Application of Reduced Gradient - Topology Optimization

Introduction

In this tutorial, you will learn to implement an optimization algorithm for minimizing the compliance of a structure at its equilibrium state with respect to its topology. The tutorial is based on Dr. Sigmund's topology optimization code and paper. The template code is attached at the end of the tutorial.

The compliance minimization problem

Topology optimization has been commonly used to design structures and materials with optimal mechanical, thermal, electromagnetic, acoustical, or other properties. The structure under design is segmented into n finite elements, and a density value x_i is assigned to each element $i \in 1, 2, \ldots, n$: A higher density corresponds to a less porous material element and higher Yong's modulus. Reducing the density to zero is equivalent to creating a hole in the structure. Thus, the set of densities $\mathbf{x} = x_i$ can be used to represent the topology of the structure and is considered as the variables to be optimized. A common topology optimization problem is compliance minimization, where we seek the "stiffest" structure within a certain volume limit to withhold a particular load:

$$\min_{\mathbf{x}} \quad \mathbf{f} := \mathbf{u}^T \mathbf{K}(\mathbf{x}) \mathbf{u}$$

$$\text{subject to:} \quad \mathbf{h} := \mathbf{K}(\mathbf{x}) \mathbf{u} = \mathbf{d},$$

$$\mathbf{g} := V(\mathbf{x}) \leq v,$$

$$\mathbf{x} \in [0, 1].$$

Here $V(\mathbf{x})$ is the total volume; v is an upper bound on volume; $\mathbf{u} \in \mathbb{R}^{n_d \times 1}$ is the displacement of the structure under the load $\mathbf{d}$, where n_d is the degrees of freedom (DOF) of the system (i.e., the number of x- and y-coordinates of nodes from the finite element model of the structure); $\mathbf{K}(\mathbf{x})$ is the global stiffness matrix for the structure.

$\mathbf{K}(\mathbf{x})$ is indirectly influenced by the topology $\mathbf{x}$, through the element-wise stiffness matrix

$$\mathbf{K}_i = \bar{\mathbf{K}}_e E(x_i),$$

and the local-to-global assembly:

$$\mathbf{K}(\mathbf{x}) = \mathbf{G}[\mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_n],$$

where the matrix $\bar{\mathbf{K}}_e$ is predefined according to the finite element type ($we use first-order quadrilateral elements throughout this tutorial$) and the $\mathbf{G}$ is an assembly matrix, $E(x_i)$ is the element-wise Young's modulus defined as a function of the density x_i: $E(x_i) := \Delta E x_i^p + E_{\text{min}}$, where p (the penalty parameter) is usually set to 3. This cubic relationship between the topology and the modulus is determined by the mate

Design sensitivity analysis

This problem has both inequality and equality constraints. However, the inequality ones are only related to the topology $\mathbf{x}$, and are either linear ($V(\mathbf{x}) \leq v$) or simple bounds ($\mathbf{x} \in [0,1]$). We will show that these constraints can be easily handled. The problem thus involves two sets of variables: We can consider $\mathbf{x}$ as the **decision variables** and $\mathbf{u}$ as the state variables that are governed by $\mathbf{x}$ through the equality constraint $\mathbf{K}(\mathbf{x})\mathbf{u} = \mathbf{d}$.

The reduced gradient (often called design sensitivity) can be calculated as

$$\frac{df}{d\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} - \frac{\partial f}{\partial \mathbf{u}}(\frac{\partial \mathbf{h}}{\partial \mathbf{u}})^{-1}\frac{\partial \mathbf{h}}{\partial \mathbf{x}},$$

which leads to

$$\frac{df}{d\mathbf{x}} = -\mathbf{u}^T\frac{\partial \mathbf{K}}{\partial \mathbf{x}}\mathbf{u}.$$

Recall the relation between $\mathbf{K}$ and $\mathbf{x}$, and notice that

$$\mathbf{u}^T\mathbf{K}\mathbf{u} = \sum_{i=1}^{n}\mathbf{u}_i^T\mathbf{K}_i\mathbf{u}_i,$$

i.e., the total compliance (strain energy) is the summation of element-wise compliance. We can further simplify the gradient as follows:

$$\begin{aligned}
-\mathbf{u}^T\frac{\partial \mathbf{K}}{\partial \mathbf{x}}\mathbf{u} &= -\frac{\partial \mathbf{u}^T\mathbf{K}\mathbf{u}}{\partial \mathbf{x}} \\
&= -\frac{\partial \sum_{i=1}^{n}\mathbf{u}_i^T\mathbf{K}_i\mathbf{u}_i}{\partial \mathbf{x}} \\
&= [\dots, -\frac{\partial \mathbf{u}_i^T\mathbf{K}_i\mathbf{u}_i}{\partial x_i}, \dots] \\
&= [\dots, -\mathbf{u}_i^T\frac{\partial \mathbf{K}_i}{\partial x_i}\mathbf{u}_i, \dots] \\
&= [\dots, -\mathbf{u}_i^T\frac{\partial \bar{\mathbf{K}}_e\Delta E x_i^3}{\partial x_i}\mathbf{u}_i, \dots] \\
&= [\dots, -3\Delta E x_i^2\mathbf{u}_i^T\bar{\mathbf{K}}_e\mathbf{u}_i, \dots]
\end{aligned}$$

The algorithm

The pseudo code for compliance minimization is the following:

1. Problem setup (see details below)
2. Algorithm setup: $\epsilon = 0.001$ (or any small positive number), $k = 0$ (counter for the iteration), $\Delta x = 1000$ (or any number larger than ϵ)
3. While $norm(\Delta x) \leq \epsilon$, Do:
   - 3.1. Update the stiffness matrix $\mathbf{K}$ and the displacement (state) $\mathbf{u}$ (finite element analysis)
   - 3.2. Calculate element-wise compliance $\mathbf{u}_i^T\bar{\mathbf{K}}_e\mathbf{u}_i$
   - 3.3. Calculate partial derivatives $\frac{df}{dx_i} = -3\Delta E x_i^2\mathbf{u}_i^T\bar{\mathbf{K}}_e\mathbf{u}_i$
   - 3.4. The gradient with respect to g is a constant vector $[1, \dots, 1]^T$
   - 3.5. Apply filter to $\frac{df}{d\mathbf{x}}$ (See discussion later)

$toasmallvalue, ortruncate$\Delta x = - (\frac{df}{d{\bf x}} + \mu {\bf 1})$ within a range (conceptually similar to the idea of trust region).

3.7. Move ${\bf x}'_{k+1} back to the feasible domain : If {\bf 1}^T{\bf x}{k} < va nd -{\bf 1}^T\frac{df}{d{\bf x}}<0, then{\bf x}'_{k+1} satisfies g. with \mu = 0. If {\bf x}{k+1} does not satisfy g, we will increase\mu using bisection, i.e., search in[0,\mu_{max}] where\mu_{max} is a large positive number. Also, we will truncate{\bf x}'_{k+1}$ between 0 and 1.

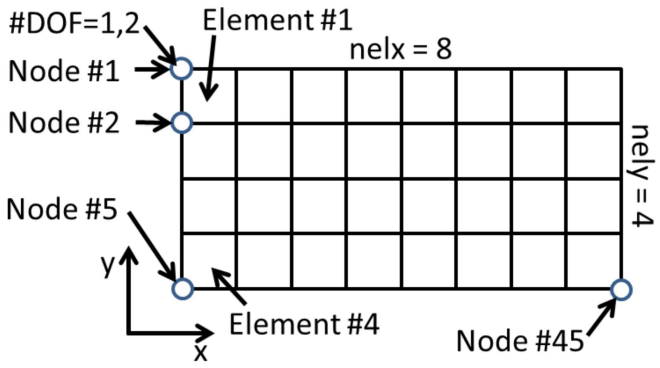
3.8. Update $norm(\Delta x)$, $k = k + 1$

Implementation details

Some details of the template code is explained as follows:

The numbering rule for elements and nodes

The template code assumes a rectangular design domain, where each element is modeled as a unit square. The numbering of elements and nodes are explained in the figure below.



Input parameters

Inputs to the program are (1) the number of elements in the x and y directions (`nelx` and `nely`), (2) the volume fraction (`volfrac`, this is the number between 0 and 1 that specifies the ratio between the volume of the target topology and the maximum volume $nelx \times nely$), (3) the penalty parameter of the Young's Modulus model (`penal`, usually **=3**), and (4) the filter radius (`rmin`, usually **=3**).

Material properties

```
%% MATERIAL PROPERTIES
E0 = 1;
Emin = 1e-9;
nu = 0.3;
```

Set the Young's Modulus (`E0`), and the Poisson's ratio (`nu`). Leave Emin as a small number.

Define loadings

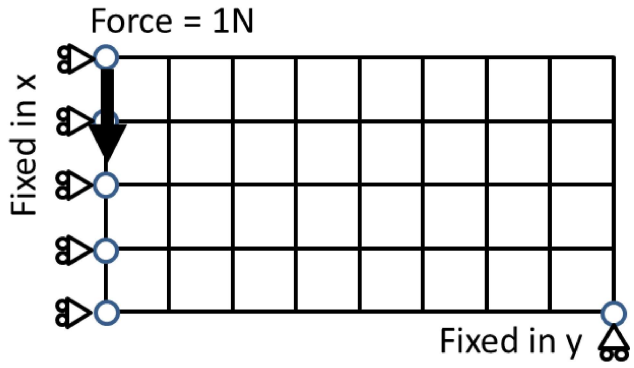
```
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
```

This line specifies the loading. Here `F` is a sparse column vector with `2(nely+1)(nelx+1)` elements. `(2,1,-1,\cdots)` specifies that there is a force of **-1** at the second row and first column of the vector. According to the numbering convention of this code, this is to say that in the y direction of the first node, there is a downward force of magnitude 1.

Define boundary conditions (Fixed DOFs)

```
fixeddofs = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
```

This line specifies the nodes with fixed DOFs. `[1:2:2*(nely+1)]` are x directions of all nodes to the left side of the structure, and `2*(nelx+1)*(nely+1)` is the y direction of the last node (right bottom corner). See figure below:



Filtering of sensitivity (gradient)

Pure gradient descent may result in a topology with checkerboard patterns. See figure below. While mathematically sound, such a solution can be infeasible from a manufacturing perspective or too expensive to realize (e.g., through additive manufacturing of porous structures). Therefore, a smoothed solution is often more preferred.

Drawing

In the template we prepare a Gaussian filter, through the following code:

```
%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
  for j1 = 1:nely
    e1 = (i1-1)*nely+j1;
    for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
      for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
        e2 = (i2-1)*nely+j2;
        k = k+1;
        iH(k) = e1;
        jH(k) = e2;
        sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
      end
    end
  end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
```

The design sensitivity can then be filtered by

```
dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
```

The template code

```
%%%% Modified by Max Yi Ren (ASU) %%%%%%%%%%%%%%%%%%%%%%

%%%% AN 88 LINE TOPOLOGY OPTIMIZATION CODE Nov, 2010 %%%%
function top88(nelx,nely,volfrac,penal,rmin,ft)

%% MATERIAL PROPERTIES
E0 = 1;
Emin = 1e-9;
nu = 0.3;
%% PREPARE FINITE ELEMENT ANALYSIS
```


```
A12 = [-6 -3  0  3; -3 -6 -3 -6;  0 -3 -6  3;  3 -6  3 -6];
B11 = [-4  3 -2  9;  3 -4 -9  4; -2 -9 -4 -3;  9  4 -3 -4];
B12 = [ 2 -3  4 -9; -3  2  9 -2;  4  9  2  3; -9 -2  3  2];
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nelx*nely,1);
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
F = sparse(2,1,-1,2*(nely+1)*(nelx+1),1);
U = zeros(2*(nely+1)*(nelx+1),1);
fixeddofs = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);

%% PREPARE FILTER
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
  for j1 = 1:nely
    e1 = (i1-1)*nely+j1;
    for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
      for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
        e2 = (i2-1)*nely+j2;
        k = k+1;
        iH(k) = e1;
        jH(k) = e2;
        sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
      end
    end
  end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);

%% INITIALIZE ITERATION
x = repmat(volfrac,nely,nelx);
xPhys = x;
loop = 0;
change = 1;

%% START ITERATION
while change > 0.01
  loop = loop + 1;

  %% FE-ANALYSIS
  K = ?;
  U(freedofs) = ?;

  %% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  ce = ?; % element-wise strain energy
  c = ?; % total strain energy
  dc = ?; % design sensitivity
  dv = ones(nely,nelx);

  %% FILTERING/MODIFICATION OF SENSITIVITIES
  if ft == 1
    dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
  elseif ft == 2
    dc(:) = H*(dc(:)./Hs);
    dv(:) = H*(dv(:)./Hs);
  end

  %% OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL DENSITIES
  while ?
```

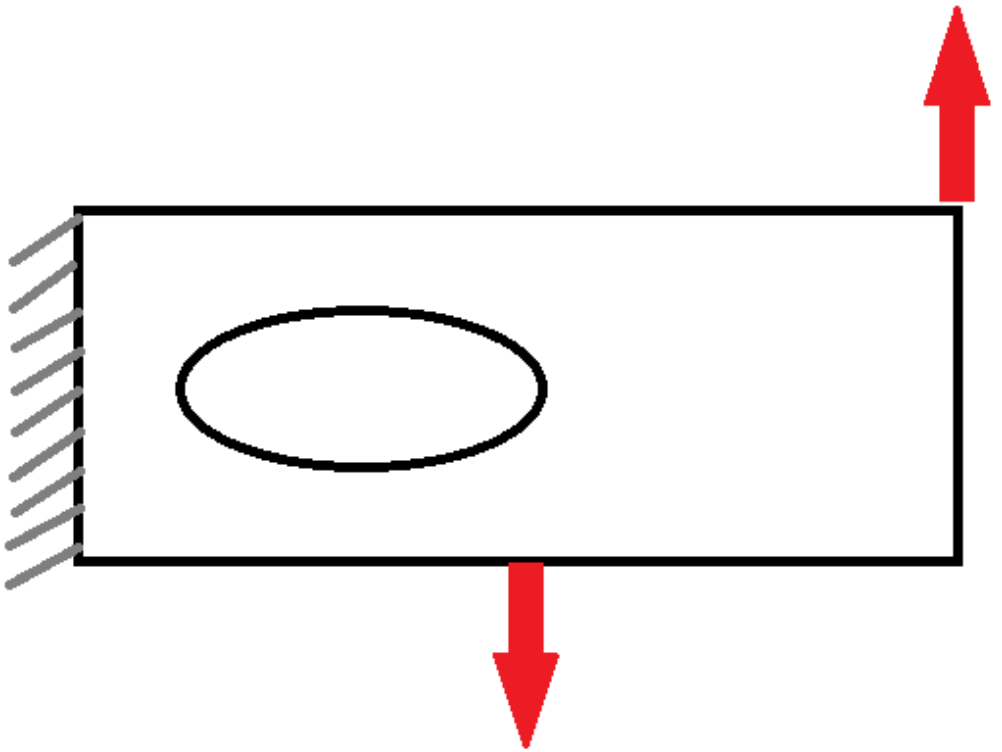
```
    change = max(abs(xnew(:)-x(:)));
    x = xnew;

    %% PRINT RESULTS
    fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c, ...
      mean(xPhys(:)),change);

    %% PLOT DENSITIES
    colormap(gray); imagesc(1-xPhys); caxis([0 1]); axis equal; axis off; drawnow;
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This Matlab code was written by E. Andreassen, A. Clausen, M. Schevenels,%
% B. S. Lazarov and O. Sigmund,  Department of Solid  Mechanics,           %
%  Technical University of Denmark,                                       %
%  DK-2800 Lyngby, Denmark.                                               %
% Please sent your comments to: sigmund@fam.dtu.dk                        %
%                                                                         %
% The code is intended for educational purposes and theoretical details   %
% are discussed in the paper                                              %
% "Efficient topology optimization in MATLAB using 88 lines of code,      %
% E. Andreassen, A. Clausen, M. Schevenels,                               %
% B. S. Lazarov and O. Sigmund, Struct Multidisc Optim, 2010               %
% This version is based on earlier 99-line code                           %
% by Ole Sigmund (2001), Structural and Multidisciplinary Optimization,   %
% Vol 21, pp. 120--127.                                                   %
%                                                                         %
% The code as well as a postscript version of the paper can be            %
% downloaded from the web-site: http://www.topopt.dtu.dk                  %
%                                                                         %
% Disclaimer:                                                             %
% The authors reserves all rights but do not guaranty that the code is    %
% free from errors. Furthermore, we shall not be liable in any event      %
% caused by the use of the program.                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Local density constraint

Local density constraint has been discussed in Wu. et al. This formulation of topology optimization derives more porous-like structures that are robust against local defects.

Here is an implementation of the algorithm from the paper, using an Augmented Lagrangian Method.

GEOMETRY CONSIDERED WITH BOUNDARY CONDITIONS



Contents

DESIGN OPTIMIZATION PROJECT 2 - TOPOLOGY OPTIMIZATION

```
%%%% Modified by Max Yi Ren (ASU) %%%%%%%%%%%%%%%%%%%%%%%%

%%%% AN 88 LINE TOPOLOGY OPTIMIZATION CODE Nov, 2010 %%%%
function top88(nelx,nely,volfrac,penal,rmin,ft)
```

PRESET VALUES

```
nelx=90;
nely=30;
volfrac=.5;
penal=3;
rmin=2;
ft =1;
```

MATERIAL PROPERTIES

```
E0 = 1;
Emin = 1e-9;
nu = 0.3;
```

PREPARE FINITE ELEMENT ANALYSIS

```
A11 = [12  3 -6 -3;  3 12  3  0; -6  3 12 -3; -3  0 -3 12];
A12 = [-6 -3  0  3; -3 -6 -3 -6;  0 -3 -6  3;  3 -6  3 -6];
B11 = [-4  3 -2  9;  3 -4 -9  4; -2 -9 -4 -3;  9  4 -3 -4];
B12 = [ 2 -3  4 -9; -3  2  9 -2;  4  9  2  3; -9 -2  3  2];
KE = 1/(1-nu^2)/24*([A11 A12;A12' A11]+nu*[B11 B12;B12' B11]);
nodenrs = reshape(1:(1+nelx)*(1+nely),1+nely,1+nelx);
edofVec = reshape(2*nodenrs(1:end-1,1:end-1)+1,nelx*nely,1);
edofMat = repmat(edofVec,1,8)+repmat([0 1 2*nely+[2 3 0 1] -2 -1],nelx*nely,1);
```

```
iK = reshape(kron(edofMat,ones(8,1))',64*nelx*nely,1);
jK = reshape(kron(edofMat,ones(1,8))',64*nelx*nely,1);
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
F = sparse([2*(nely+1)*nelx+2,2*(nely+1)*(nelx/2)+2*(nely+1)],[1 2],[1 -1],2*(nely+1)*(nelx+1),2);
U = zeros(2*(nely+1)*(nelx+1),2);
fixeddofs = [1:2*nely+1];
alldofs = [1:2*(nely+1)*(nelx+1)];
freedofs = setdiff(alldofs,fixeddofs);
```

PREPARE FILTER

```
iH = ones(nelx*nely*(2*(ceil(rmin)-1)+1)^2,1);
jH = ones(size(iH));
sH = zeros(size(iH));
k = 0;
for i1 = 1:nelx
  for j1 = 1:nely
    e1 = (i1-1)*nely+j1;
    for i2 = max(i1-(ceil(rmin)-1),1):min(i1+(ceil(rmin)-1),nelx)
      for j2 = max(j1-(ceil(rmin)-1),1):min(j1+(ceil(rmin)-1),nely)
        e2 = (i2-1)*nely+j2;
        k = k+1;
        iH(k) = e1;
        jH(k) = e2;
        sH(k) = max(0,rmin-sqrt((i1-i2)^2+(j1-j2)^2));
      end
    end
  end
end
H = sparse(iH,jH,sH);
Hs = sum(H,2);
```

INITIALIZE ITERATION

```
x = repmat(volfrac,nely,nelx);
xPhys = x;
loop = 0;
change = 1;
```

START ITERATION

```
while change > 0.01
```

```
  loop = loop + 1;
```

FE-ANALYSIS

```
  sK = reshape(KE(:)*(Emin+xPhys(:)'.^penal*(E0-Emin)),64*nelx*nely,1);
  K = sparse(iK,jK,sK);
  K = (1/2)*(K+K');
  U(freedofs, :) =K(freedofs,freedofs)\F(freedofs, :);
```

OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS

```
c=0;
dc=0;
for elx=1:size(F,2)
Ui=U(:,elx);
ce=reshape(sum((Ui(edofMat)*KE).*Ui(edofMat),2), nely,nelx);
c=c+sum(sum((Emin+xPhys.^penal*(E0-Emin)).*ce));
dc=dc-penal*(E0-Emin)*xPhys.^(penal-1).*ce;
end
dv = ones(nely,nelx);
```

FILTERING/MODIFICATION OF SENSITIVITIES

```
if ft == 1
  dc(:) = H*(x(:).*dc(:))./Hs./max(1e-3,x(:));
elseif ft == 2
  dc(:) = H*(dc(:)./Hs);
  dv(:) = H*(dv(:)./Hs);
end
```

OPTIMALITY CRITERIA UPDATE OF DESIGN VARIABLES AND PHYSICAL DENSITIES

```
ls = 0;
le = 1e9;
m = 0.2;
ellipse=zeros(nely,nelx);
for elx = 1:nelx
 for ely = 1:nely
  if sqrt((((ely-nely/2)^2)/.4)+(((elx-nelx/3)^2)/1.6)) < nely/2
     ellipse(ely,elx) = 1;
  end
 end
end
while (le-ls)/(ls+le) > 1e-3
  lc = (1/2)*(le+ls);
  xnew = max(0,max(x-m,min(1,min(x+m,x.*sqrt(-dc./dv/lc)))));
  xPhys = xnew;
  xPhys(ellipse==1) = 0;
  if sum(xPhys(:)) > volfrac*nelx*nely
      ls = lc;
  else
      le = lc;
  end
end
change = max(abs(xnew(:)-x(:)));
x = xnew;
```

PRINT RESULTS

```
fprintf(' It.:%5i Obj.:%11.4f Vol.:%7.3f ch.:%7.3f\n',loop,c, mean(xPhys(:)),change);
```

```
It.:    1 Obj.:  1153.4708 Vol.:  0.500 ch.:  0.200

It.:    2 Obj.:   897.5000 Vol.:  0.500 ch.:  0.200

It.:    3 Obj.:   528.5137 Vol.:  0.500 ch.:  0.200

It.:    4 Obj.:   428.5412 Vol.:  0.500 ch.:  0.200

It.:    5 Obj.:   404.1470 Vol.:  0.500 ch.:  0.200

It.:    6 Obj.:   391.0556 Vol.:  0.500 ch.:  0.200

It.:    7 Obj.:   380.7910 Vol.:  0.500 ch.:  0.192

It.:    8 Obj.:   374.4093 Vol.:  0.500 ch.:  0.165

It.:    9 Obj.:   369.8465 Vol.:  0.500 ch.:  0.128

It.:   10 Obj.:   367.2186 Vol.:  0.500 ch.:  0.124

It.:   11 Obj.:   364.7805 Vol.:  0.500 ch.:  0.124

It.:   12 Obj.:   362.7396 Vol.:  0.500 ch.:  0.106

It.:   13 Obj.:   360.3134 Vol.:  0.500 ch.:  0.101

It.:   14 Obj.:   358.3772 Vol.:  0.500 ch.:  0.089

It.:   15 Obj.:   356.6177 Vol.:  0.500 ch.:  0.079

It.:   16 Obj.:   355.3643 Vol.:  0.500 ch.:  0.065

It.:   17 Obj.:   354.3815 Vol.:  0.500 ch.:  0.058
```

```
It.:   18 Obj.:   353.6774 Vol.:  0.500 ch.:  0.052

It.:   19 Obj.:   353.2771 Vol.:  0.500 ch.:  0.052

It.:   20 Obj.:   352.9358 Vol.:  0.500 ch.:  0.058

It.:   21 Obj.:   352.7411 Vol.:  0.500 ch.:  0.068

It.:   22 Obj.:   352.5941 Vol.:  0.500 ch.:  0.073

It.:   23 Obj.:   352.3897 Vol.:  0.500 ch.:  0.071

It.:   24 Obj.:   352.1815 Vol.:  0.500 ch.:  0.061

It.:   25 Obj.:   352.0551 Vol.:  0.500 ch.:  0.063

It.:   26 Obj.:   351.9162 Vol.:  0.500 ch.:  0.066

It.:   27 Obj.:   351.8269 Vol.:  0.500 ch.:  0.074

It.:   28 Obj.:   351.6903 Vol.:  0.500 ch.:  0.081

It.:   29 Obj.:   351.4656 Vol.:  0.500 ch.:  0.098

It.:   30 Obj.:   350.8597 Vol.:  0.500 ch.:  0.114

It.:   31 Obj.:   350.0952 Vol.:  0.500 ch.:  0.113

It.:   32 Obj.:   349.0751 Vol.:  0.500 ch.:  0.097

It.:   33 Obj.:   348.2861 Vol.:  0.500 ch.:  0.050

It.:   34 Obj.:   348.0412 Vol.:  0.500 ch.:  0.057
```

```
It.:   35 Obj.:   347.9529 Vol.:  0.500 ch.:  0.060

It.:   36 Obj.:   347.9835 Vol.:  0.500 ch.:  0.060

It.:   37 Obj.:   347.8416 Vol.:  0.500 ch.:  0.062

It.:   38 Obj.:   347.7770 Vol.:  0.500 ch.:  0.064

It.:   39 Obj.:   347.6970 Vol.:  0.500 ch.:  0.065

It.:   40 Obj.:   347.6155 Vol.:  0.500 ch.:  0.067

It.:   41 Obj.:   347.4046 Vol.:  0.500 ch.:  0.067

It.:   42 Obj.:   347.2817 Vol.:  0.500 ch.:  0.067

It.:   43 Obj.:   347.1328 Vol.:  0.500 ch.:  0.064

It.:   44 Obj.:   346.9751 Vol.:  0.500 ch.:  0.063

It.:   45 Obj.:   346.7282 Vol.:  0.500 ch.:  0.055

It.:   46 Obj.:   346.6526 Vol.:  0.500 ch.:  0.059

It.:   47 Obj.:   346.5029 Vol.:  0.500 ch.:  0.055

It.:   48 Obj.:   346.4430 Vol.:  0.500 ch.:  0.050

It.:   49 Obj.:   346.3902 Vol.:  0.500 ch.:  0.045

It.:   50 Obj.:   346.3509 Vol.:  0.500 ch.:  0.043

It.:   51 Obj.:   346.3076 Vol.:  0.500 ch.:  0.041
```

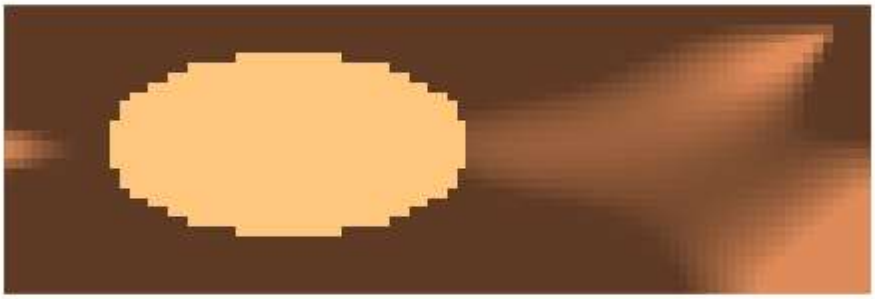
```
It.:   52 Obj.:   346.2616 Vol.:  0.500 ch.:  0.043

It.:   53 Obj.:   346.2097 Vol.:  0.500 ch.:  0.044

It.:   54 Obj.:   346.0298 Vol.:  0.500 ch.:  0.045

It.:   55 Obj.:   345.9290 Vol.:  0.500 ch.:  0.045

It.:   56 Obj.:   345.8374 Vol.:  0.500 ch.:  0.045

It.:   57 Obj.:   345.7434 Vol.:  0.500 ch.:  0.045

It.:   58 Obj.:   345.6613 Vol.:  0.500 ch.:  0.043

It.:   59 Obj.:   345.5920 Vol.:  0.500 ch.:  0.040

It.:   60 Obj.:   345.5319 Vol.:  0.500 ch.:  0.043

It.:   61 Obj.:   345.3734 Vol.:  0.500 ch.:  0.044

It.:   62 Obj.:   345.3174 Vol.:  0.500 ch.:  0.041

It.:   63 Obj.:   345.2692 Vol.:  0.500 ch.:  0.038

It.:   64 Obj.:   345.2483 Vol.:  0.500 ch.:  0.035

It.:   65 Obj.:   345.2335 Vol.:  0.500 ch.:  0.032

It.:   66 Obj.:   345.2213 Vol.:  0.500 ch.:  0.031

It.:   67 Obj.:   345.1025 Vol.:  0.500 ch.:  0.026

It.:   68 Obj.:   345.1765 Vol.:  0.500 ch.:  0.036
```

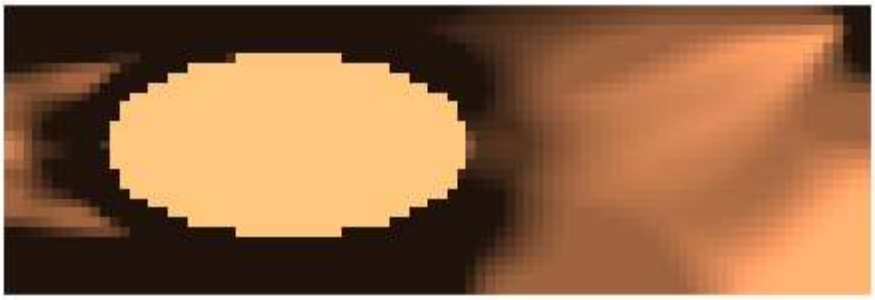
```
It.:   69 Obj.:   345.0580 Vol.:  0.500 ch.:  0.047

It.:   70 Obj.:   345.1418 Vol.:  0.500 ch.:  0.044

It.:   71 Obj.:   345.0344 Vol.:  0.500 ch.:  0.035

It.:   72 Obj.:   345.0173 Vol.:  0.500 ch.:  0.028

It.:   73 Obj.:   344.9944 Vol.:  0.500 ch.:  0.023

It.:   74 Obj.:   344.9802 Vol.:  0.500 ch.:  0.022

It.:   75 Obj.:   344.9653 Vol.:  0.500 ch.:  0.021

It.:   76 Obj.:   344.9513 Vol.:  0.500 ch.:  0.032

It.:   77 Obj.:   344.9411 Vol.:  0.500 ch.:  0.031

It.:   78 Obj.:   344.9321 Vol.:  0.500 ch.:  0.023

It.:   79 Obj.:   344.9201 Vol.:  0.500 ch.:  0.019

It.:   80 Obj.:   344.9089 Vol.:  0.500 ch.:  0.043

It.:   81 Obj.:   344.8998 Vol.:  0.500 ch.:  0.088

It.:   82 Obj.:   344.8876 Vol.:  0.500 ch.:  0.097

It.:   83 Obj.:   344.8797 Vol.:  0.500 ch.:  0.081

It.:   84 Obj.:   344.8708 Vol.:  0.500 ch.:  0.061

It.:   85 Obj.:   344.8616 Vol.:  0.500 ch.:  0.047
```

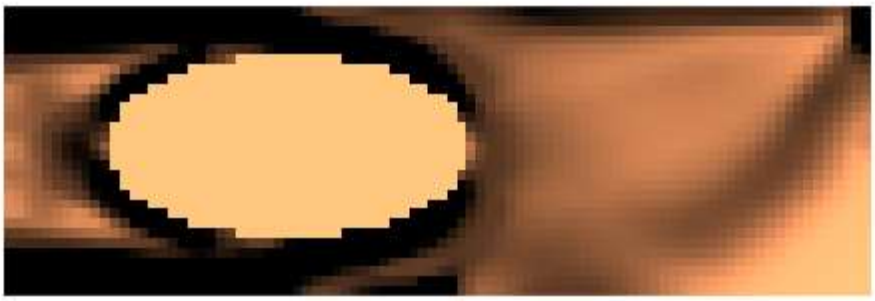
```
It.:   86 Obj.:   344.8531 Vol.:  0.500 ch.:  0.037

It.:   87 Obj.:   344.8439 Vol.:  0.500 ch.:  0.030

It.:   88 Obj.:   344.8364 Vol.:  0.500 ch.:  0.026

It.:   89 Obj.:   344.8331 Vol.:  0.500 ch.:  0.022

It.:   90 Obj.:   344.8277 Vol.:  0.500 ch.:  0.019

It.:   91 Obj.:   344.8210 Vol.:  0.500 ch.:  0.018

It.:   92 Obj.:   344.8136 Vol.:  0.500 ch.:  0.017

It.:   93 Obj.:   344.8087 Vol.:  0.500 ch.:  0.017

It.:   94 Obj.:   344.8020 Vol.:  0.500 ch.:  0.017

It.:   95 Obj.:   344.6866 Vol.:  0.500 ch.:  0.015

It.:   96 Obj.:   344.7619 Vol.:  0.500 ch.:  0.014

It.:   97 Obj.:   344.7604 Vol.:  0.500 ch.:  0.015

It.:   98 Obj.:   344.6586 Vol.:  0.500 ch.:  0.012

It.:   99 Obj.:   344.7393 Vol.:  0.500 ch.:  0.013

It.:  100 Obj.:   344.6328 Vol.:  0.500 ch.:  0.011

It.:  101 Obj.:   344.7218 Vol.:  0.500 ch.:  0.011

It.:  102 Obj.:   344.6156 Vol.:  0.500 ch.:  0.010
```

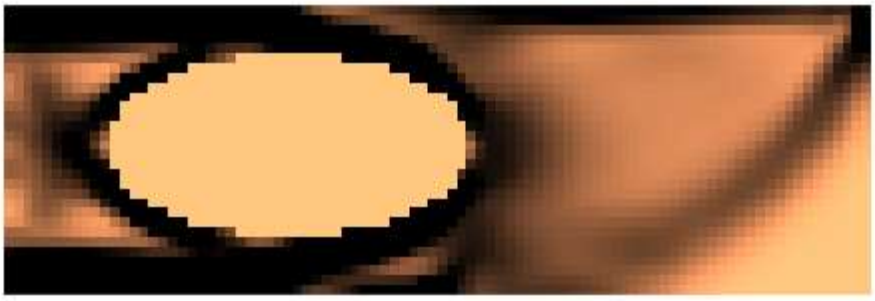
PLOT DENSITIES

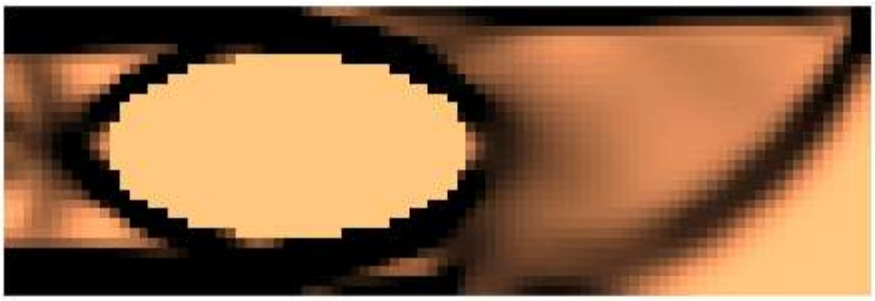
```
colormap("copper"); imagesc(1-xPhys); caxis([0 1]); axis equal; axis off; drawnow;
```

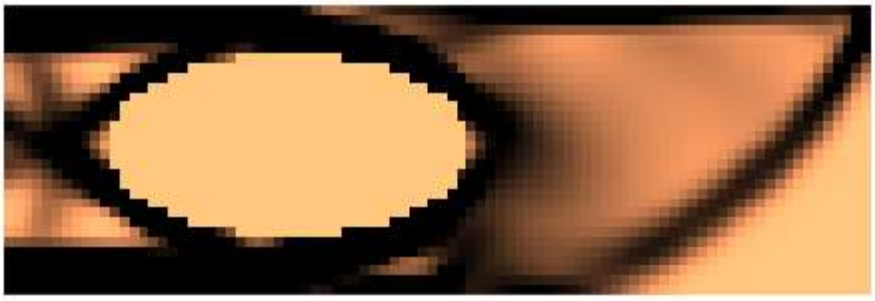


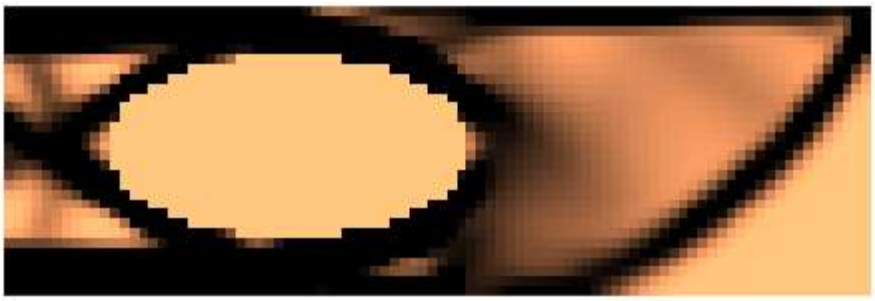


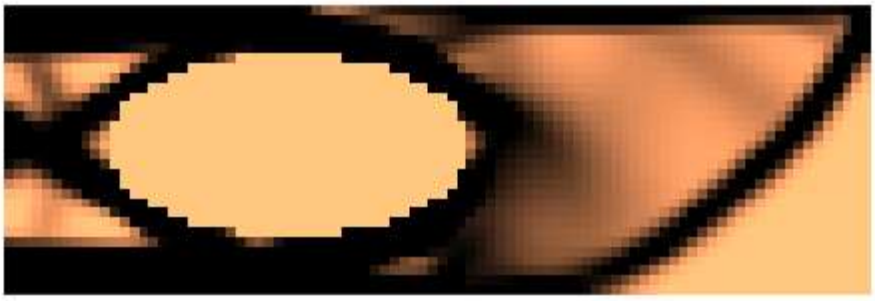


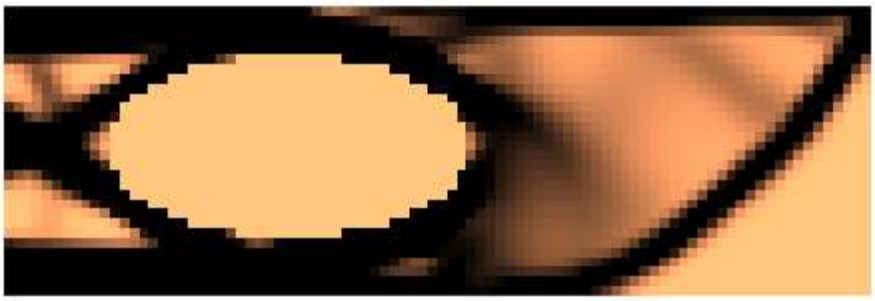


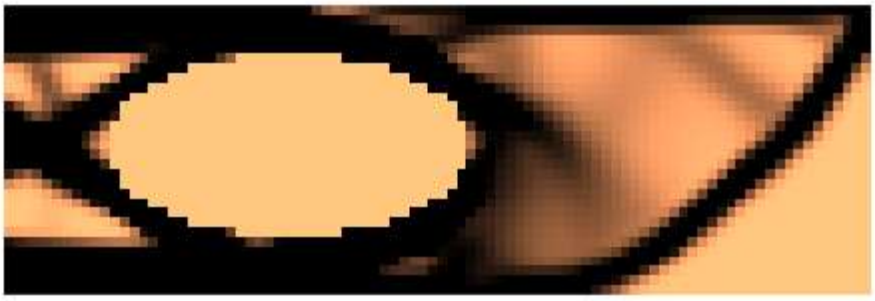


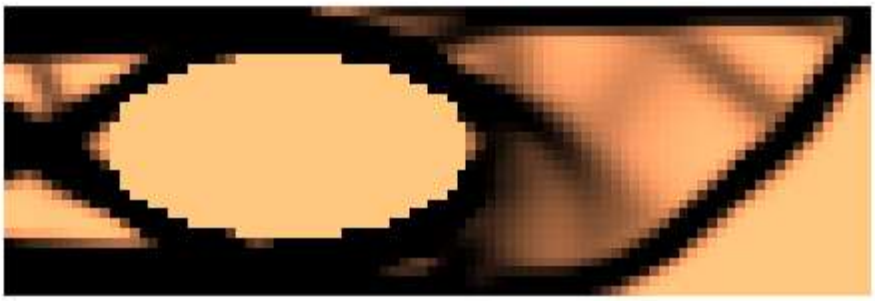


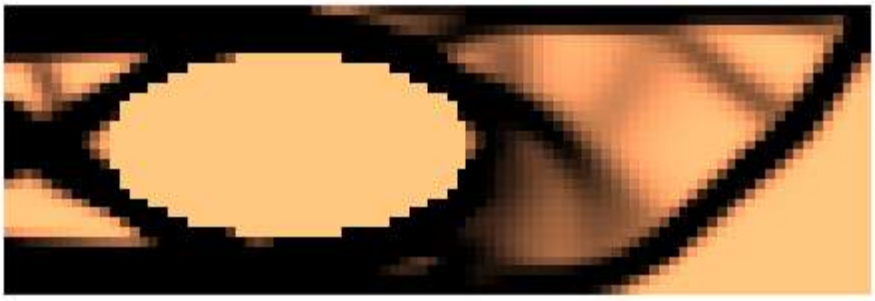


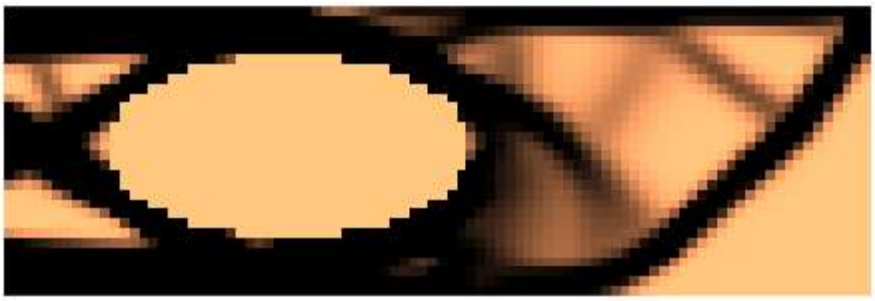


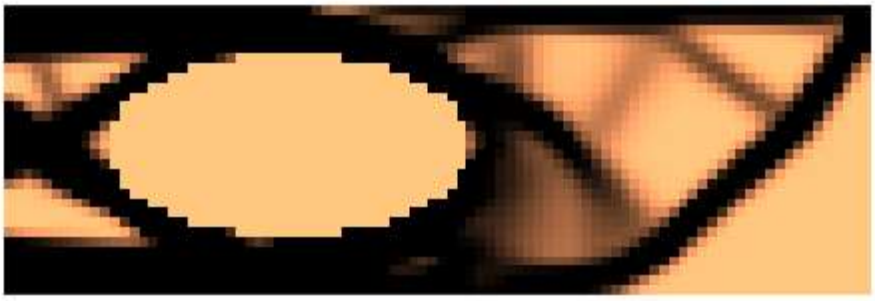


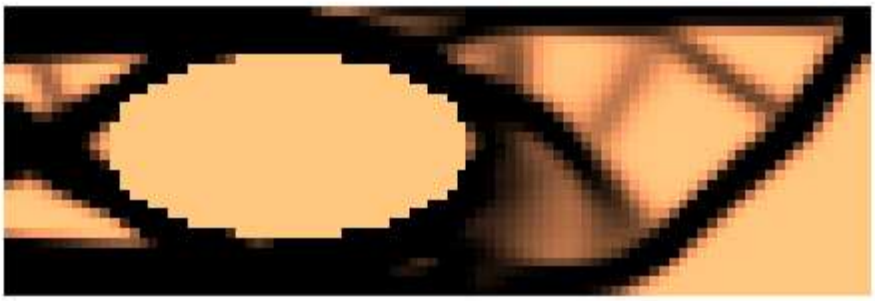


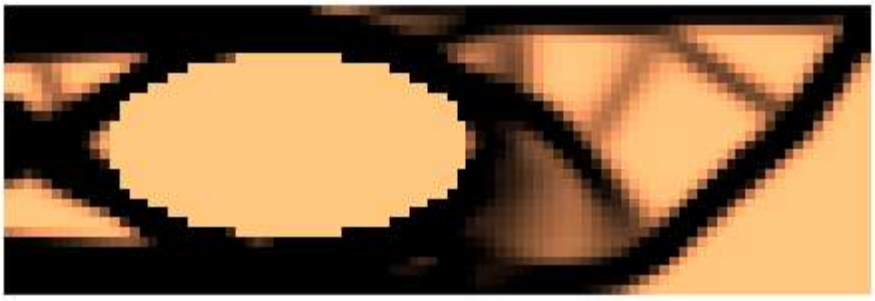


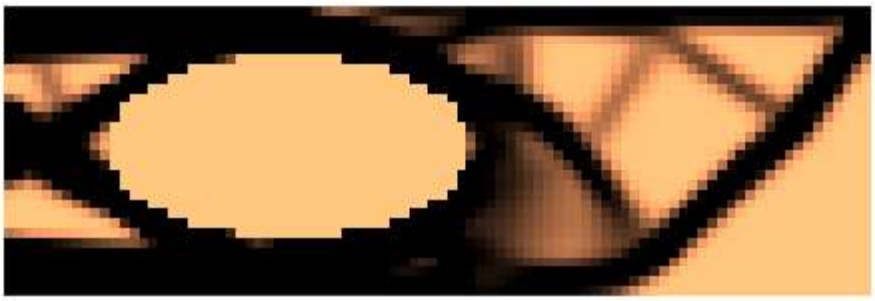


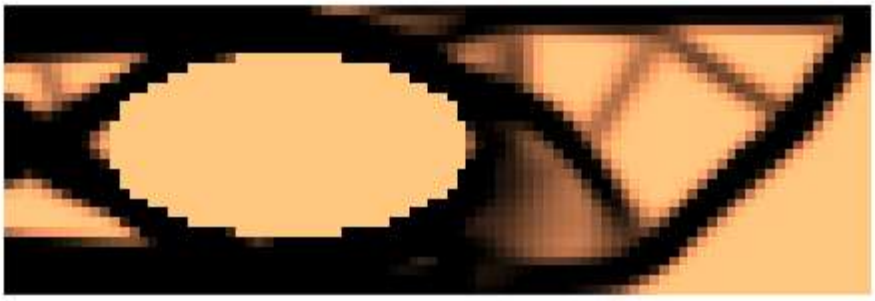


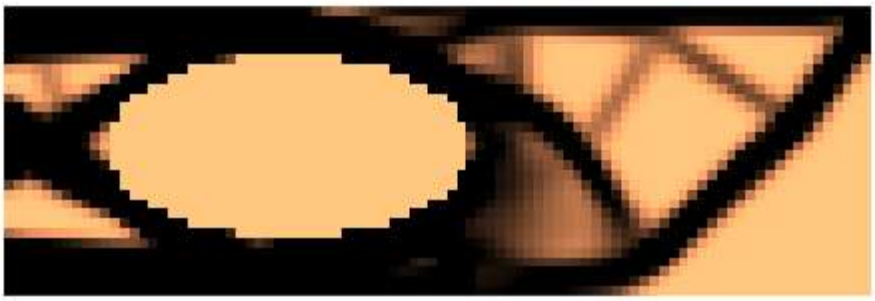


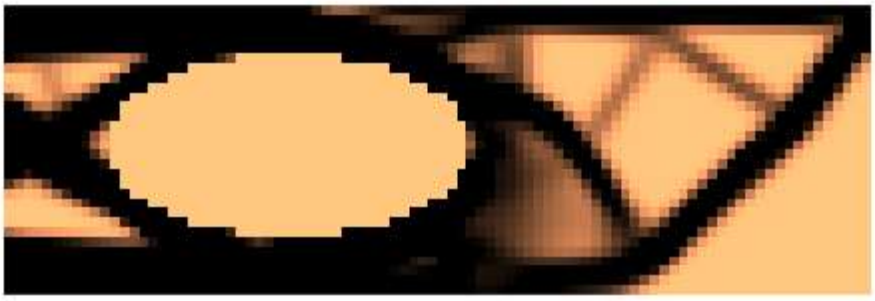


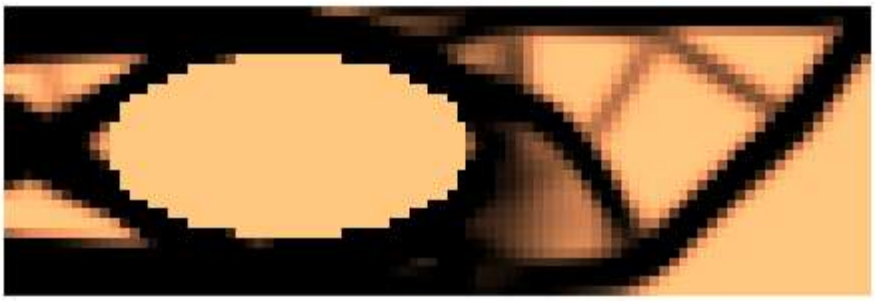


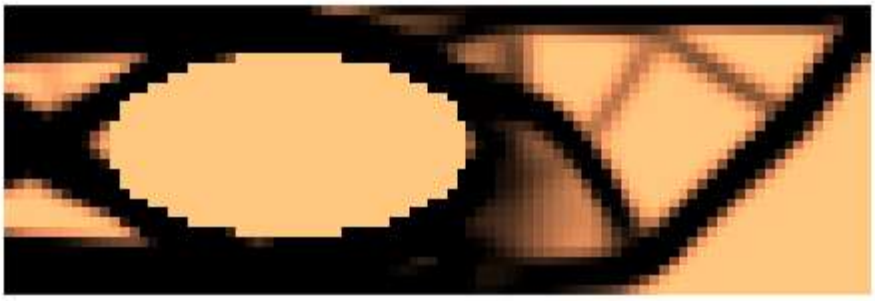


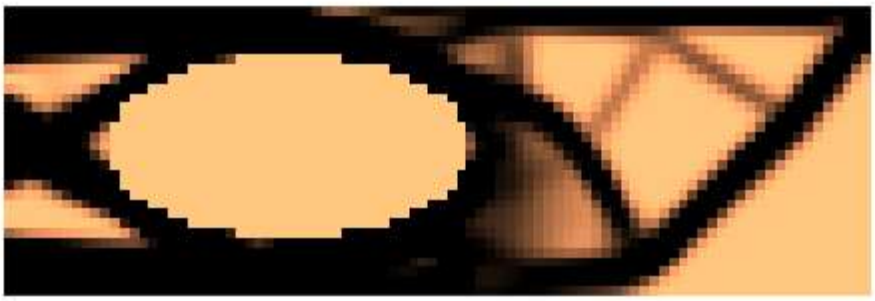


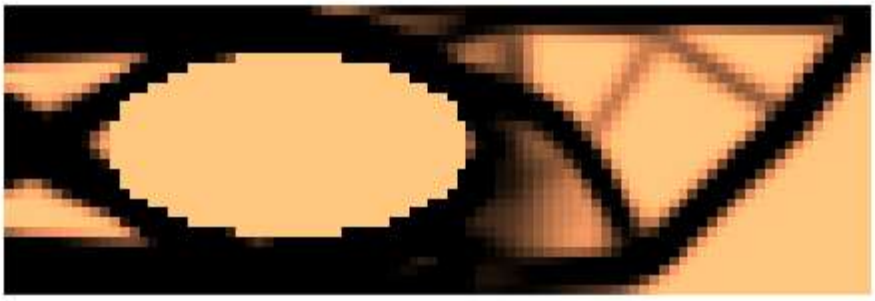


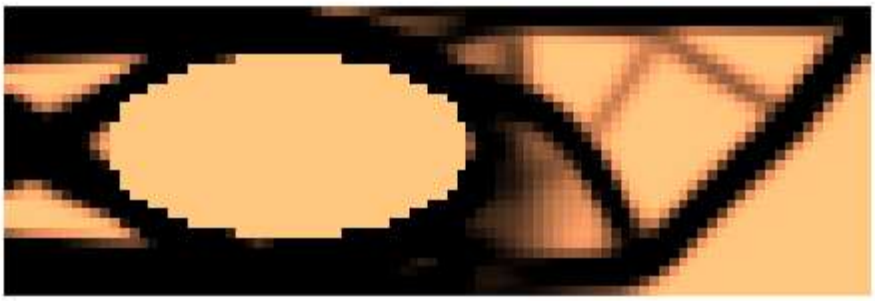


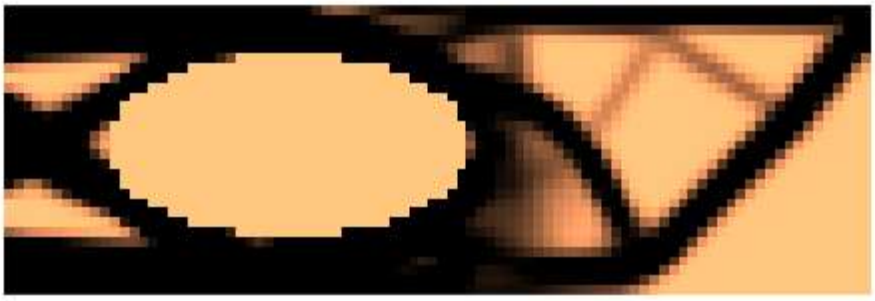


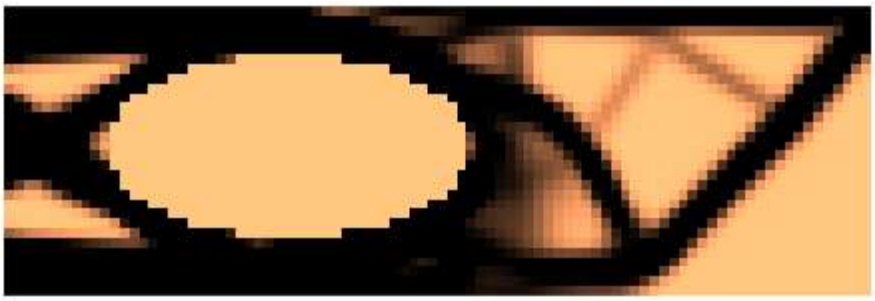


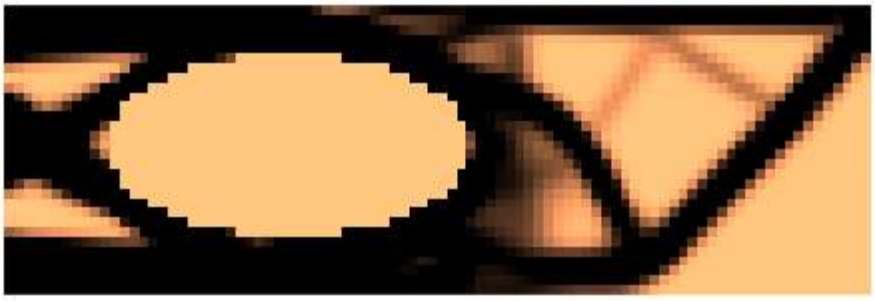


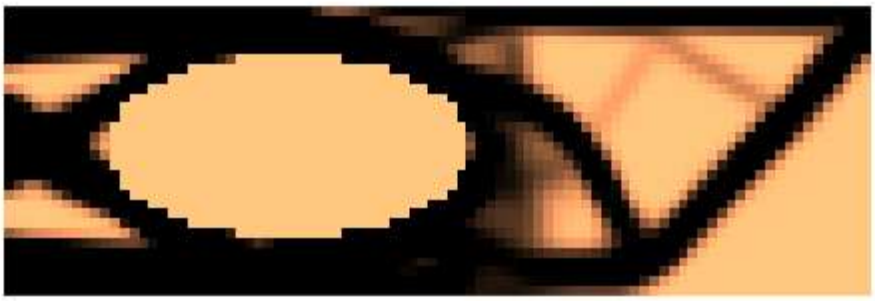


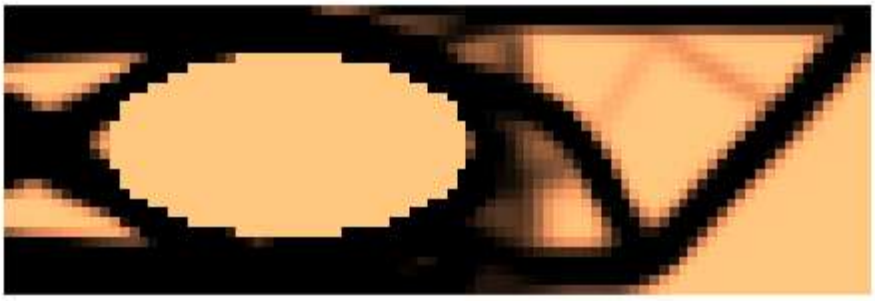


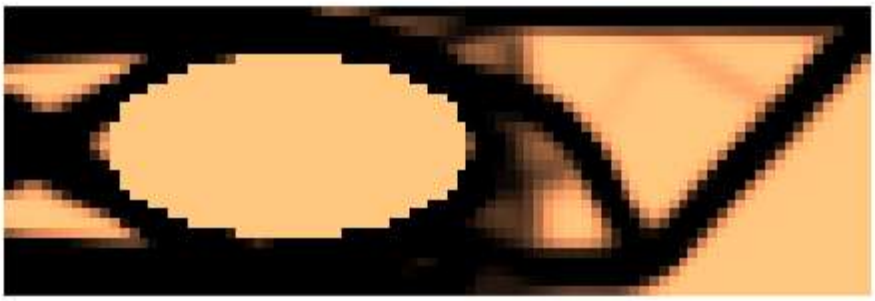


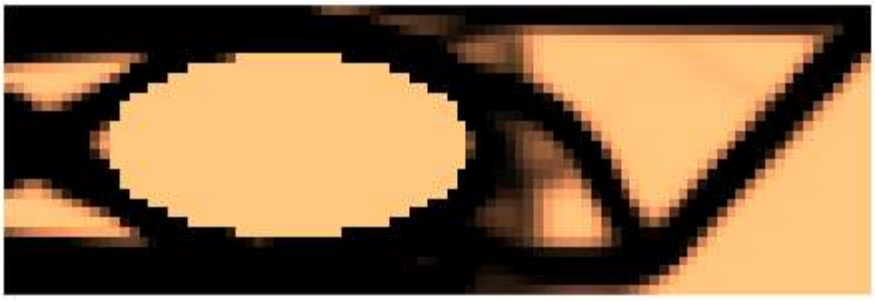


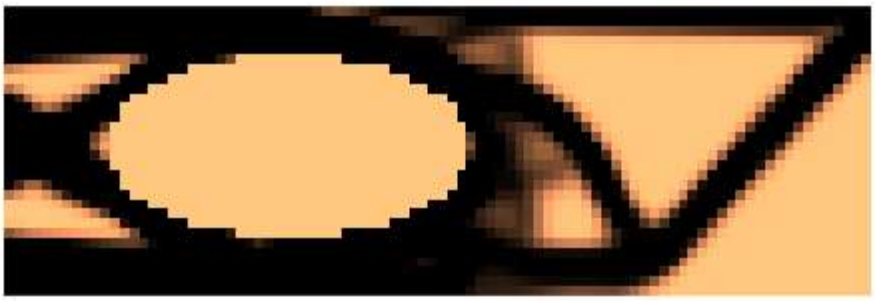


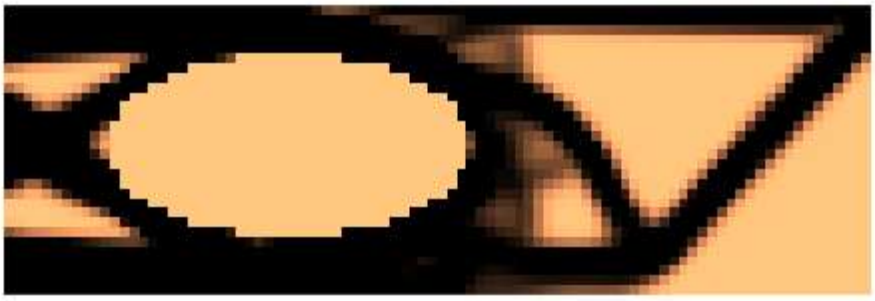


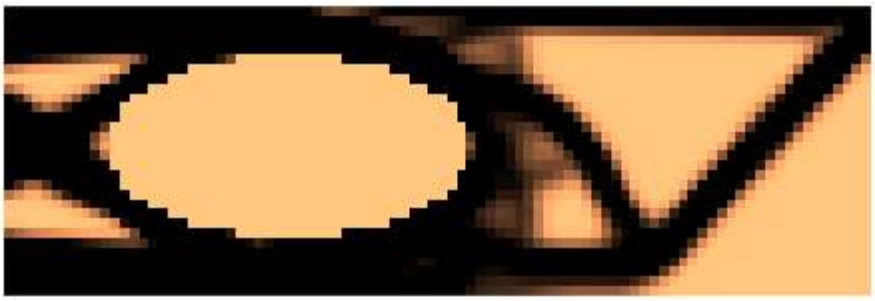


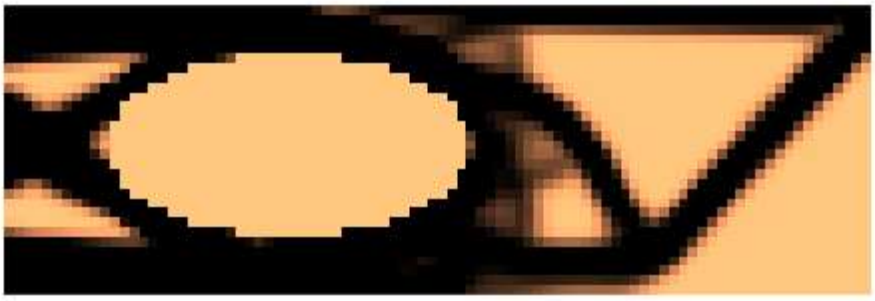


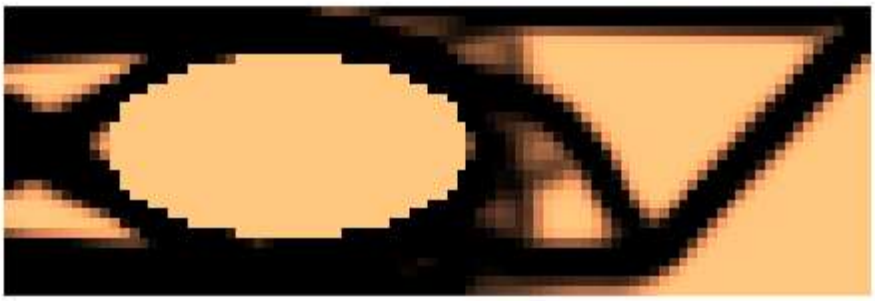


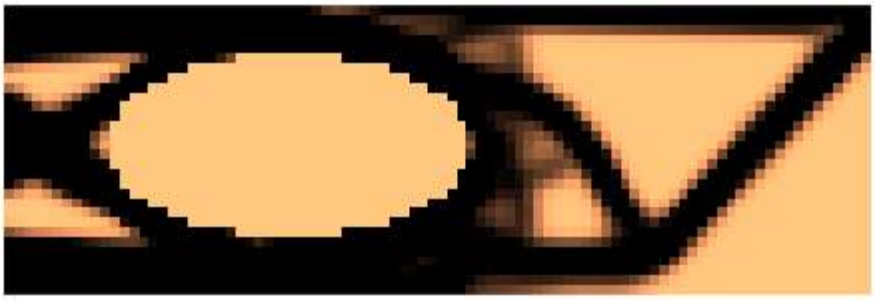


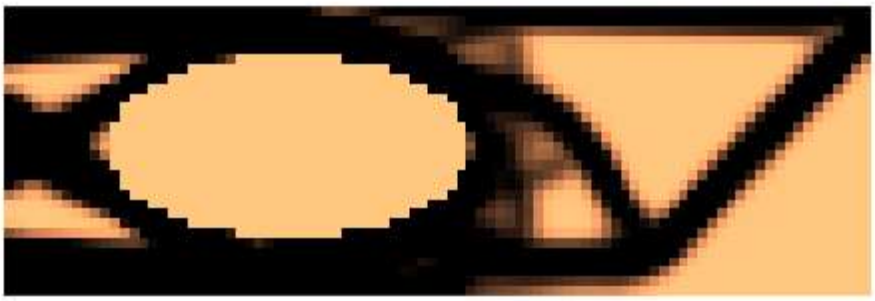


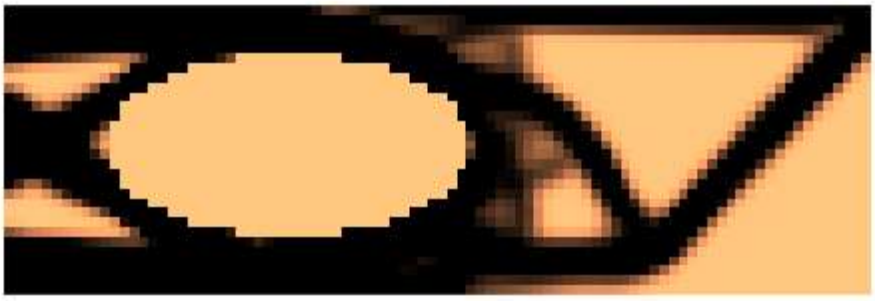


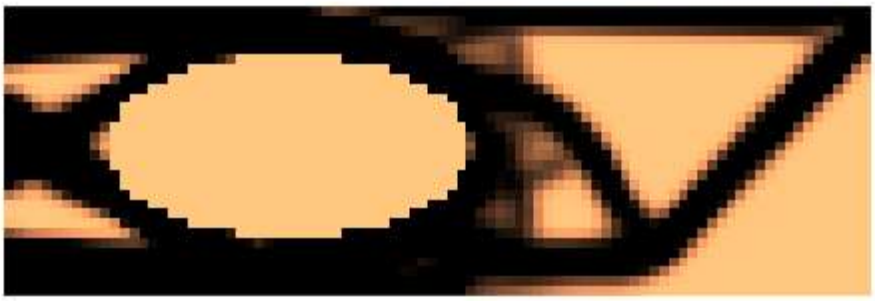


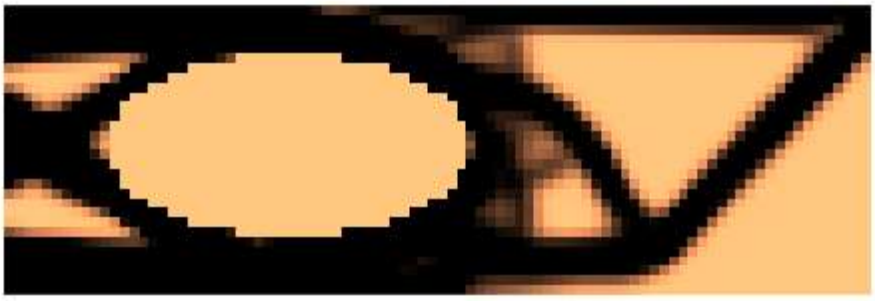


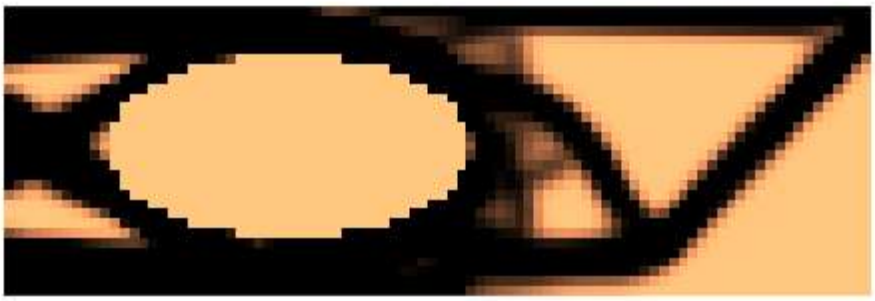


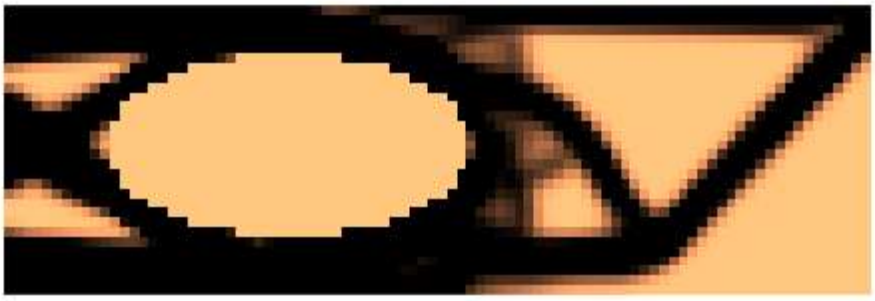


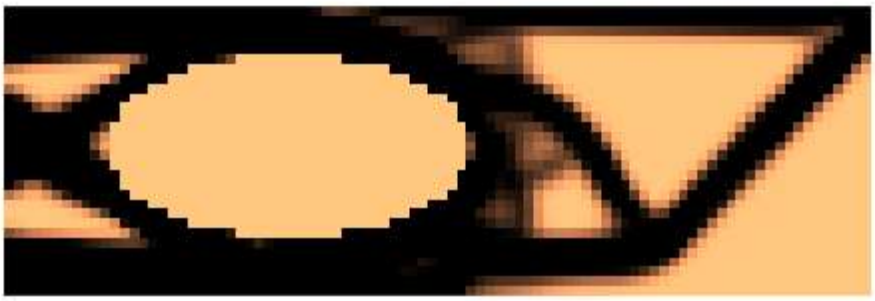


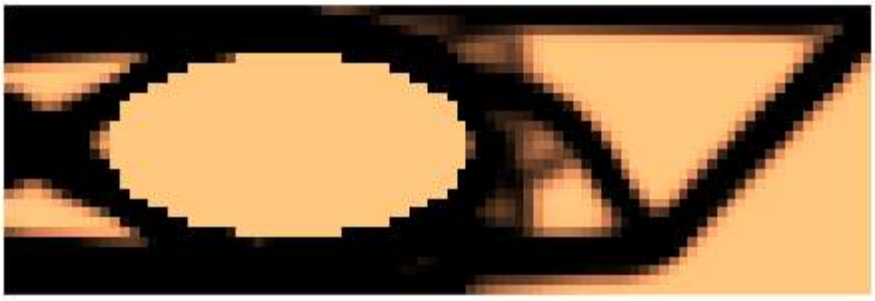


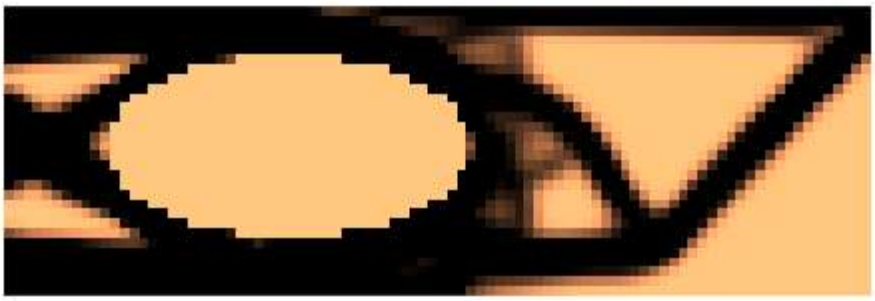


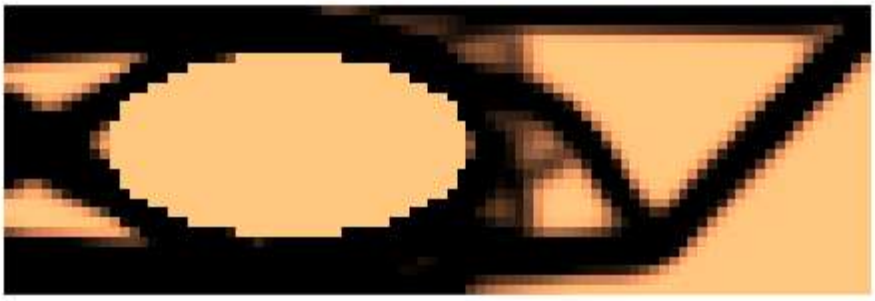


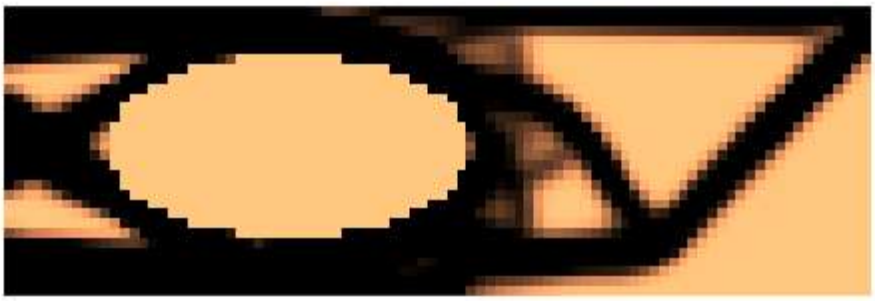


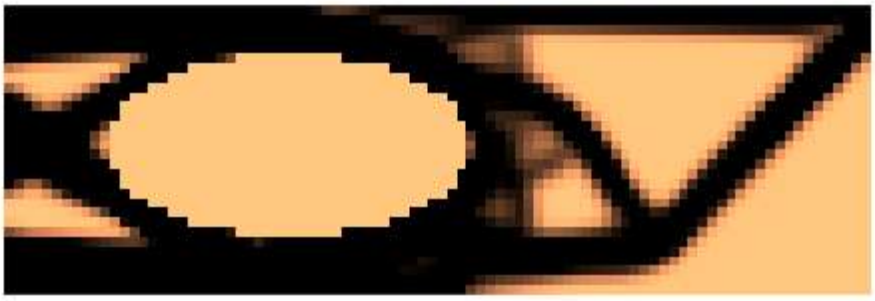


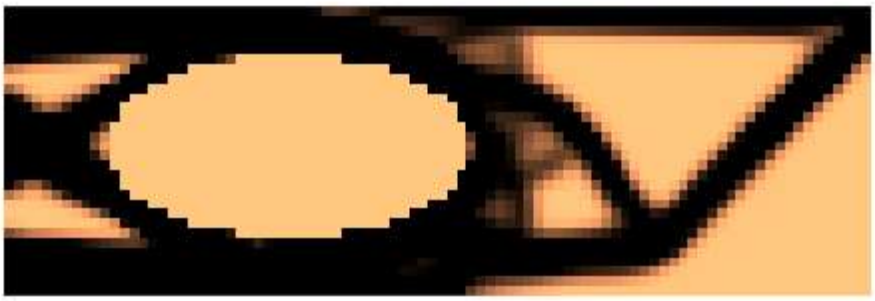


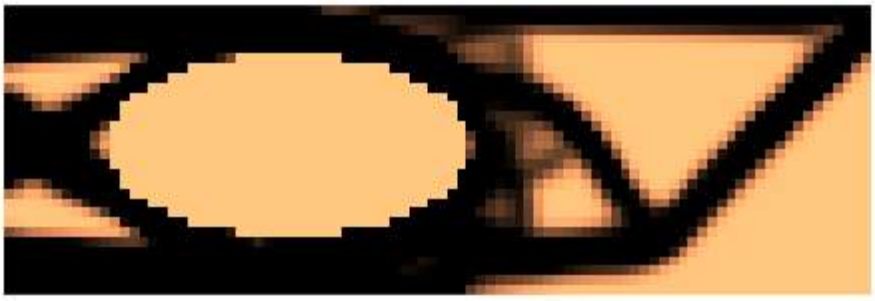


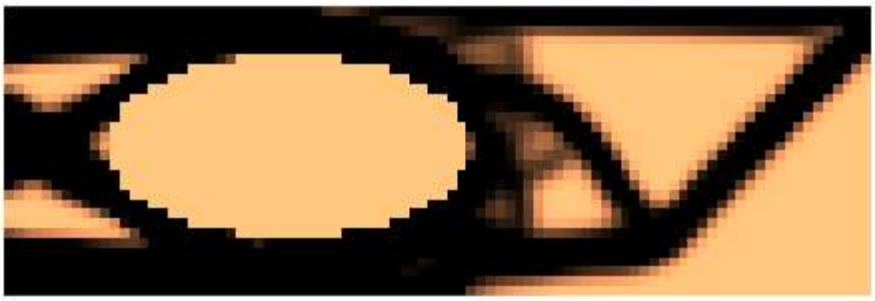


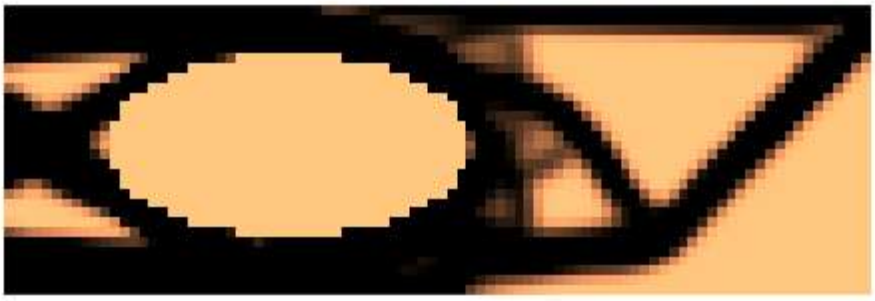


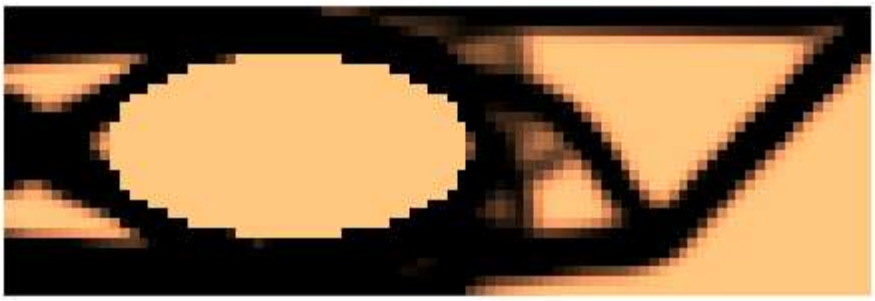


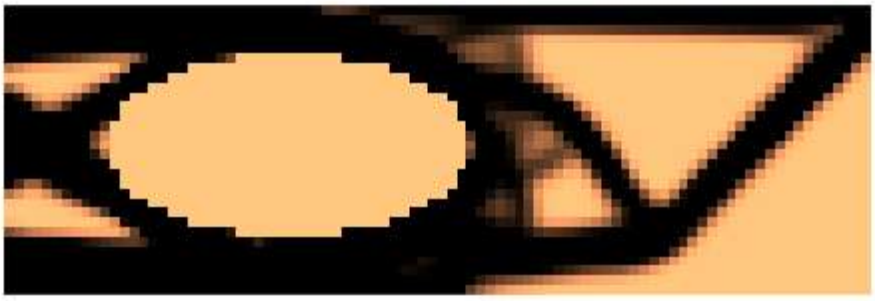


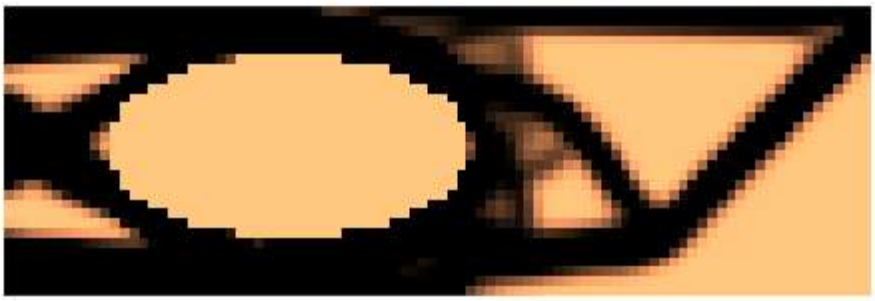


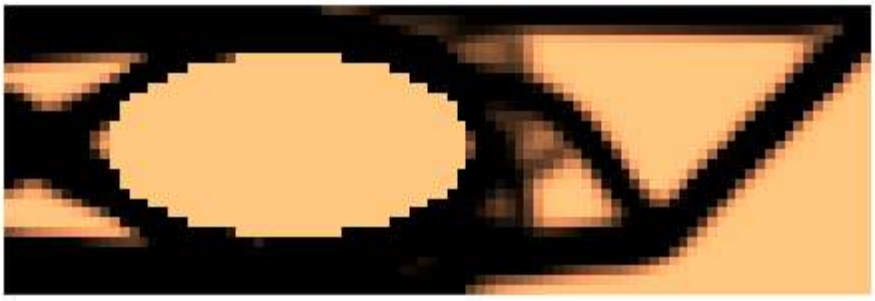


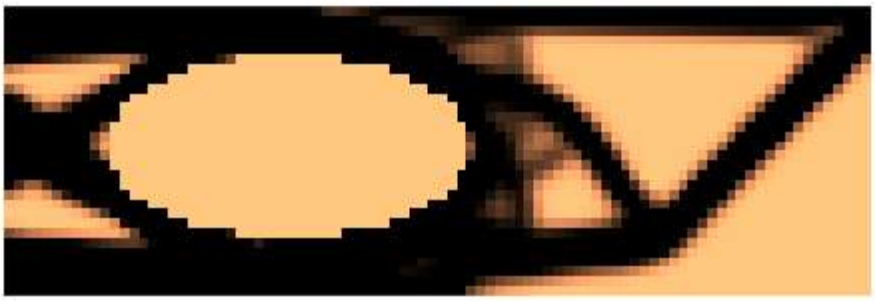


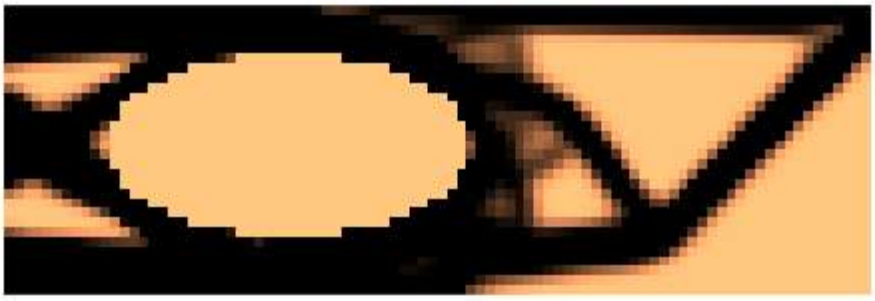


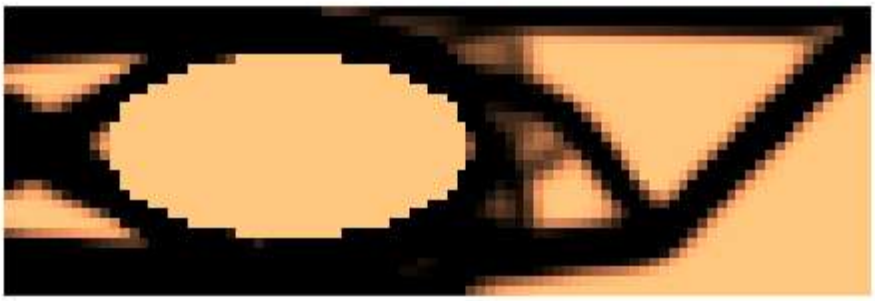


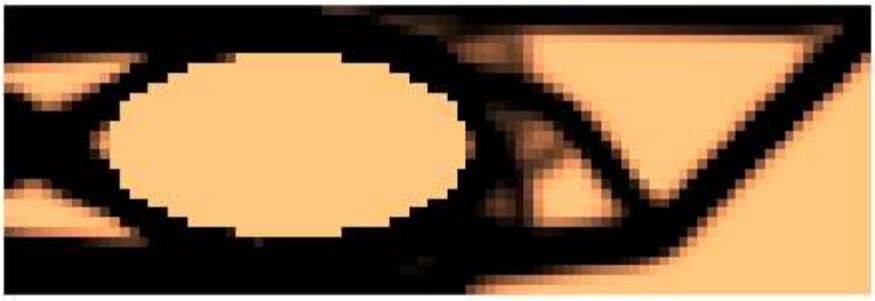


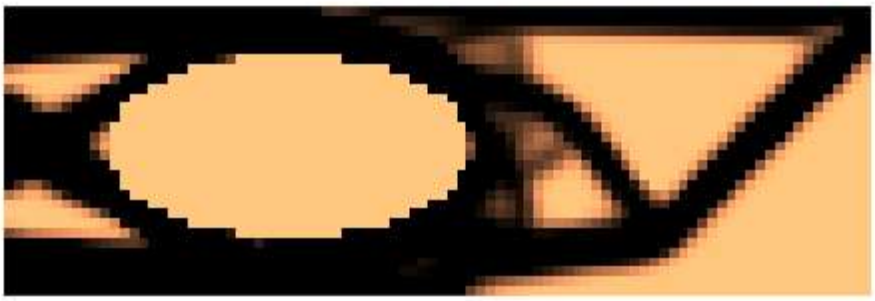


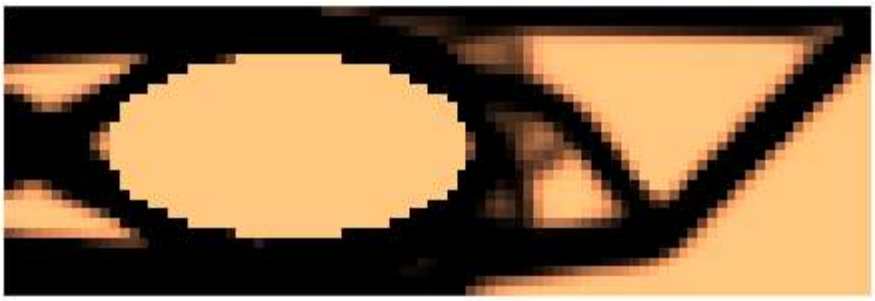


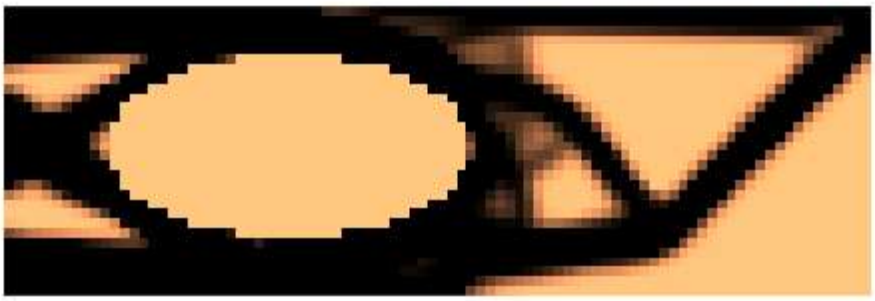


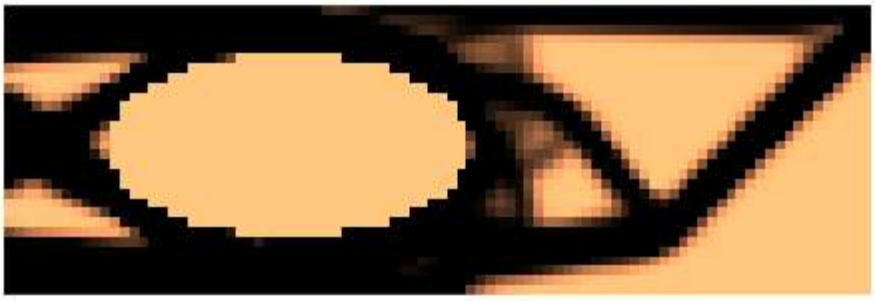


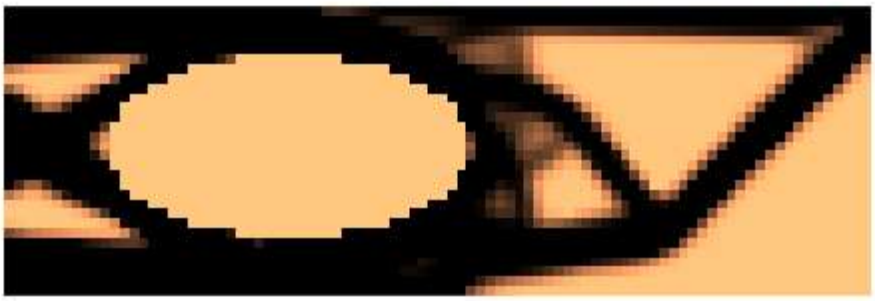


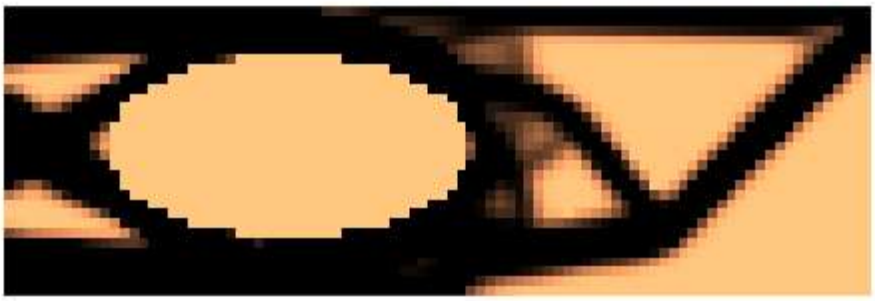


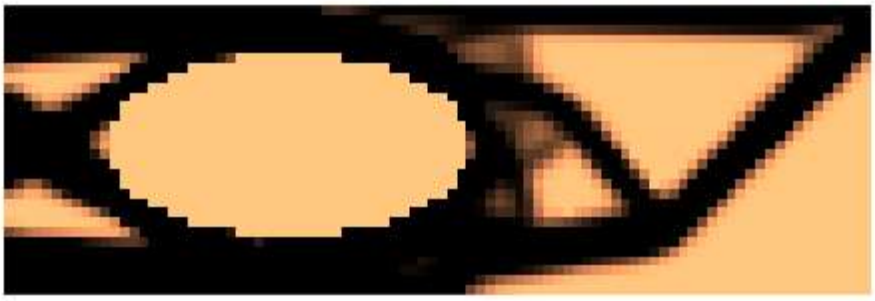


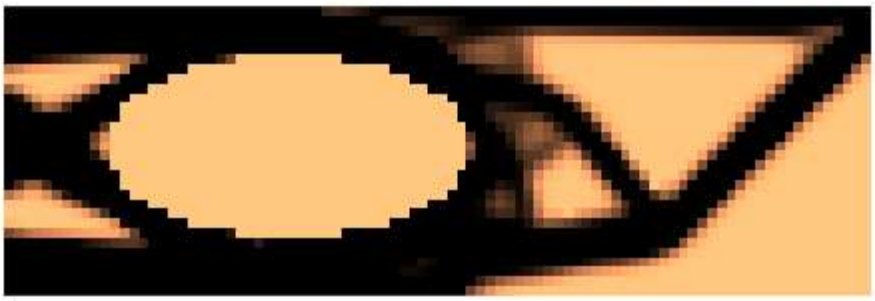


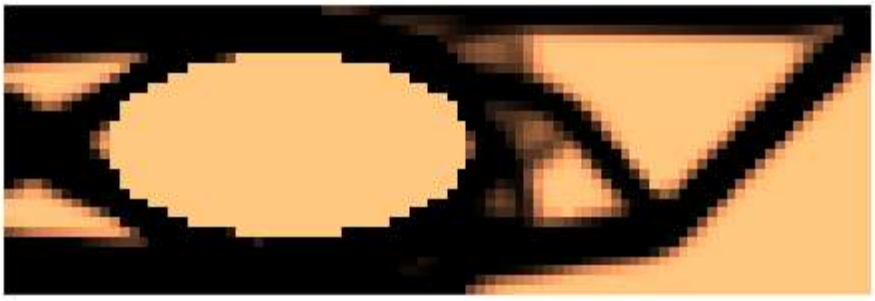


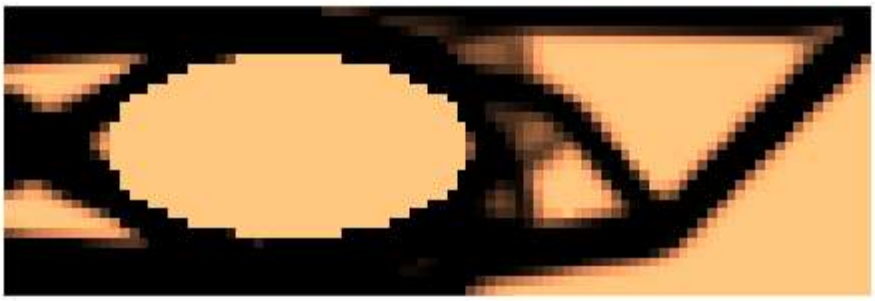


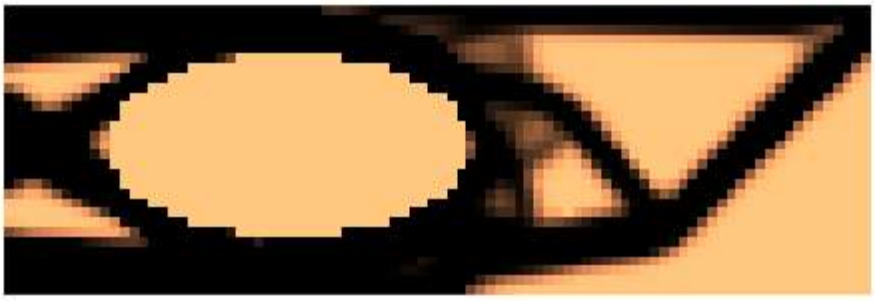


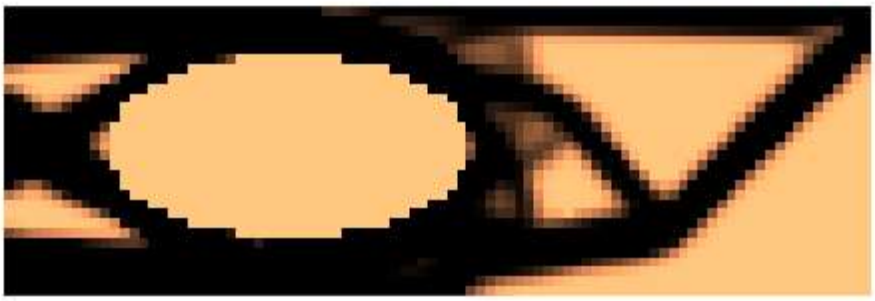


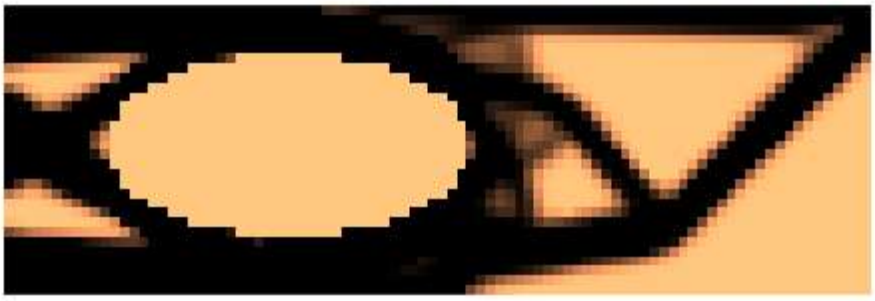


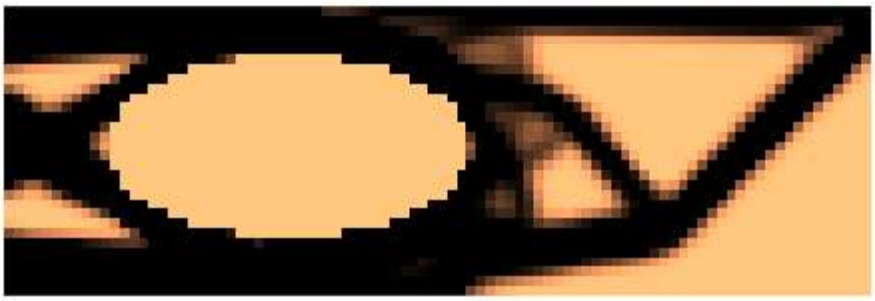


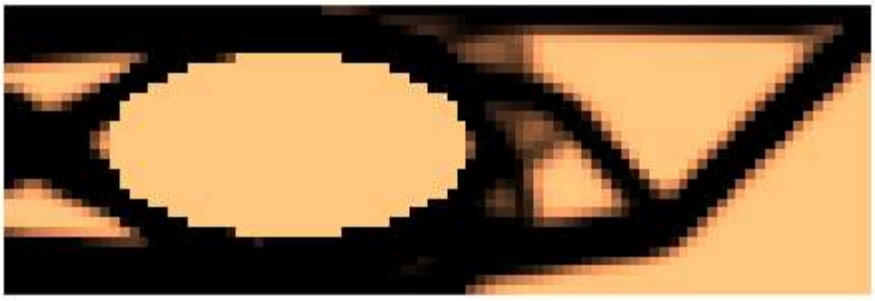


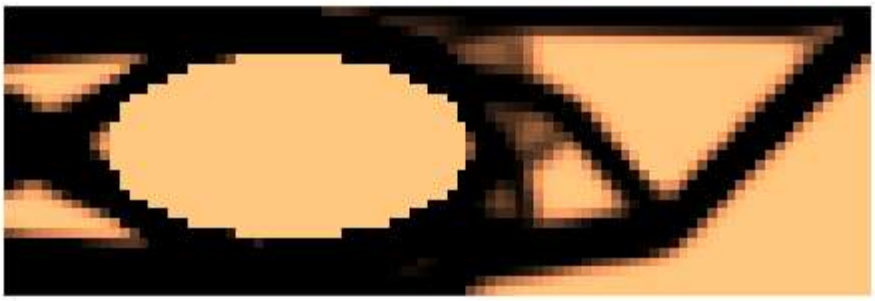


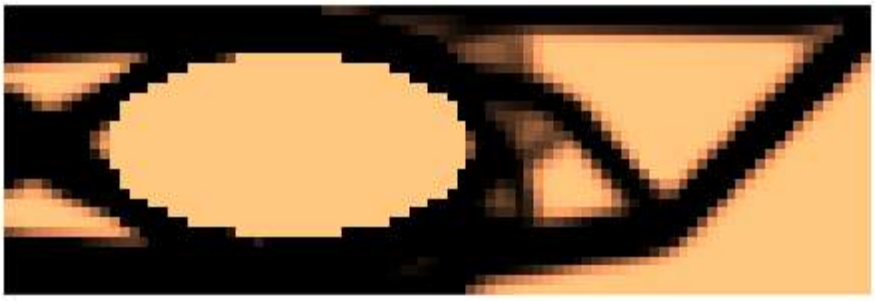


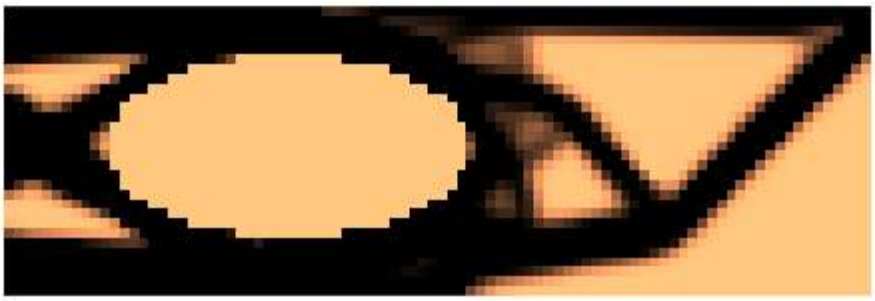


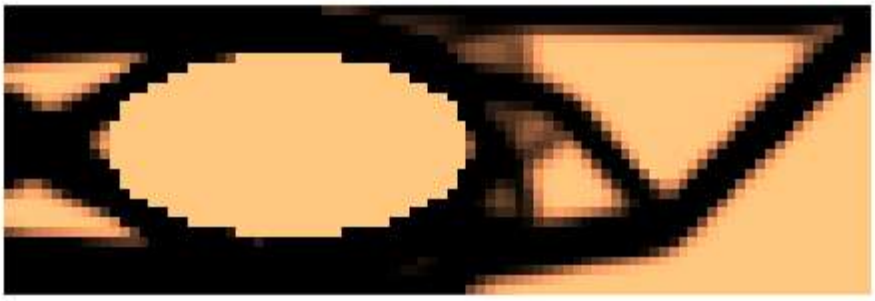


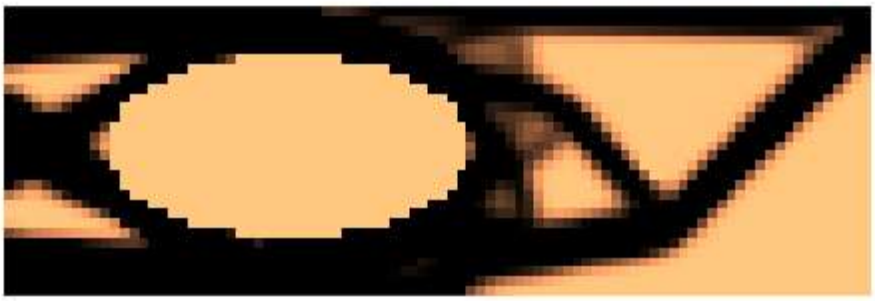


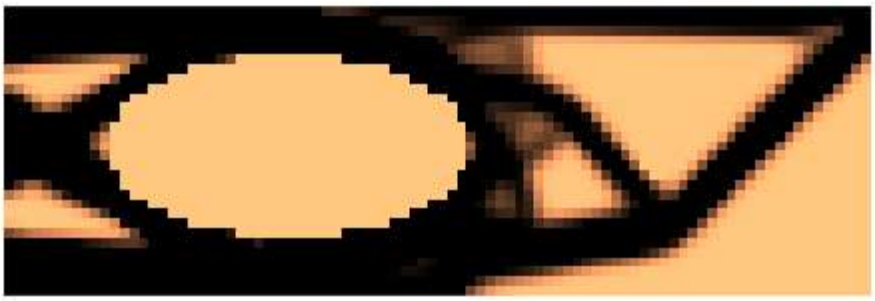


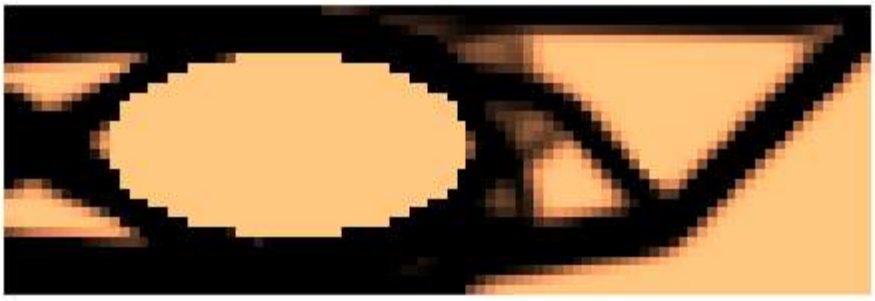


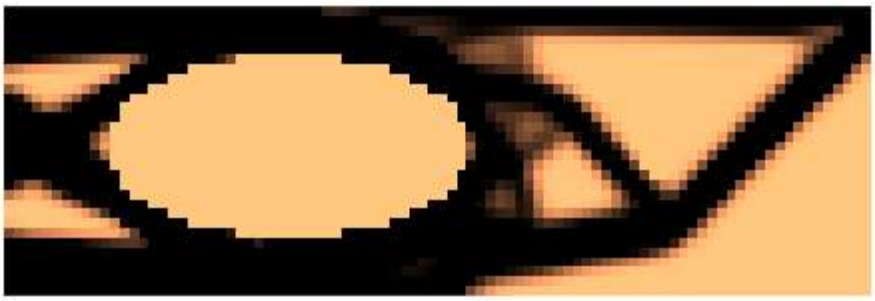












```
end
```

```
end
```

Published with MATLAB® R2022b