**Task 1 - Low level image file reading and writing**

%%load an example image in mat file

load('example\_image.mat')

%%Write the image into a file “data/image.sim

xd = double(vol); % x is a double

gim =reshape(xd,224,2240,3);

imwrite(gim,'image.sim.jpg');

%Read the file “data/image.sim” using simple\_image\_read function. [2]

gim=imread('image.sim.jpg');

%Plot 3 images at different z-coordinates to verify the read images

imshow(gim,[]),title('titl');



**Task 2 - Plotting multivariate Gaussian probability density surfaces**

%%Plotting multivariate Gaussian probability density surfaces

%%Randomly generate 10,000 samples of ?from any distribution

mu = [0 0];

Sigma = [0.25 0.3; 0.3 1];

x1 = -3:0.2:3;

x2 = -3:0.2:3;

[X1,X2] = meshgrid(x1,x2);

X = [X1(:) X2(:)];

%%Fit a Gaussian to the generated random points, i.e. computing the mean and covariance

%%matrix from the available data

Compute and plot the pdf of a bivariate normal distribution with parameters mu = [0 0] and Sigma = [0.25 0.3; 0.3 1].

%%probability densities

y = mvnpdf(X,mu,Sigma);

y = reshape(y,length(x2),length(x1));

%%Plot the three ellipsoid surfaces

surf(x1,x2,y)

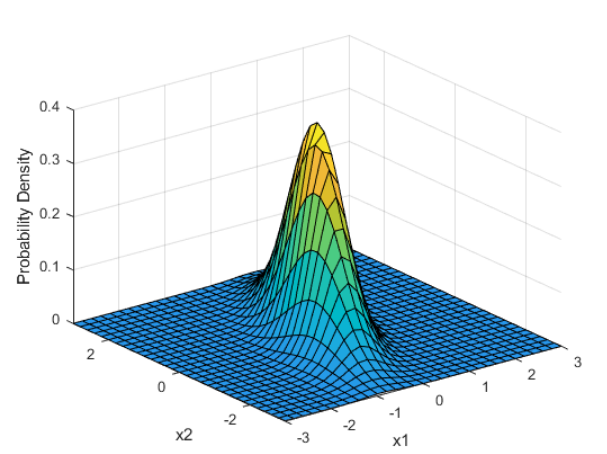
caxis([min(y(:))-0.5\*range(y(:)),max(y(:))])

axis([-3 3 -3 3 0 0.4])

xlabel('x1')

ylabel('x2')

zlabel('Probability Density')



mu = [1 -1];

Sigma = [.9 .4; .4 .3];

[X1,X2] = meshgrid(linspace(-1,3,25)',linspace(-3,1,25)');

X = [X1(:) X2(:)];

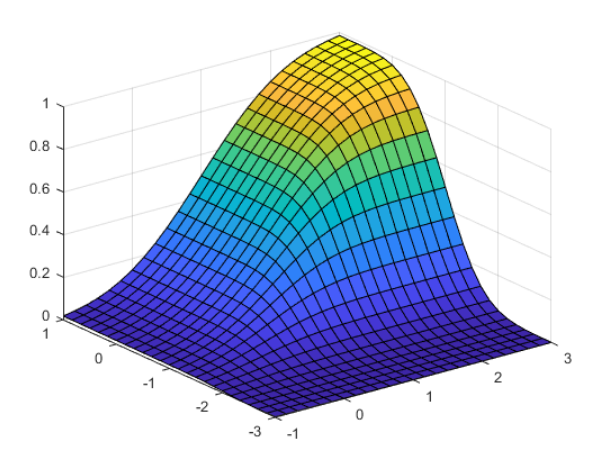
p = mvncdf(X,mu,Sigma);

Z = reshape(p,25,25);

surf(X1,X2,Z)

Compute and plot the cdf of a bivariate normal distribution.

Define the mean vector mu and the covariance matrix Sigma.



mu = [0 0];

Sigma = [0.25 0.3; 0.3 1];

p = mvncdf([0 0],[1 1],mu,Sigma)

x1 = -3:.2:3;

x2 = -3:.2:3;

[X1,X2] = meshgrid(x1,x2);

X = [X1(:) X2(:)];

y = mvnpdf(X,mu,Sigma);

y = reshape(y,length(x2),length(x1));

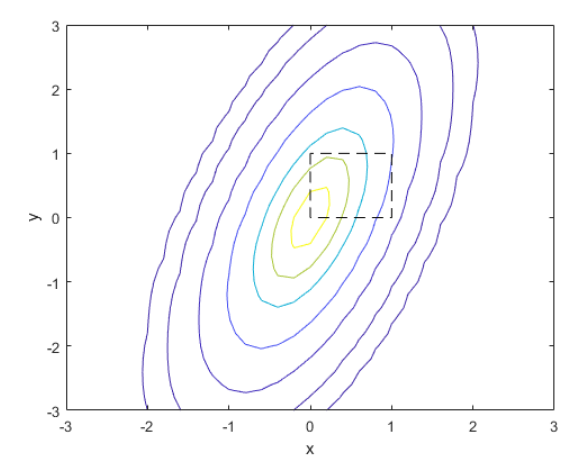
contour(x1,x2,y,[0.0001 0.001 0.01 0.05 0.15 0.25 0.35])

xlabel('x')

ylabel('y')

line([0 0 1 1 0],[1 0 0 1 1],'Linestyle','--','Color','k')

Finally, create a contour plot of the multivariate normal distribution that includes the unit square.



Computing a multivariate cumulative probability requires significantly more work than computing a univariate probability. By default, the mvncdf function computes values to less than full machine precision, and returns an estimate of the error as an optional second output.

**Task 3 - Gradient descent from scratch**

% Gradient descent from scratch

clc

clear

format long

% Function Definition (Enter your Function here):

syms X Y;

f = X - Y + 2\*X^2 + 2\*X\*Y + Y^2;

% Initial Guess:

x(1) = 1;

y(1) = -5;

e = 10^(-8); % Convergence Criteria

i = 1; % Iteration Counter

% Gradient Computation:

df\_dx = diff(f, X);

df\_dy = diff(f, Y);

J = [subs(df\_dx,[X,Y], [x(1),y(1)]) subs(df\_dy, [X,Y], [x(1),y(1)])]; % Gradient

S = -(J); % Search Direction

% Minimization Condition:

while norm(J) > e

I = [x(i),y(i)]';

syms h; % Step size

g = subs(f, [X,Y], [x(i)+S(1)\*h,y(i)+h\*S(2)]);

dg\_dh = diff(g,h);

h = solve(dg\_dh, h); % Optimal Step Length

x(i+1) = I(1)+h\*S(1); % Updated x value

y(i+1) = I(2)+h\*S(2); % Updated y value

i = i+1;

J = [subs(df\_dx,[X,Y], [x(i),y(i)]) subs(df\_dy, [X,Y], [x(i),y(i)])]; % Updated Gradient

S = -(J); % New Search Direction

end

% Result Table:

Iter = 1:i;

X\_coordinate = x';

Y\_coordinate = y';

Iterations = Iter';

T = table(Iterations,X\_coordinate,Y\_coordinate);

% Plots:

fcontour(f, 'Fill', 'On');

hold on;

plot(x,y,'\*-r');

% Output:

fprintf('Initial Objective Function Value: %d\n\n',subs(f,[X,Y], [x(1),y(1)]));

if (norm(J) < e)

fprintf('Minimum succesfully obtained...\n\n');

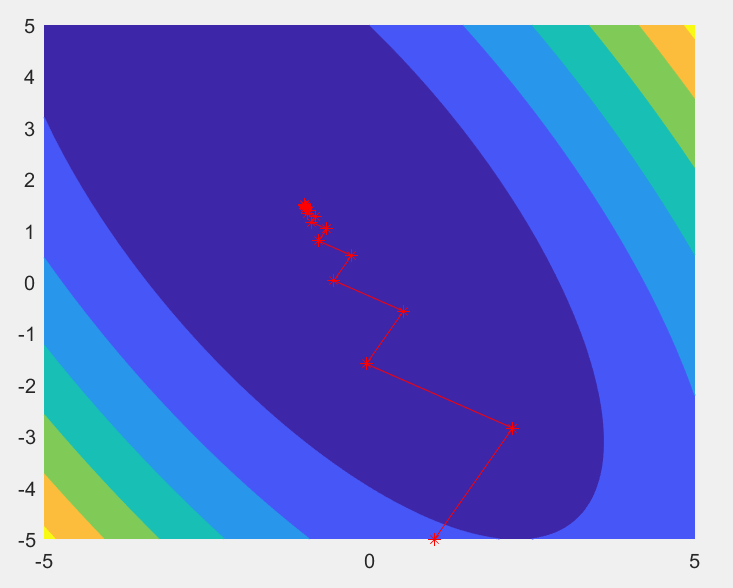
end

fprintf('Number of Iterations for Convergence: %d\n\n', i);

fprintf('Point of Minima: [%d,%d]\n\n', x(i), y(i));

fprintf('Objective Function Minimum Value Post-Optimization: %d\n\n', subs(f,[X,Y], [x(i),y(i)]));

disp(T);



Task 4 – Triangulation surface mesh smoothing

Implement a function lowpass\_mesh\_smoothing, which takes the input arguments: 1) A list of vertices  
– points in 3D; 2) A list of triangles that are the point indices forming the triangles; 3) Number of the  
iterations to apply the filtering, with a default value of 10; 4) A regularisation parameter ‘lambda’ with  
a default value of 0.9; and 5) A ratio of the reduction parameter ‘mu’ with a default value of -1.02\*  
lambda, as described in the paper. The function outputs a list of vertices with adjusted point  
coordinates, representing the smoothed surface. [15]

% Read an example triangulation mesh in the csv files, “data/example\_vertices.csv” and

% “data/example\_triangles.csv”. [2] (N.B. Examples of using such triangulation mesh can be

% found in the <iterative\_closes\_point> tutorial in the module repository.)

% o Smooth the surface mesh with three different numbers of iterations, 5, 10, 25. [3]

% o Plot the surface meshes before and after each of filtering operations, in a clear and visually

% comparable manner. [5]

filename = 'H:\service\2021\1 29 241269735\coursework202021\coursework202021\data\example\_vertices.csv';

delimiter = ',';

%% 每个文本行的格式:

% 列1: 双精度值 (%f)

% 列2: 双精度值 (%f)

% 列3: 双精度值 (%f)

% 有关详细信息，请参阅 TEXTSCAN 文档。

formatSpec = '%f%f%f%[^\n\r]';

%% 打开文本文件。

fileID = fopen(filename,'r');

%% 根据格式读取数据列。

% 该调用基于生成此代码所用的文件的结构。如果其他文件出现错误，请尝试通过导入工具重新生成代码。

dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'ReturnOnError', false);

%% 关闭文本文件。

fclose(fileID);

%% 对无法导入的数据进行的后处理。

% 在导入过程中未应用无法导入的数据的规则，因此不包括后处理代码。要生成适用于无法导入的数据的代码，请在文件中选择无法导入的元胞，然后重新生成脚本。

%% 创建输出变量

examplevertices = table(dataArray{1:end-1}, 'VariableNames', {'VarName1','VarName2','VarName3'});

%% 清除临时变量

clearvars filename delimiter formatSpec fileID dataArray ans;

filename = 'H:\service\2021\1 29 241269735\coursework202021\coursework202021\data\example\_triangles.csv';

delimiter = ',';

%% 每个文本行的格式:

% 列1: 双精度值 (%f)

% 列2: 双精度值 (%f)

% 列3: 双精度值 (%f)

% 有关详细信息，请参阅 TEXTSCAN 文档。

formatSpec = '%f%f%f%[^\n\r]';

%% 打开文本文件。

fileID = fopen(filename,'r');

%% 根据格式读取数据列。

% 该调用基于生成此代码所用的文件的结构。如果其他文件出现错误，请尝试通过导入工具重新生成代码。

dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string', 'ReturnOnError', false);

%% 关闭文本文件。

fclose(fileID);

%% 对无法导入的数据进行的后处理。

% 在导入过程中未应用无法导入的数据的规则，因此不包括后处理代码。要生成适用于无法导入的数据的代码，请在文件中选择无法导入的元胞，然后重新生成脚本。

%% 创建输出变量

exampletriangles = table(dataArray{1:end-1}, 'VariableNames', {'VarName1','VarName2','VarName3'});

%% 清除临时变量

clearvars filename delimiter formatSpec fileID dataArray ans;

[x,y,z]=peaks(14); % sample data

v=-3:1.2:6; % contour levels to draw

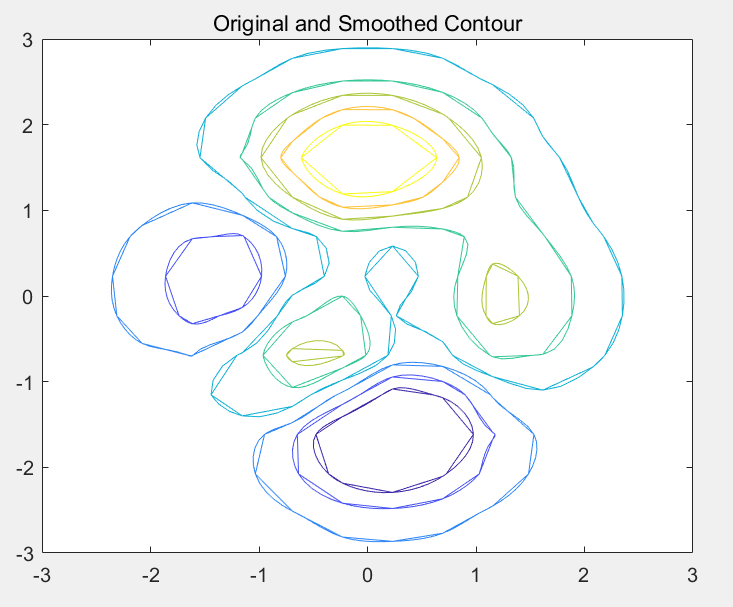
contour(x,y,z,v); % standard contour

hold on

lowpass\_mesh\_smoothing(x,y,z,v); % smoothed contours

title('Original and Smoothed Contour')

hold off



**Part 2 Spatial data augmentation using a Gaussian spline**

classdef RBFSpline

% end

methods (Static)

% function obj =

% %UNTITLED2 Construct an instance of this class

% % Detailed explanation goes here

% obj.Property1 = inputArg1 + inputArg2;

% end

function alpha = fit(source,target,lambda,sigma)

p\_sz = size(source); % source list of coords for each point

n = p\_sz(1);

% W = zeros(n);

%

alpha = zeros(3,n);

[K\_x,K\_y,K\_z] = RBFSpline.kernel\_gaussian(source(:,1)',source(:,2)',source(:,3)',source,sigma);

% for ii = 1:n

% W(ii,ii) = 1/(sigma(ii)^2);

% end

q1 = target(:,1);

q2 = target(:,2);

q3 = target(:,3);

%Ax = b, least squares problem: (K+lambda\*W^-1)Alpha = q\_k. x =

%pinv(A)\*b

A\_x = (K\_x + lambda\*eye(n));

A\_y = (K\_y + lambda\*eye(n));

A\_z = (K\_z + lambda\*eye(n));

%

alpha(1,:) = pinv(A\_x)\*q1;

%

alpha(2,:) =pinv(A\_y)\*q2;

%

alpha(3,:) = pinv(A\_z)\*q3;

end

function [u\_x,u\_y,u\_z] = evaluate(x,y,z,control,alpha,sigma)

[K\_x,K\_y,K\_z] = RBFSpline.kernel\_gaussian(x,y,z,control,sigma);

K\_x = alpha(1,:) .\* K\_x;

K\_y = alpha(2,:) .\* K\_y;

K\_z = alpha(3,:) .\* K\_z;

u\_x = x' + sum(K\_x,2);

u\_y = y' + sum(K\_y,2);

u\_z = z' + sum(K\_z,2);

end

function [K\_x,K\_y,K\_z] = kernel\_gaussian(x,y,z,control,sigma)

m = size(x,2); % source list of coords for each point

l = size(control,1);

r\_x = x'-control(:,1)';

r\_y = y'-control(:,2)';

r\_z = z'-control(:,3)';

% R = zeros(m,l);

% for ii = 1:(m\*l)

% R(ii) = norm([r\_x(ii) r\_y(ii) r\_z(ii) ]);

%

% end

K\_x = exp(-r\_x.^2 ./(2\*sigma^2));

K\_y = exp(-r\_y.^2 ./(2\*sigma^2));

K\_z = exp(-r\_z.^2 ./(2\*sigma^2));

end

end

end

Gaussian spline class

A spline fitting function fit, which uses: 1) A set of (pre-transformation) source points; 2) A set  
of (transformed) target points; 3) Weighting parameter lambda representing the  
approximated localisation errors; 4) Any parameter(s) required for the selected kernel. The  
function outputs the spline coefficients representing the fitted spline. [10]  
Report Q1: Is the polynomial part needed and why? [1]

Polynomials are equations of a single variable with nonnegative integer exponents. The poly function converts the roots back to polynomial coefficients.

Polynomials are, essentially by definition, precisely the operations one can write down starting from addition and multiplication. More formally, polynomials with coefficients in a commutative ring RR are precisely the morphisms in the Lawvere theory of commutative RR-algebras. So in some sense caring about polynomials is equivalent to caring about commutative rings and, more generally, commutative algebras.

Report Q2: Write down the linear algebra used to represent the spline fitting problem and  
the solution. Cite the key formula in the report in the relevant part of your code. [6]

Polynomial equations are equations that contain polynomial expressions on both sides of the equation. Here’s the standard form of a polynomial equation.

IMG_256

Note that  a­n, an-1, … ao can be any complex numbers for and the exponents can only be whole numbers for these to be considered as polynomial expressions.

Report Q3: What is the best linear algebra algorithm should be implemented to solve this  
spline fitting problem and why? [1]

 Having an equal sign followed by another polynomial expression is what makes the polynomial equation distinct from polynomial expressions.

As can be confirmed from the equation shown above, polynomial equation is said to be in standard form when the terms are arranged from the term with the highest power to the one with the lowest power.

o A evaluate function evaluate, which uses: 1) A set of query points that needed to be  
transformed to target space; 2) A set of control points; 3) The spline coefficients; 4) Any  
parameter(s) required for the selected kernel. The function output the transformed query  
point set. [8]  
Report Q4: What are the control points here? Can we choose any points as control points at  
evaluation stage, and why? [2]

Report Q5: Do we need the weighting parameter lambda at evaluation stage, and why? [2]

o A Gaussian kernel function kernel\_gaussian, which uses: 1) A set of query points; 2) A set of  
control points; 3) Any parameter(s) required for the selected kernel. The function outputs K,  
the kernel values between the query and the control point sets. [10]  
Report Q6: Describe the details of your vectorisation strategies for kernel computing for large  
point sets. [2]

Report Q7: Discussion of the utility of the Gaussian kernel parameter, sigma. [1]

Result presents the mean and the standard deviation for the percentage of improvement in the objective function of the best solution of the final iteration in relation to the best solution of the initial population. presents the mean and the standard deviation for the time spent to build the metamodel. It is important to present some comments regarding ∆% of y6  
function when n = 15; 20. Since the absolute difference between the current value and the initial value is very high,  
it resulted in a proportional improvement greater than 99:9% but not yet 100%. Although this 0:01% value is a  
small relative value, this proportional improvement represents a high absolute difference due to the characteristics  
of the function. The y6 function might assume high values when the number of variables increases. Finally, Fig. 1  
presents different charts, for each problem, in order to illustrate problem-dependent performance of the metamodels

Free-form deformation class

%

%

% Image3D Provide a demo of functionality.

%

% H = vol3d('CData',data) Create volume render object from input

% 3-D data. Use interp3 on data to increase volume

% rendering resolution. Returns a struct

% encapsulating the pseudo-volume rendering object.

% XxYxZ array represents scaled colormap indices.

% XxYxZx3 array represents truecolor RGB values for

% each voxel (along the 4th dimension).

%

% vol3d(...,'Alpha',alpha) XxYxZ array of alpha values for each voxel, in

% range [0,1]. Default: data (interpreted as

% scaled alphamap indices).

%

% vol3d(...,'Parent',axH) Specify parent axes. Default: gca.

%

% vol3d(...,'XData',x) 1x2 x-axis bounds. Default: [0 size(data, 2)].

% vol3d(...,'YData',y) 1x2 y-axis bounds. Default: [0 size(data, 1)].

% vol3d(...,'ZData',z) 1x2 z-axis bounds. Default: [0 size(data, 3)].

%

% vol3d(...,'texture','2D') Only render texture planes parallel to nearest

% orthogonal viewing plane. Requires doing

% vol3d(h) to refresh if the view is rotated

% (i.e. using cameratoolbar).

%

% vol3d(...,'texture','3D') Default. Render x,y,z texture planes

% simultaneously. This avoids the need to

% refresh the view but requires faster OpenGL

% hardware peformance.

%

% vol3d(H) Refresh view. Updates rendering of texture planes

% to reduce visual aliasing when using the 'texture'='2D'

% option.

%

% NOTES

% Use vol3dtool (from the original FreeFormDeformationFEX submission) for editing the

% colormap and alphamap. Adjusting these maps will allow you to explore

% your 3-D volume data at various intensity levels. See documentation on

% alphamap and colormap for more information.

%

% Use interp3 on input date to increase/decrease resolution of data

%

% Visualizing fluid flow

v = flow(50);

h = Image3D('cdata',v,'texture','2D');

view(3);

% Update view since 'texture' = '2D'

Image3D(h);

alphamap('rampdown'), alphamap('decrease'), alphamap('decrease')

% Visualizing MRI data

load mri.mat

D = squeeze(D);

h = Image3D('cdata',D,'texture','3D');

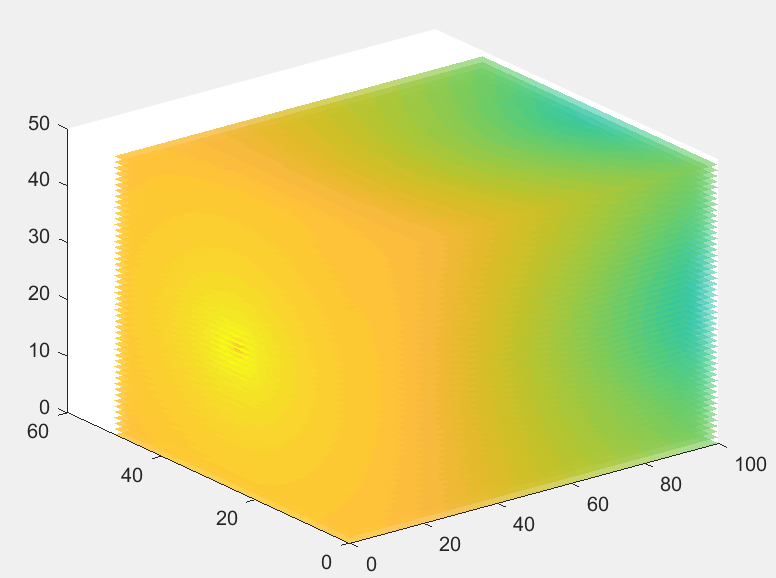
view(3);

axis tight; daspect([1 1 .4])

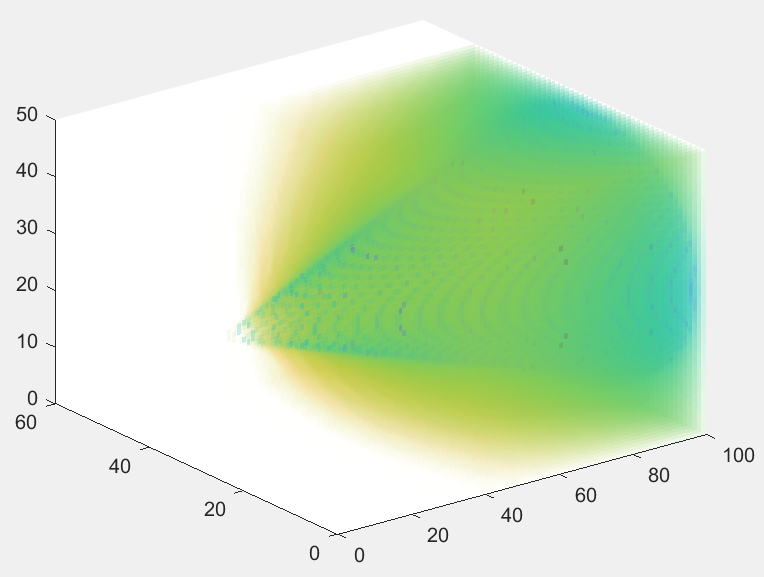
alphamap('rampup');

alphamap(.06 .\* alphamap);

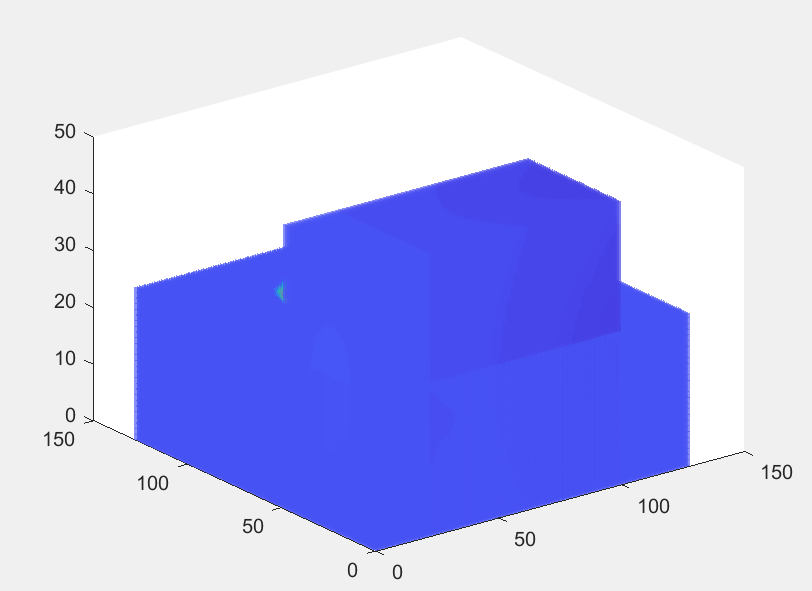
Load the example image in “data/example\_image.mat” and instantiate an Image3D object to  
store the read image data.



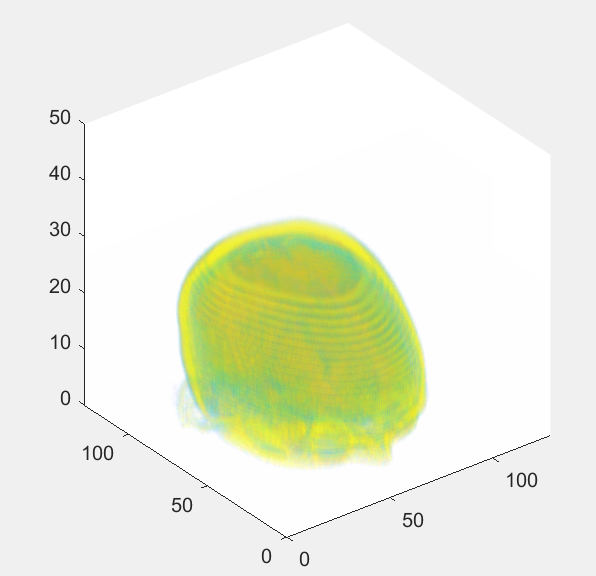
Generating 10 different randomly transformed images and plot 5 images for each  
transformed image at different z depths



Change the strength parameter in random\_transform\_generator; change the number of  
control points; and change the Gaussian kernel parameter. Visualise the randomly  
transformed images.



Summarise the effect of the three changes, with example images included in the  
report



I uses the orthogonal plane 2-D texture mapping technique for volume rending 3-D data in OpenGL. Use the 'texture' option to fine tune the texture mapping technique. This function is best used with fast OpenGL hardware.