Introduction to Scientific Computing
Report Practical : Cellular automata, Mandelbrot set

Denis Garabajiu S4142551
Constantin Cainarean S4142152

Mar 1, 2021

PART 1 Cellular Automata with "majority voting"

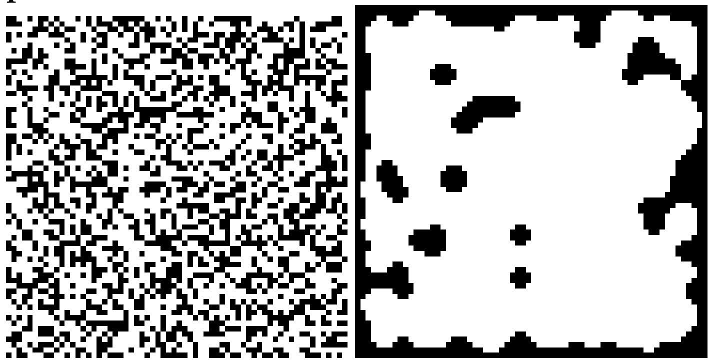
Cellular Automata is is a discrete model of computation. It consists of a regular grid of cells, each in one of a finite number of states. A new generation is created, according to some fixed rule that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. In the first part of the assignment we were told to use the *Majority Voting Rule*

- A live cell dies in the next generation if more than 4 of its immediate neighbours in the present generation are dead, otherwise it stays alive.

- A dead cell comes to life in the next generation if more than 4 of its immediate neighbours in the present generation are alive, otherwise it stays dead.

We run the simulation twice, first one had the initial fraction of living cells = 0.6 (so 60% of the cells were alive) and the second one had the initial fraction of living cells = 0.2. We can notice that in the first case, the dead cells have grouped in patterns in the center, looking like islands in the ocean and formed a perimeter on the outside while in the second case, the majority of the dead cells influenced the living cells and in the end all cells died after a few generations. We can conclude that when the relation living cells / dead cells is more close to 1 then the evolution of the generations will have similar ratio at the end.

p = 0.6



p = 0.2

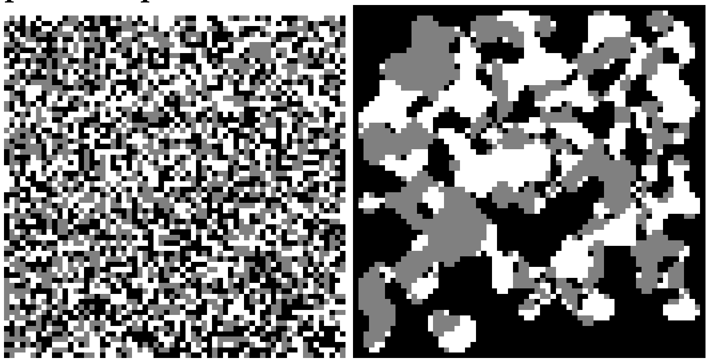
PART 2 Cellular Automata with “majority voting” and three states

In the part 2 of the assignment we had to implement the same *Majority Voting* algorithm but with 3 states and different transition rules.
The rules were

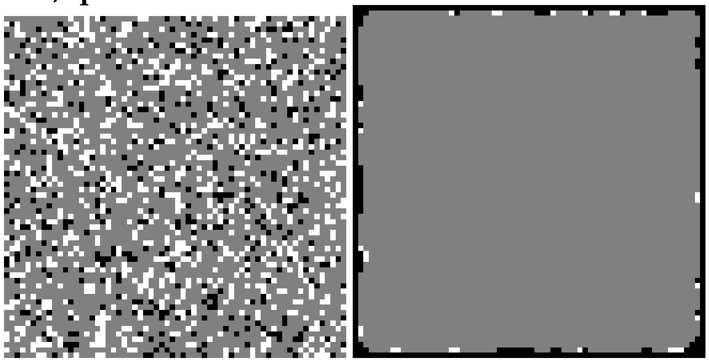
- if one of the states S(living, sleeping, ordead)of the 9 cells in the 3×3 neighbourhood1 of cell (i, j) in generation n has the absolute majority, then cell (i, j) will be in that state S in generation n + 1. (We say that a state S has the absolute majority when at least 5 of the cells are in state S.)
- if none of the states of the 9 cells in the 3×3 neighbourhood of cell(i,j)has the absolute majority in generation n, then the state of cell (i, j) remains the same.

This algorithm it is very similar to the first, but due to the fact that it has 3 states, a little variety in the evolution of the generations appeared. Let’s analyze the results of our tests.

p = 0.33, q = 0.33



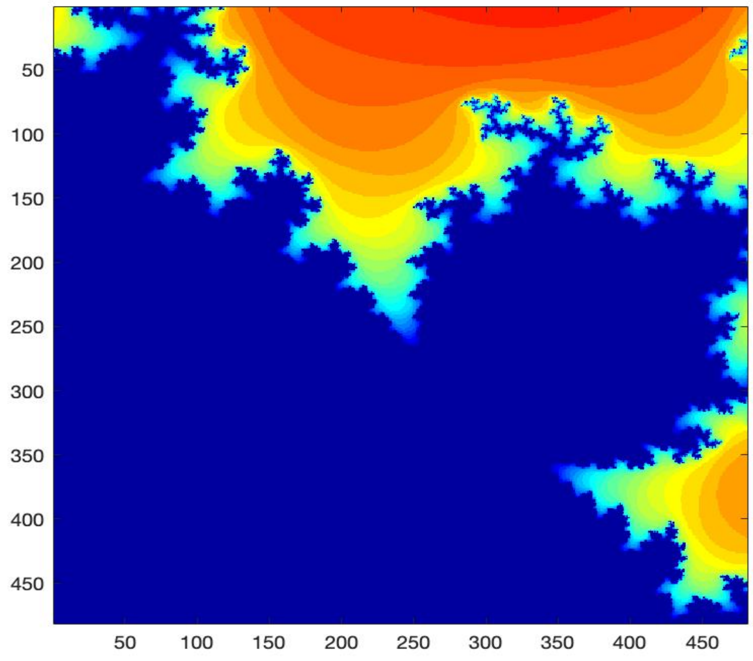
p = 0.2, q = 0.7



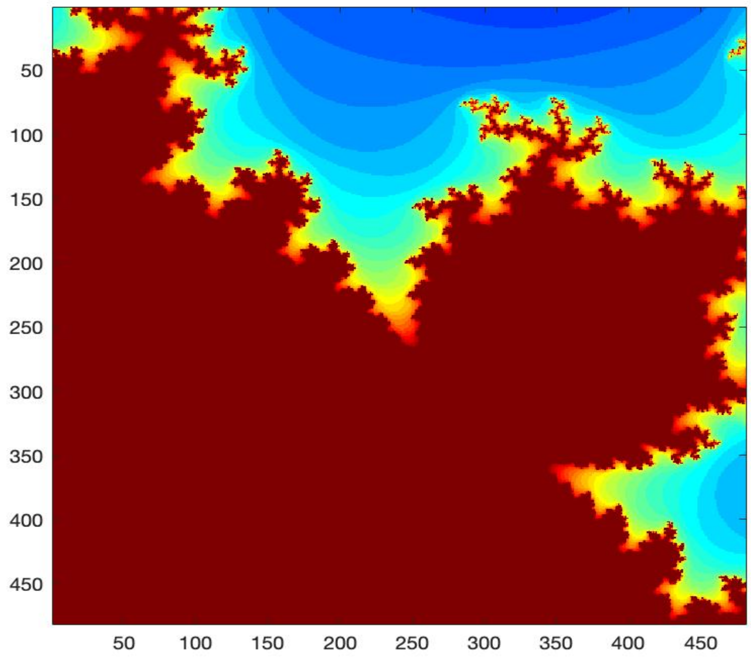
First of all we notice that if a cell has a great majority from the beginning then it will dominate the pattern in the following generations and the evolution will be shorter than the generations with equal spread of states at the start. This happens because the algorithm runs on strict predefined conditions that benefit the state which has the majority at the beginning. Maybe that’s why the algorithm called *Majority Voting*, but who knows.

PART 3 Mandelbrot set

a. We have followed the instructions from the assignment and have created a function *ISCMandelbrot.m* with the parameters **x = .105 : .0005 : .345** and **y = .660 : .0005 : .420, depth = 32** . Using the jet colormap on the image we have received the following results

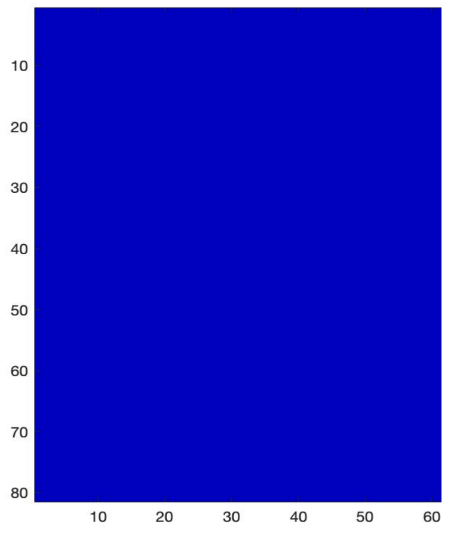


:

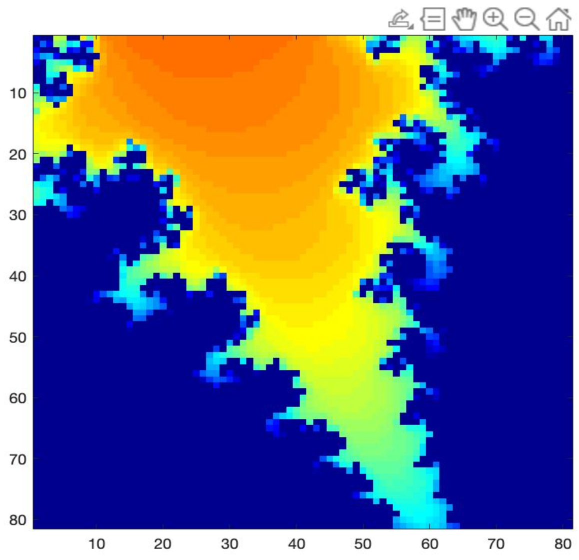


The first figure is the result when the code was executed with the function *flipud*, and the second figure is without flipping the array up - down. The image represents a part of Mandelbrot set visualization.

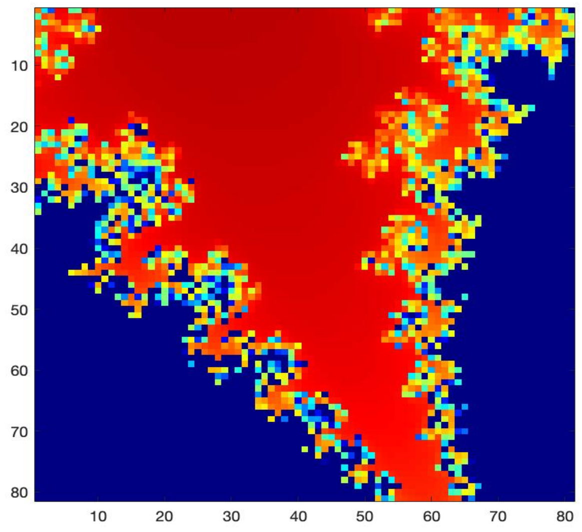
b. Following the instructions in the assignment we have modified the original file "ISC-Mandelbrot.m", more specificaly we have changed the value of x to **x = .205 : .0005 : .245**, and the value of y to **y = .560 : .0005 : .520;**. We have conducted the tests with these parameters changing the depth to 16 then to 64 and then to 256.



(a) Depth = 16



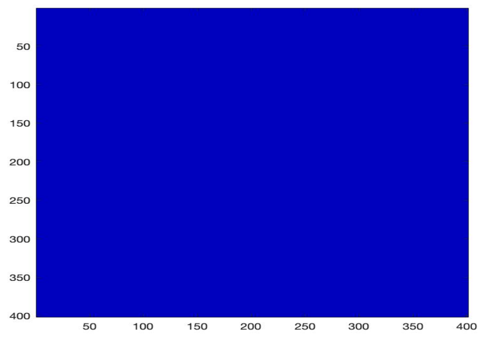
(b) Depth = 64

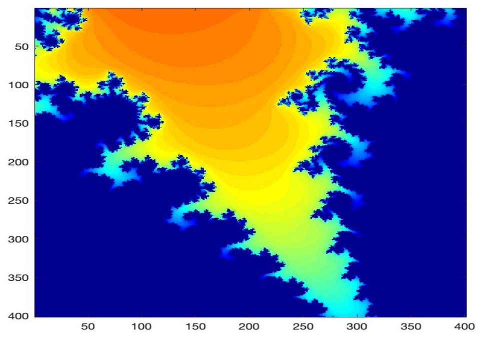


(c) Depth = 256

Figure 1: 7b) The output images with the Step = 0.0005 and different depths

In the second step of the sub-problem, we have checked how the output will differ when we change the step to the value of **0.0001**. By this we have increased the resolution by decreasing the step size. When the step size is **0.0005** the resolution is **80**, but when we decrease the step size to **0.0001** the resolution gets higher to : **400**. This explains the increase of the image quality, when the step size decreases.


(a) Depth = 16


(b) Depth = 64

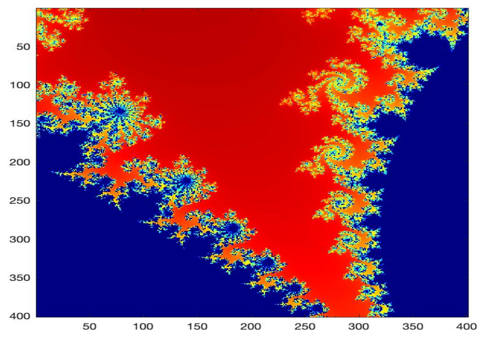

(c) Depth = 256

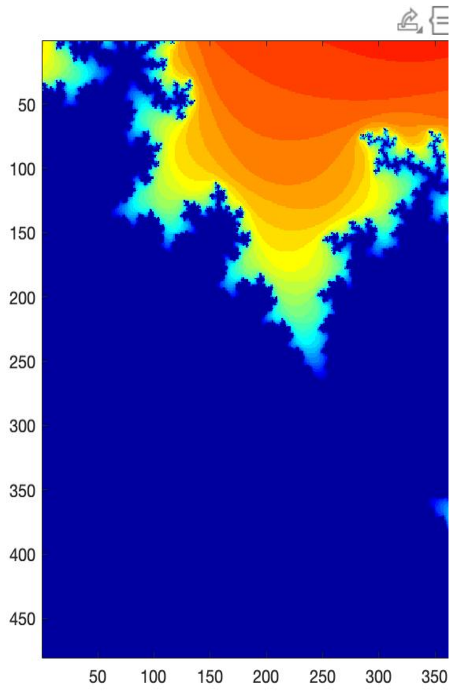
Figure 2: 7b) The output images with the Step = 0.0001 and different depths

Analyzing the output by comparing images from *Figure 1* with the ones from *Figure 2*, the first thing that is easily noticeable is when we decreased the step size, the images were increased in resolution, therefore the quality of the images is higher and the images appear to be clearer, without evident pixelation.

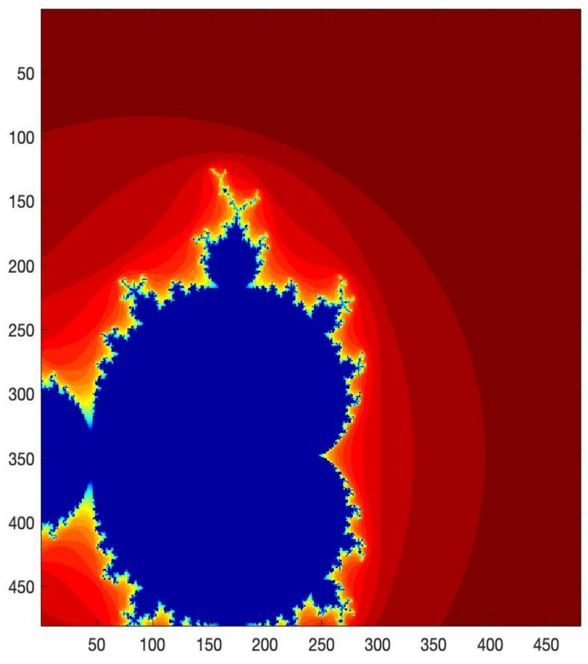
c. Applying the parameters that were described in this part of the assignment to the original code, more specifically generating the x and y vector using the function *linspace* with the arguments x_0 and x_1, also y_0 and y_1 respectively, we tried to reproduce the picture from the subpoint 7A (Figure 3a). This is the reason why we have set (xc, yc) at (.225, .540), halfsize as 0.12, zoom as 1 and res as 480.

   After running the above described code we have received exactly the same image as in the subpoint 7A. Also, we have tested how will the output images change when we adjust the values for *zoom, res, halfsize*. Below we attached the different images we got using different values. When we were increasing/decreasing a variable's value, we kept the other 2 at the default values as were given in the assignment. This means that for example when we produced 3 different images for **zoom =1 then zoom = 0.1 and finally zoom =5** the values for halfsize and res were default, **0.12 and 480 respectively**.

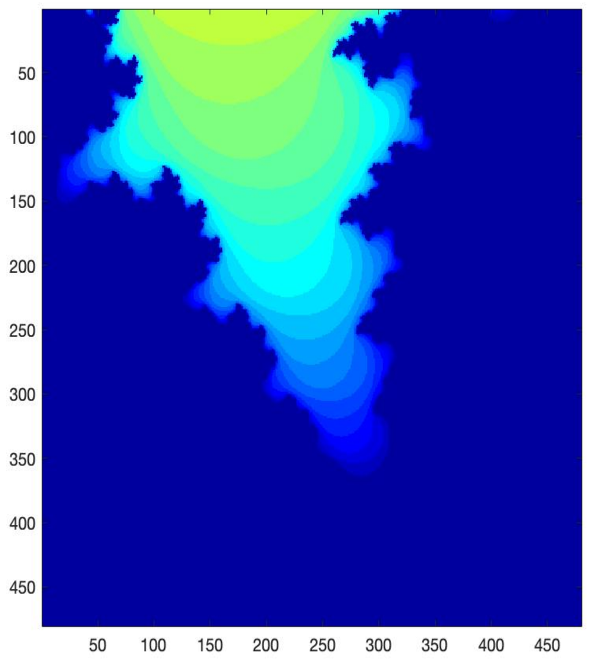
Following our observations on adjusting the values we can conclude that it is pretty intuitive to understand how do this values change the picture. Zoom is basically zooming image in it's centre, when we increase it's value the Mandelbrot set is more zoomed, when we decrease it we can see it in full size. Resolution is responsible for the picture's resolution, when it is low, the quality is bad, image is not clear and we can see pixelation, absolutely the opposite happens when we increase the resolution. Halfsize is actually responsible for something simmilar with image cropping. The higher the value the biggest part of the Mandelbrot set we can observe. To support our conclusions : **Figure 3, Figure 4, Figure 5**.



(a) Zoom = 1 (Identical image as in 7A)

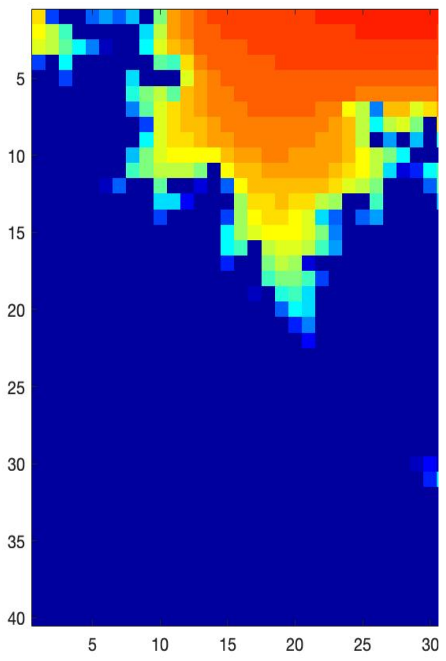


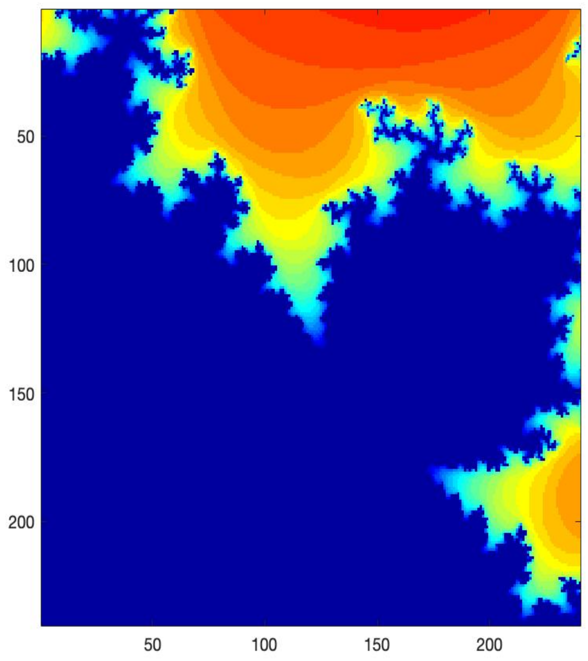
(b) Zoom = 0.1



(c) Zoom = 5

Figure 3: 7c) The output images with different zoom values (res and halfsize on default values


(a) Resolution = 40


(b) Resolution = 240

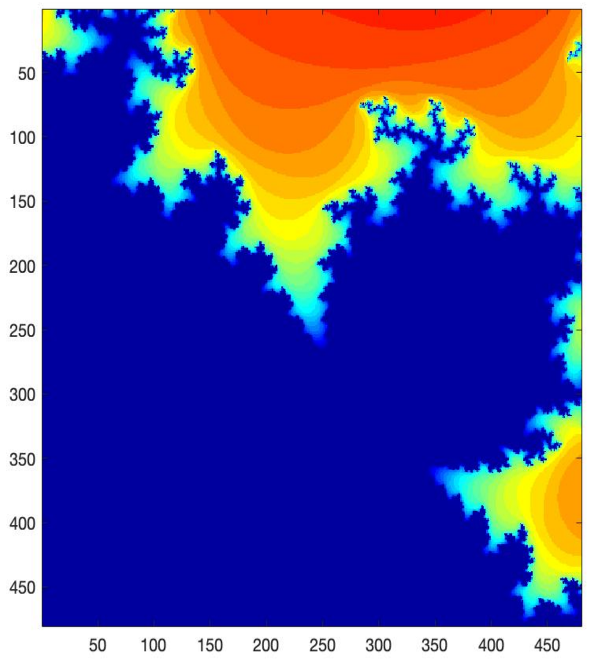
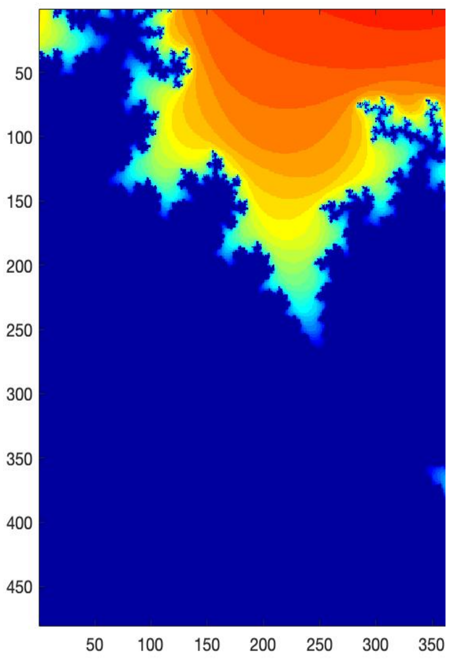
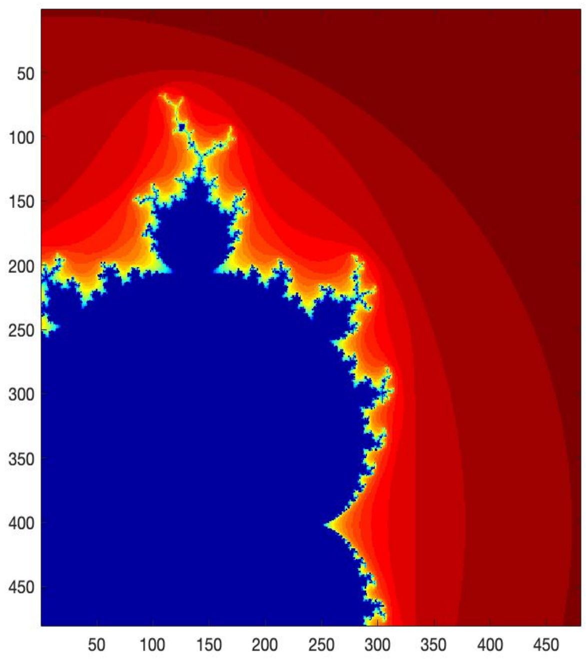

(c) Resolution = 480

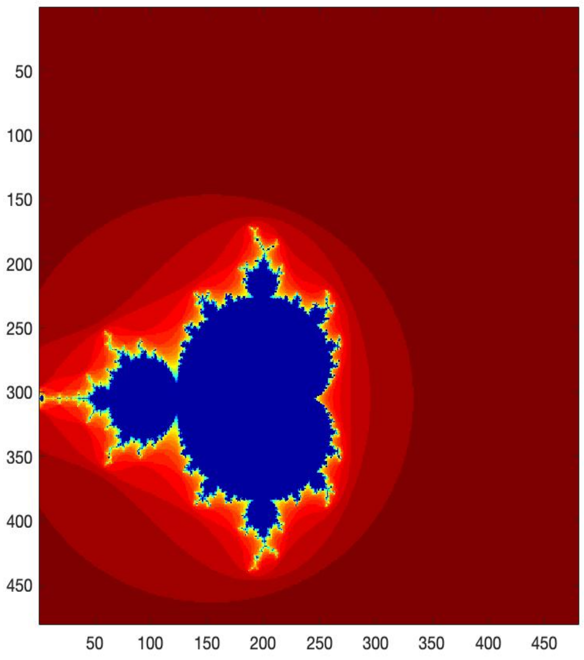
Figure 4: 7c) The output images with different res values (zoom and halfsize on default values



(a) Halfsize = 0.12



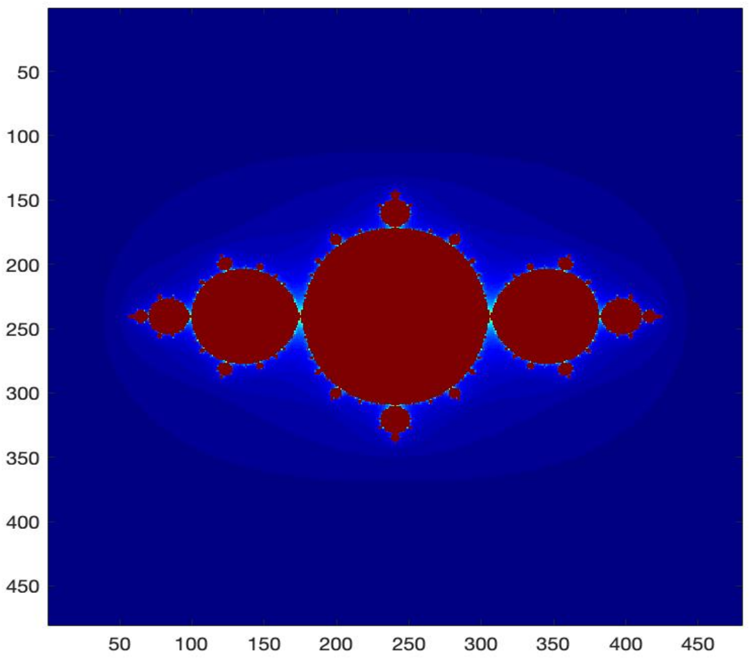
(b) Halfsize = 0.8

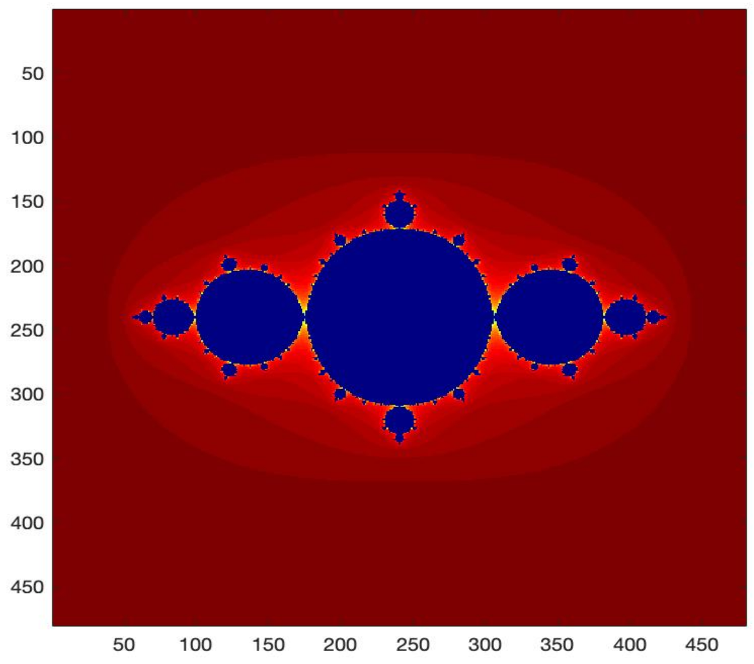


(c) Halfsize = 2

Figure 5: 7c) The output images with different halfsize values (res and zoom on default values

d. For this sub-point of the assignment 7, we followed the instructions described in the text of the exercise. Our goal was to compute the Julia set. It is defined, for a fixed value of c, as the set Jc of complex numbers z0 for which the orbit z0, z1, z2, . . . , zk , . . . with $z_{k+1} = z^{2k}c$ remains bounded. In our example we have used the value of $c = -.85$. Also we have used the region $[-2; 2]$ and the value for maximum iterations $max\_iter = 50$. We have used this exact region, because the example outputs in the Reader were on the same region, so we decided it is a good idea to illustrate how the value **c = -.85** compares to the one's from Reader. We have generated two pictures one using flipud function on the *jet* colormap, and second one is without flipping the array up-down. Below we have attached the output images :





ISCa1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Introduction to Scientific Computing - WBCS14003              %
%                                                               %
%  Simulate spatial pattern formation in Matlab                 %
%  via cellular automata                                        %
%                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;                                  % clear the command window
close all                             % close open figure windows
clear all;                            % remove items from the workspace

n=64;                                 % number of cells horizontally/vertically
p=0.2;                                % probability that a cell is alive
max_gen=100;                          % maximal number of generations

% Initialize matrix A
imname='random';                      % name of the pattern
A=rand(n,n)<p;                        % n x n matrix A with random zeroes/ones
                                      % expected number of living cells is p (for n lar

% Print the initial fraction of living cells on the screen
fprintf('initial fraction of living cells=%f\n',sum(sum(abs(A)))/n^2);

% Display the initial pattern of living and dead cells as an image
% The living cells are white, the dead cells black.
figure;                               % open a figure window
imHandle = imagesc(A,[0 1]);          % display the matrix A as an image.
                                      % The value range of A is [0 1].
colormap(gray);                       % set a gray scale color table

% Write the image to a PNG file
gen=1;                                % current generation number
imfile = [imname,’_n=’,int2str(n),’_p=’,num2str(p),’_gen=’,int2str(gen),’.png’];
imwrite(A, imfile);                   % write A

% Expand matrix A to matrix A1 because of the extra borders needed
A1=zeros(n+2,n+2);                    % initialise (n+2)x(n+2) matrix with zeroes
A1(2:n+1,2:n+1)=A;                    % Insert matrix A in matrix A1

% Now compute the successive generations via the majority rule.
% The algorithm should terminate as soon as no more differences
% occur between successive generations.

% Here comes your code ....
for g = 1 : max_gen
    A_current = A1;
    for i = 2 : (n + 1)
        for j = 2 : (n + 1)
            if (A_current(i,j) == 0)
                x =  getValue(A_current, 1, i, j);
                if (x == 1)
                    A1(i,j) = 1;
                end
            end

            if (A_current(i,j) == 1)
                x = getValue(A_current, 0, i, j);
                if (x == 1)
                    A1(i,j) = 0;
                end
            end
        end
    end
    set(imHandle, ’CData’ , A1);
    drawnow ;
    pause(0.1);
    if (isequal(A1,A_current == true))
        fprintf(’initial fraction of living cells=%f\n’,sum(sum(abs(A1)))/n^2);
        imfile = [imname,’_n=’,int2str(n),’_p=’,num2str(p),’_gen=’,int2str(g),’.png’]
        imwrite(A1, imfile);
        break;
    end
    A_current = A1;
end

function x=getValue(Arr, state, row, col)
    counter = 0;
    for r = (row - 1) : (row + 1)
        for c = (col - 1) : (col + 1)
            if (((r ~= row) || (c ~= col)) && (Arr(r,c) == state))
                counter = counter + 1;
            end
        end
    end
    if (counter > 4)
        x = 1;
    else
        x = 0;
    end

end
```

ISCa2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Introduction to Scientific Computing - WBCS14003                 %
%                                                                  %
%  Simulate spatial pattern formation in Matlab                    %
%  via cellular automata                                           %
%                                                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;                                  % clear the command window
close all                             % close open figure windows
clear all;                            % remove items from the workspace

n=64;                                 % number of cells horizontally/vertically
p=0.2;                                % probability that a cell is alive
q = 0.7;                              % probability that a cell is sleeping
max_gen=100;                          % maximal number of generations

% Initialize matrix A
imname='random3';                      % name of the pattern
R=rand(n,n); % matrix with random values between 0 and 1
A=(R>1-p-q)+(R>1-p); % matrix A with random values 0,1,2

% Print the initial fraction of living cells on the screen
fprintf('initial fraction of dead cells=%f\n',sum(sum(abs(A == 0)))/n^2);
fprintf('initial fraction of sleeping cells=%f\n',sum(sum(abs(A == 1)))/n^2);
fprintf('initial fraction of living cells=%f\n',sum(sum(abs(A == 2)))/n^2);

% Display the initial pattern of living and dead cells as an image
% The living cells are white, the dead cells black.
figure;                               % open a figure window
imHandle = imagesc(A,[0 2]);          % display the matrix A as an image.
colorbar                                      % The value range of A is [0 1].
colormap(gray);                       % set a gray scale color table

% Write the image to a PNG file
gen=1;                                % current generation number
imfile = [imname,'_n=',int2str(n),'_p=',num2str(p),'_q=',num2str(q),'_gen=',int2str(ge
imwrite(rescale(A), imfile);                  % write A

% Expand matrix A to matrix A1 because of the extra borders needed
A1=zeros(n+2,n+2);                  % initialise (n+2)x(n+2) matrix with zeroes
A1(2:n+1,2:n+1)=A;                  % Insert matrix A in matrix A1

% Now compute the successive generations via the majority rule.
% The algorithm should terminate as soon as no more differences
% occur between successive generations.

% Here comes your code ....
pause(1)
for g = 1 : max_gen
    A_current = A1;
    for i = 2 : (n + 1)
        for j = 2 : (n + 1)
            AX = A_current(i-1:i+1,j-1:j+1);
            x = getValue(AX, 0, 2 ,2);
            y = getValue(AX, 1, 2 ,2);
            z = getValue(AX, 2, 2 ,2);
            if (x == 1)
                A1(i,j) = 0;
            elseif (y == 1)
                A1(i,j)= 1;
            elseif (z == 1)
                A1(i,j)= 2;
            end
        end
    end
    set(imHandle, 'CData' , A1);
    drawnow ;
    pause(0.1);
    if (isequal(A1,A_current) == true)
        fprintf(' fraction of dead cells=%f\n',sum(sum(abs((A_current(2:n+1,2:n+1))==
        fprintf(' fraction of sleeping cells=%f\n',sum(sum(abs((A_current(2:n+1,2:n+1)
        fprintf(' fraction of living cells=%f\n',sum(sum(abs((A_current(2:n+1,2:n+1))

        imfile = [imname,'_n=',int2str(n),'_p=',num2str(p),'_q=',num2str(q),'_gen=',in
        imwrite(rescale(A1), imfile);
        break;
    end
    A_current = A1;
end

function x=getValue(Arr, state, row, col)
    counter = 0;
    for r = (row - 1) : (row + 1)
        for c = (col - 1) : (col + 1)
            if (((r ~= row) || (c ~= col)) && (Arr(r,c) == state))
                counter = counter + 1;
            end
        end
    end
    if (counter > 4)
        x = 1;
    else
        x = 0;
    end

end
```

ISCmandelbrot.m

```
% The give regions and step size from the assignment
x = .105 : .0005 : .345;
y = -0.660 : .0005 : -.420;

n = length(x);
e = ones(n,1);
z0 = x(e,:) + i*y(:,e);

[X,Y] = meshgrid(x,y);
z0 = X + i*Y;

z = zeros(n,n);
c = zeros(n,n);

% The value of depth was given in the assignment
depth = 32;
   for k = 1:depth
      z = z.^2 + z0;
      c(abs(z) < 2) = k;
   end

c
image(c)
axis image

colormap(flipud(jet(depth)))
```

ISCmandelbrot2.m

```
% The given values from the assignment to get the image from 7A
xc = .225;
yc = -.540;
halfsize =  0.12;
res = 480;
zoom = 1;

% Calculating the region
x0 = xc - halfsize / zoom;
x1 = xc + halfsize / zoom;
x = linspace(x0,x1,res);

y0 = yc - halfsize / zoom;
y1 = yc + halfsize / zoom;
```

```
y = linspace(y0,y1,res);

[X,Y] = meshgrid(x,y);
z0 = X + i*Y;

n = length(x);
z = zeros(n,n);
c = zeros(n,n);

% Same depth as in 7A to be able to get the same image.
depth = 32;
   for k = 1:depth
      z = z.^2 + z0;
      c(abs(z) < 2) = k;
   end

c
image(c)
axis image

colormap(flipud(jet(depth)))
```

ISCmandelbrot7b.m

```
% The step size was subject to changes from 0.0005 to 0.0001 during
% different tests, this is the last value from the last test.
x = .205 : .0001 : .245;
y = -.560 : .0001 : -.520;

[X,Y] = meshgrid(x,y);
z0 = X + i*Y;

n = length(x);
z = zeros(n,n);
c = zeros(n,n);
%The depth size was subject to changes (16, 64 and 256) during different
%tests, this is the last value from last test.

depth = 256;
   for k = 1:depth
      z = z.^2 + z0;
      c(abs(z) < 2) = k;
   end

c
image(c)
axis image

colormap(flipud(jet(depth)))
```

ISCjulia.m

```
% Computing the filled Julia set, for c = -.85 . Region of the z0-plane: [2, 2] × [2,
res = 480;
x0 = 2;
x1 = -2;
x = linspace(x0,x1,res);

y0 = 2;
y1 = -2;
y = linspace(y0,y1,res);

[X,Y] = meshgrid(x,y);
z = X + i*Y;

c = -.85;
B = zeros(size(c));

% Number of maximum iterations is 50
max_iter = 50;
   for k = 1:max_iter;
      z = z.^2 + c;
      B = B + (abs(z) < 2);
   end

imagesc(B)
colormap(flipud(jet))
axis image
```

Work Distribution : As usually, we have distributed the work equally, speaking percent-wise it's 50% / 50%. We have collaborated on all three parts of the Practical.