \) using \texttt{axis} for all the figures you display in this assignment). Then the 3D depth values, \( Z_1 \) to \( Z_4 \), can be generated by a vectorized calculation for these 4 functions. For the 1st surface you can use:

\[
Z_1 = -sqrt(2*X.^2 + 2*Y.^2);
\]
(a) \( f_1(x, y) = 2 * x^2 + 2 * y^2 \)

(b) \( f_2(x, y) = -2 * x^2 + 2 * y^2 \)

(a) \( f_3(x, y) = -2 * x^2 + 2 * y^2 \)

(b) \( f_4(x, y) = 2 * x^2 - 2 * y^2 \)

Figure 1: Surface plots of the four functions.

Note that surfaces Z1, Z2, Z3 and Z4 correspond to functions \( f_1(x, y) \), \( f_2(x, y) \), \( f_3(x, y) \) and \( f_4(x, y) \) respectively. You should perform numerical and symbolic integration on these 4 functions and use \texttt{text} to print out these values in cyan and blue respectively. Choose an appropriate fontsize and appropriate 3D coordinates for positioning the text. See below for details on performing numerical and symbolic integration.

2. You must numerically integrate the 4 functions. You can integrate an anonymous function defined for the 1\(^{st}\) surface as:

\[
\text{fun1} = @(X,Y)(2*X.^2+2*Y.^2);
\]
num_area1=integral2(fun1,xmin,xmax,ymin,ymax);

Here fun1 is the handle to an anonymous function (a function that has no name). You can pass this handle to another function to use. In this assignment, you can integrate it using integral2. This function evaluates the area under this function using numerical quadrature.

3. You must symbolically integrate the 4 functions. First, declare f1, x and y to be symbol variables,

syms f1 x y

and then use:

f1=2*x^2+2*y^2;
sym_area1=eval(int(int(f1,y,ymin,ymax),x,xmin,xmax));

to integrate the 1st surface. Note the use of eval to evaluate the symbolic integration result. For this assignment numerical and symbolic integration always yield the same answers.

4. You can use the title command to print out the mathematical formula for each of the 4 plots. [An expression is interpreted as latex in MatLab, with _ indicating a subscript and a ^ indicating a superscript. Thus x_2 is x^2 while x^-2 is x^2.] Also use xlabel, ylabel and zlabel to label the x, y and z axes in your figures. Use text to print out the numerical and symbolic integration results on these graphs (see below). You have to chose the x, y and z values to position these integration values on the plots.

5. You create the animation by warping Z1 into Z2, then Z2 into Z3, Z3 into Z4 and finally Z4 back into Z1. So the initial and final surfaces are the same. To code the warping for Z1 to Z2 you can use something like:

for t=0:delta_t:1
    Z=Z1*(1-t)+Z2*(t);
    mesh(X,Y,Z)
    axis([xmin xmax ymin ymax zmin zmax]);
text(xpos,ypos,zpos1,['\fontsize{20} \color{magenta} ' ...
    'Numerical Integration: ' ...  
    sprintf('%8.3f',num_area1*(1-t)+num_area2*t));

% {1.0 0.6 0.0} is orange

text(xpos,ypos,zpos2,['\fontsize{20} \color[rgb]{1.0 0.6 0.0} ' ... 
    'Symbolic Integration: ' ...  
    sprintf('%8.3f',sym_area1*(1-t)+sym_area2*t));

xlabel('X azis');
ylabel('Y azis');
zlabel('Z azis');
pause(0.01);
drawnow;
end % for t

The statement \(Z=Z1(1-t)+Z2(t)\) does the surface warping, i.e. this is the linear interpolation of the two surfaces \(Z1\) and \(Z2\). For \(t = 0\), \(Z=Z1\) while for \(t = 1\), \(Z=Z2\). Intermediate values of \(t\) give you the various combined surfaces of \(Z1\) and \(Z2\). \(\delta t\) is a small number, say 0.05. Thus when this runs you will see a total of 21 surfaces displayed rapidly as \(Z1\) warps into \(Z2\). If the display is too rapid, you can pause a small amount of time between displays to slow things down). Use text to print out the numerical and symbolic integration areas. Set the variables \(xpos\), \(ypos\) and \(zpos1\) and \(zpos2\) appropriately, where \(zpos1\) and \(zpos2\) are the vertical dimension for this figure. Trial and error is required here. You could use grid on and box on to get some good initial values. Note that the text is printed in magenta or orange and the numerical and symbolic integration areas are linearly interpolated to correspond with the current surface being displayed. After warping \(Z1\) into \(Z2\), you need to warp \(Z2\) into \(Z3\), \(Z3\) into \(Z4\) and finally \(Z4\) back into \(Z1\). At this point you will have your animation working.

6. Finally, the last task is to make the animation figure bigger than normal. The handle for the screen is 0. \texttt{get(0,'screensize')} gets the lower \(x\) and \(y\) coordinates of the screen plus its width and height. Set the height of the figure to be 75\% of the height of the screen. Given the height, compute the width that satisfies an aspect ratio of 3/4. Finally, compute the lower \(x\) and \(y\) coordinates of the figure to be small percentages of
the screen width and height. These will offset your figure from the lower left corner of
the screen. When you generate the animation figure (the 5th figure in your program)
you must save its handle and use this handle to set the figure’s position properties
using set. The following MatLab code outlines how all this might be done:

```
set(0, 'units', 'pixels');
screenSizePixels = get(0, 'screensize');
screenWidth = screenSizePixels(3);
screenHeight = screenSizePixels(4);
figureAspectRatio = 3/4;  % height to width

figureHeight = screenHeight * 0.75;
figureWidth = screenHeight * 1.0 / figureAspectRatio;

% shift left 5% of the screen width
leftx = screenWidth * 0.05;
% shift up 15% of the screen height
lefty = screenHeight * 0.15;

h5 = figure;
set(h5, 'Position', [leftx lefty figureWidth figureHeight]);
```

By setting all position and height/width figure information in terms of the computer’s
screen height and width, you will make the animation figure have the same relative
dimensions on all computers that run your program (regardless of their screen sizes).