Contents

- Published Test Case
- Published Test Case
- Additional Test Case

E7 Lab 8 Solutions

Spring 2016

```
format compact
format short
clear all
clc
close all
```

Question 1.1

```
type myLinearSolver
```

```
function [x1, x2, x3] = myLinearSolver(a1, b1, a2, b2, a3, b3)
A=[a1;a2;a3];
b=[b1;b2;b3];
X=A\b;
x1=X(1);
x2=X(2);
x3=X(3);
end

%%Publish a test case for this one
```

Published Test Case

```
[x1,x2,x3] = myLinearSolver([1 0 -1],1,[0 -1 1],2,[2 1 1],0)
```

```
x1 =
     1
x2 =
    -2
x3 =
     0
```

Additional Test Case

```
[x1,x2,x3] = myLinearSolver([1 1 2],1,[0 1 2],1,[2 2 3],2)
```

```
x1 =
     0
```

```
x2 =
     1
x3 =
     0
```

Question 1.2

```
type checkSingular
```

```
function [flag] = checkSingular(A)
[s1,s2]=size(A);
flag = (s1~=s2) || abs(det(A)) < 1e-12;
end
```

Published Test Case

```
Output = checkSingular([1 2;3 4])
```

```
Output =
     0
```

Published Test Case

```
Output = checkSingular([1 2; 3 4; 5 6])
```

```
Output =
     1
```

Published Test Case

```
Output = checkSingular([1 0 1; 2 0 2; 1 2 0])
```

```
Output =
     1
```

Additional Test Case

```
Output = checkSingular(1)
```

```
Output =
     0
```

Additional Test Case

```
Output = checkSingular(0)
```

```
Output =
     1
```

Additional Test Case

```
Output = checkSingular([1 2 3])
```

```
Output =
     1
```

Question 1.3

```
type matInv
```

```
function [B] = matInv(A)

[s1,s2]=size(A);

if (s1~=s2) || (abs(det(A)) < 1e-10)
    B=[];
else
    B=inv(A);

end
```

Published Test Case

```
Output = matInv([1 2; 3 4])
```

```
Output =
   -2.0000    1.0000
    1.5000   -0.5000
```

Published Test Case

```
Output = matInv([1 2; 3 4; 5 6])
```

```
Output =
     []
```

Published Test Case

```
Output = matInv([1 0 1; 2 0 2; 1 2 0])
```

```
Output =
     []
```

Additional Test Case

```
Output = matInv([0 1;-1 0])
```

```
Output =
     0    -1
     1     0
```

Question 2

```
type MyCircuit
```

```
function [I1,I2,I3,I5,I6,I7] = MyCircuit(V, R1, R2, R3, R4, R5, R6, R7, R8)

A = [1 -1 -1  0  0  0   0  0; ...
     0  1  1 -1  0  0   0  0; ...
     0  0  0  1 -1 -1  -1  0; ...
     0  0  0  0  1  1   1  -1; ...
     0 R2 -R3 0  0  0   0  0;...
     0  0  0  0 R5 -R6  0  0;...
     0  0  0  0  0  R6 -R7 0;...
     R1 R2 0 R4  R5 0   0  R8]

 b = [0; 0; 0; 0; 0; 0; 0; V]

currents = A \ b;

I1 = currents(1);
I2 = currents(2);
I3 = currents(3);
I4 = currents(4);
I5 = currents(5);
I6 = currents(6);
I7 = currents(7);
I8 = currents(8);

end
```

Pulished Test Case

```
 [I1,I2,I3,I5,I6,I7] = MyCircuit(10,1,4,4,2,2,1,4,5)
```

```
A =
     1    -1    -1     0     0     0     0     0
     0     1     1    -1     0     0     0     0
     0     0     0     1    -1    -1    -1     0
     0     0     0     0     1     1     1    -1
     0     4    -4     0     0     0     0     0
     0     0     0     0     2    -1     0     0
     0     0     0     0     0     1    -4     0
     1     4     0     2     2     0     0     5
b =
     0
     0
     0
     0
     0
     0
     0
    10
I1 =
    0.9459
I2 =
    0.4730
I3 =
    0.4730
I5 =
    0.2703
I6 =
    0.5405
I7 =
    0.1351
```

Question 3.1

```
type myNewton
```

```
function [R, E] = myNewton(f, df, x0, tol)

% if abs(f(x0))<tol
%     R=x0;
%     E=abs(f(x0));
% else
%     x1=x0-f(x0)/df(x0);
%     [r,e]=myNewton(f,df,x1,tol);
%     R=[x0,r];
%     E=[abs(f(x0)),e];
% end

R = x0;
E = abs(f(x0));

for i = 1:100
    if (E(end) < tol)
```

```
        break;
    else
        x0 = x0 - (f(x0)/df(x0));
        R = [R x0];
        E = [E abs(f(x0))];
    end
end

end
```

Published Test Case

```
f = @(x) x^2-2;
df = @(x) 2*x;
[R, E] = myNewton(f, df, 1, 1e-5)
```

```
R =
    1.0000    1.5000    1.4167    1.4142
E =
    1.0000    0.2500    0.0069    0.0000
```

Published Test Case

```
f=@(x) sin(x)-cos(x);
df=@(x) cos(x)+sin(x);
[R, E] = myNewton(f, df, 1, 1e-5)
```

```
R =
    1.0000    0.7820    0.7854
E =
    0.3012    0.0047    0.0000
```

Additional Test Case

```
f = @(x) sinh(x)-1;
df = @(x) cosh(x);
[R, E] = myNewton(f, df, -1, 1e-8)
```

```
R =
   -1.0000    0.4096    0.9431    0.8827    0.8814    0.8814
E =
    2.1752    0.5788    0.0892    0.0019    0.0000    0.0000
```

Question 3.2

```
type myBisection
```

```
function [R, E] = myBisection(f, a, b, tol)

% if abs(f((a+b)/2))<tol
%     R=(a+b)/2;
%     E=abs(f((a+b)/2));
% elseif sign(f((a+b)/2))==sign(f(a))
%     a=(a+b)/2;
%     [r,e]=myBisection(f,a,b,tol);
%     R=[a,r];
%     E=[abs(f(a)),e];
% elseif sign(f((a+b)/2))==sign(f(b))
%     b=(a+b)/2;
%     [r,e]=myBisection(f,a,b,tol);
%     R=[b,r];
%     E=[abs(f(b)),e];
%
% end

R = (a + b)/2;
E = abs(f((a + b)/2));

for i = 1:100

    if E(end) < tol
        break;
    elseif sign(f((a + b)/2)) == sign(f(a))
        a = (a + b)/2;
        R = [R (a + b)/2];
        E = [E abs(f((a + b)/2))];
    elseif sign(f((a + b)/2)) == sign(f(b))
        b = (a + b)/2;
        R = [R (a + b)/2];
        E = [E abs(f((a + b)/2))];
    end

end

end
```

Published Test Case

```
f = @(x) x.^2-2;
[R, E] = myBisection(f, 0, 2, 1e-1)
```

```
R =
    1.0000    1.5000    1.2500    1.3750    1.4375
E =
    1.0000    0.2500    0.4375    0.1094    0.0664
```

Published Test Case

```
f=@(x) sin(x)-cos(x);
```

```
[R, E] = myBisection(f, 0, 2, 1e-2)
```

```
R =
    1.0000    0.5000    0.7500    0.8750    0.8125    0.7813
E =
    0.3012    0.3982    0.0501    0.1265    0.0383    0.0059
```

Additional Test Case

```
f=@(x) exp(-x)-x^2;
[R, E] = myBisection(f, 0, 1, 1e-2)
```

```
R =
    0.5000    0.7500    0.6250    0.6875    0.7188    0.7031
E =
    0.3565    0.0901    0.1446    0.0302    0.0292    0.0007
```

Question 4.1

```
type PolynomialExactSolutions
```

```
function [ solution ] = PolynomialExactSolutions(d)
%PolynomialExactSolution solves the polar form of a complex
%polynomial equation Z^d = 1, providing the respective number of roots
%for the giving value of d.
%   If d = 3, the solution will have three roots.

roots = [];                        %initiates root array
n = 0:1:d-1;                       %accounts for possible number of roots
for t = 1:numel(n)
    phi = (2.*pi.*n(t))./d;           %solves for the argument of the complex polynomial
    real = cos(phi);               %calculates teh real portion
    img = sin(phi).*1i;             %calculates the imaginary portion
    complex = real + img;          %Adds real and imaginary to get full complex root
    roots = [roots, complex];      %appends to root array
end
solution = roots;

end
```

Published Test Case

```
solutions = PolynomialExactSolutions(3)
```

```
solutions =
   1.0000 + 0.0000i  -0.5000 + 0.8660i  -0.5000 - 0.8660i
```

Published Test Case

```
solutions=PolynomialExactSolutions(9)
```

```
solutions =
  Columns 1 through 4
   1.0000 + 0.0000i   0.7660 + 0.6428i   0.1736 + 0.9848i  -0.5000 + 0.8660i
  Columns 5 through 8
  -0.9397 + 0.3420i  -0.9397 - 0.3420i  -0.5000 - 0.8660i   0.1736 - 0.9848i
  Column 9
   0.7660 - 0.6428i
```

Additional Test Case

```
solutions=PolynomialExactSolutions(4)
```

```
solutions =
   1.0000 + 0.0000i   0.0000 + 1.0000i  -1.0000 + 0.0000i  -0.0000 - 1.0000i
```

Question 4.2

```
type NewtonConv
```

```
function [IsConv, ConvTo] = NewtonConv(d, z, n, tol)
  solutions=PolynomialExactSolutions(d);
        for k=1:n;
                z=z - ((z.^d)-1)./(d*(z.^(d-1)));
    end
    abs_diff=abs(z-solutions);
    closest_ind=find(abs_diff==min(abs_diff) & abs_diff<=tol);
    if length(closest_ind)>0
        IsConv=1;
        ConvTo=solutions(closest_ind);
    else
        IsConv=0;
        ConvTo=0.0/0.0;
    end
end
```

Published Test Case

```
[ConvergeOrNot,ConvergeToRoot]=NewtonConv(3,1+i,5,0.001)
```

```
ConvergeOrNot =
     0
ConvergeToRoot =
   NaN
```

Published Test Case

```
[ConvergeOrNot,ConvergeToRoot]=NewtonConv(3,1+i,10,0.001)
```

```
ConvergeOrNot =
     1
ConvergeToRoot =
     1
```

Published Test Case

```
[ConvergeOrNot,ConvergeToRoot]=NewtonConv(3,10+10*i,10,0.001)
```

```
ConvergeOrNot =
     0
ConvergeToRoot =
   NaN
```

Published Test Case

```
[ConvergeOrNot,ConvergeToRoot]=NewtonConv(3,-10+10*i,100,0.001)
```

```
ConvergeOrNot =
     1
ConvergeToRoot =
  -0.5000 + 0.8660i
```

Additional Test Case

```
[ConvergeOrNot,ConvergeToRoot]=NewtonConv(3,2,100,1e-10)
```

```
ConvergeOrNot =
     1
ConvergeToRoot =
     1
```

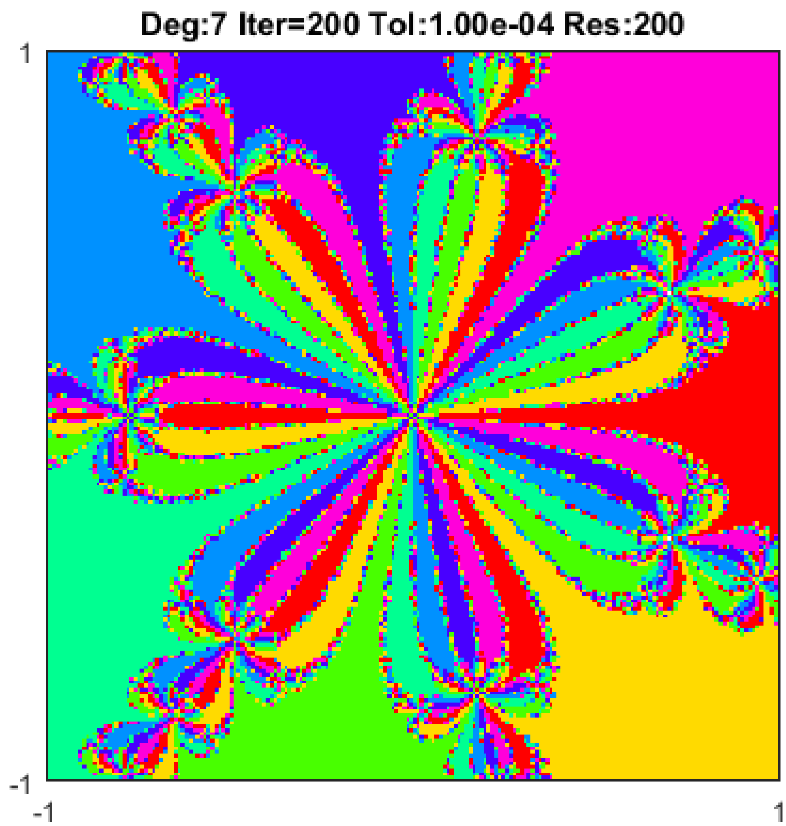
Question 4.3

```
type NewtonFractal
```

```
function [output] = NewtonFractal(d,n,tol,res,ULcorner,sqrL)

x = linspace(ULcorner(1), ULcorner(1) + sqrL, res);      % Real space
y = linspace(ULcorner(2) - sqrL, ULcorner(2), res);      % Imaginary space
[X, Y] = meshgrid(x, y);

Z = X + (1i*Y);

% Perform Newton iterations
for k = 1:n;
    Z = Z - ((Z.^d)-1)./(d*(Z.^(d-1)));
end

% Plotting the fractal
output = zeros(size(Z));

% Find convergence for each point in Z
solutions=PolynomialExactSolutions(d);

for j = 1:d;
    root = solutions(j);                 % The jth root of the d roots of unity
    Mj = abs(Z - root);                          % Where did Z converge close to root
    mask = (Mj <= tol)*j;                        % Each point gets a unique number in [1,d] or
 remains 0
    output = output + mask;      % Add it to the rendering matrix
end

% plotting
if ~isempty(find(output == 0, 1)) % if there is 0 in output
    MyCustomColor = [1 1 1; hsv(length(unique(output)) - 1)];
else
    MyCustomColor = hsv(numel(unique(output)));
end

colormap(MyCustomColor)
imagesc(output)
shading flat;            % Turn off grid lines
axis('equal','square');
axis([1 res 1 res]);
NumTicks = 2;
L = get(gca,'XLim');
set(gca,'XTick',linspace(L(1),L(2),NumTicks))
set(gca,'YTick',linspace(L(1),L(2),NumTicks))
ax = gca;
ax.XTickLabel = {num2str(min(x)),num2str(max(x))};
ax.YTickLabel = {num2str(max(y)),num2str(min(y))};


titleStr=sprintf('Deg:%i Iter=%i Tol:%3.2e Res:%i', d,n,tol,res);
title(titleStr)
end
```

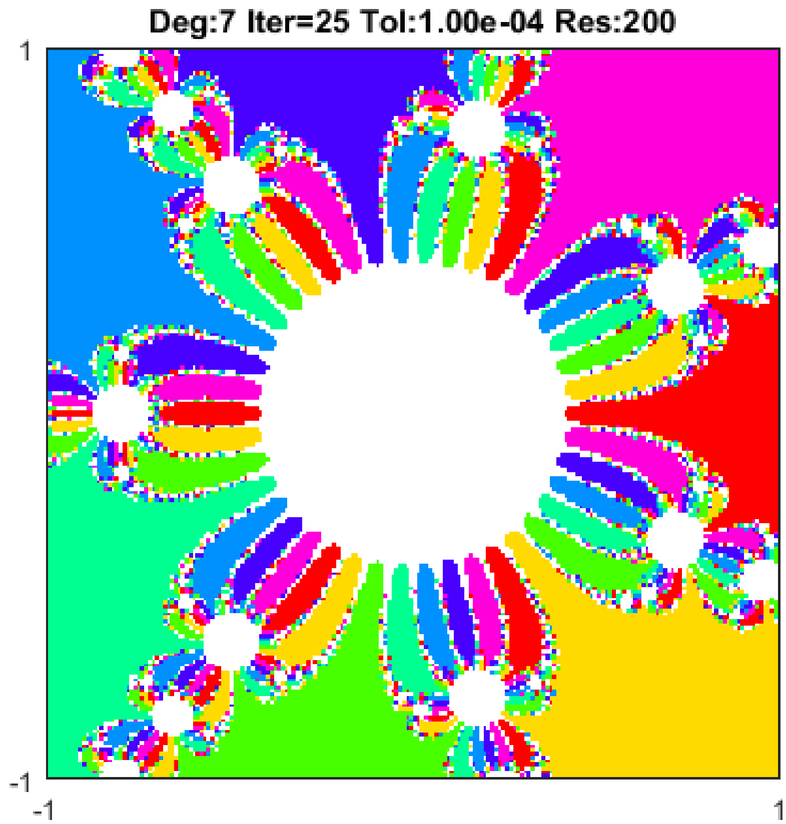
Published Test Case

```
d=7;n=200;tol=1e-4;
res=200;ULcorner=[-1 1];sqrL=2;
output=NewtonFractal(d,n,tol,res,ULcorner,sqrL);
```



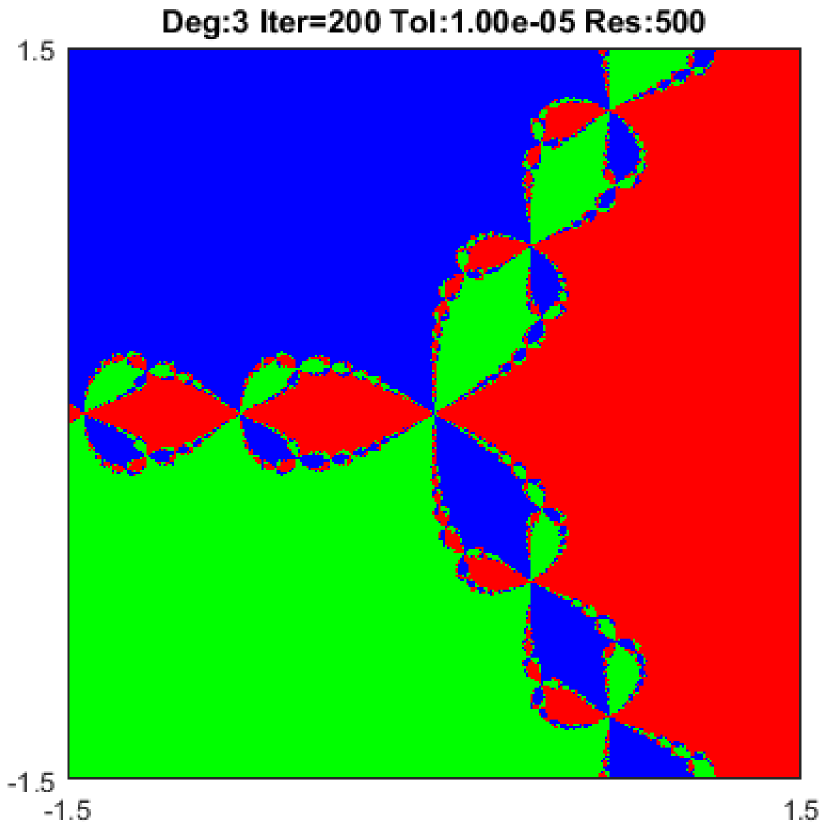
Published Test Case

```
n=25;
output=NewtonFractal(d,n,tol,res,ULcorner,sqrL);
```



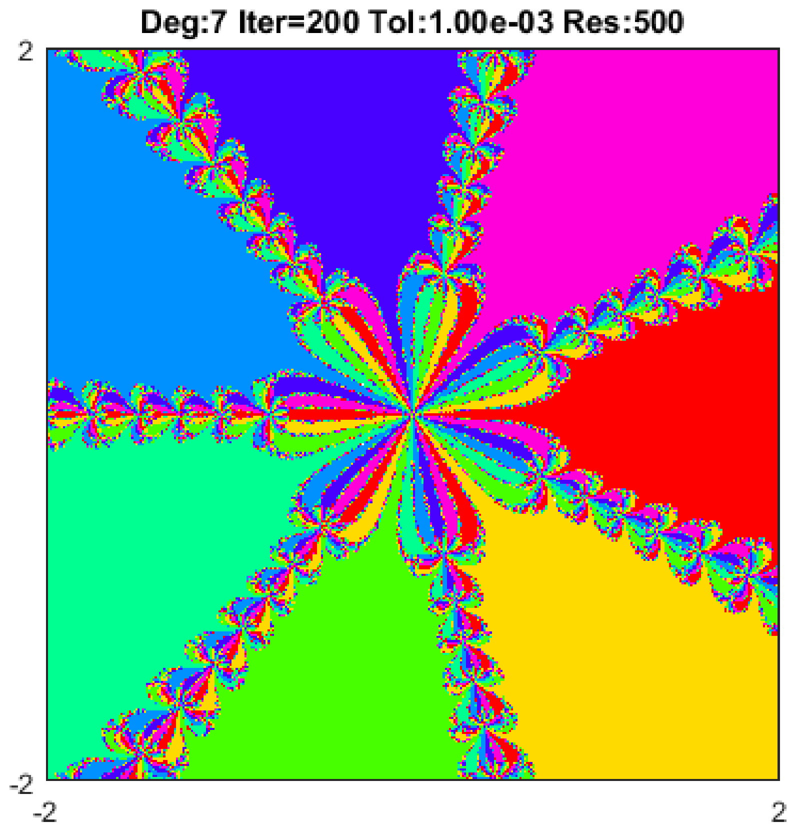
Published Test Case

```
d=3;n=200;tol=1e-5;
res=500;ULcorner=[-1.5 1.5];sqrL=3;
output=NewtonFractal(d,n,tol,res,ULcorner,sqrL);
```



Additional Test Case

```
d=7;n=200;tol=1e-3;
res=500;ULcorner=[-2 2];sqrL=4;
output=NewtonFractal(d,n,tol,res,ULcorner,sqrL);
```



Published with MATLAB® R2015b