Project 2
FMN011

Robert Foss (dt08rf1@student.lth.se)

Contents

1 Introduction and problem background **4**

2 Numerical Considerations **4**

3 Results **4**
- 3.1 Task 1 . . . 4
  - 3.1.1 Computational cost as per definition . . . 4
  - 3.1.2 Computational cost of de Casteljau's algorithm . . . 4
- 3.2 Task 2 . . . 4
- 3.3 Task 3 . . . 5
- 3.4 Task 4 . . . 5
- 3.5 Task 5 . . . 5
- 3.6 Task 6 . . . 5
- 3.7 Task 7 . . . 5
- 3.8 Task 8 . . . 5
- 3.9 Task 9 . . . 5
- 3.10 Task 10 . . . 5

4 Analysis **6**

5 Lessons learned **6**

6 Acknowledgements **6**

7 Appendix: A **7**
- 7.1 de Casteljau . . . 7
- 7.2 Wrap . . . 7
- 7.3 Split lower . . . 7
- 7.4 Split upper . . . 8
- 7.5 Box . . . 8
- 7.6 Intersection . . . 8

8 Appendix: B **11**
- 8.1 Task 2 . . . 11
- 8.2 Task 3a . . . 11
- 8.3 Task 3b . . . 12
- 8.4 Task 3c . . . 12
- 8.5 Task 3d . . . 13
- 8.6 Task 3e . . . 13
- 8.7 Task 4a . . . 14
- 8.8 Task 4b . . . 14
- 8.9 Task 5 . . . 15
- 8.10 Task 6 . . . 15
- 8.11 Task 7 . . . 16
- 8.12 Task 8 . . . 16
- 8.13 Task 9a . . . 17
- 8.14 Task 9b . . . 17
- 8.15 Task 10a . . . 18
- 8.16 Task 10b . . . 18

9 Appendix: C **19**
9.1 Program listing 19

1 Introduction and problem background

The issue of drawing Bézier curves involves iterative calculation of points that lie on a curve. Due to the iteration a computational approach is the easiest way to visually represent a Bézier curve.

2 Numerical Considerations

Points in Bézier curves were with found with de Casteljau's algorithm. The algorithm was implemented in Matlab and can be found in appendix A, section 7.1. Specific Matlab functions used include flipud, rectint, cell matrices, saveas, axis and set.

3 Results

3.1 Task 1

3.1.1 Computational cost as per definition

$P(t) = \sum_{k=0}^{m} P_k B_k^m(t)$ $\qquad\qquad$ $B_k^m = \frac{m!}{k!(m-k)!} t^k (1-t)^{m-k}$

The number of operations for each factorial is $n-1$ given $n!$. The same applies to exponential expression such as n^k which requires $k-1$ operations. The calculation of a Bernstein polynomial of degree m requires 3m+2 operations.

The summation requires $m+1$ calculations of $P_k B_k^m(t)$. $P_k B_k^m(t)$ requires 2 calculations, one for each coordinate of the point. The actual summation requiers m operations $\rightarrow$(m+1) $\cdot(3m+3)+m = 3m^2+7m+3$. That is $\mathrm{O}\left(m^2\right)$.

3.1.2 Computational cost of de Casteljau's algorithm

de Casteljau's algorithm:
for j=1:m
for i=0:m-j
$P_i^j = P_i^{j-1} + (P_{i+1}^{j-1} - P_i^{j-1})t_0$

The work carried out in the inner loop is $(m-1)+(m-2)+..+1$, which can be represented with the geometric sum $\frac{(m-1)m}{2}$. 3 operations are carried for each point totaling 3*2 operations. Totaling $3\frac{(m-1)m}{2} = 3m^2 - 3m$ number fo operations.

3.2 Task 2

The implementation of de Casteljau's algorithm can be found in appendix A, section 7.1. A secondary help function used for calculating multiple points of a Bézier curve was implemented and can be found in appendix A, section 7.2. Resulting plots can be found in appendix B, section 8.1. Program listing can be found in appendix C, section 9.1.

3.3 Task 3

Resulting plots can be found in appendix B, section 8.2, 8.3, 8.4, 8.5 and 8.6. Program listing can be found in appendix C, section 9.1. Points used were chosen experimentally and verified visually in the plots.

3.4 Task 4

Resulting plots can be found in appendix B, section 8.7 and 8.8. Program listing can be found in appendix C, section 9.1. The first set of control points used in a) ended where the second set of control points began. Resulting in a sharp angle. In b) the end of the first set of control points were made to have the same differential quotient as the beginning of the second set of control points. Resulting in a smooth link.

3.5 Task 5

Resulting plots can be found in appendix B, section 8.9. Program listing can be found in appendix C, section 9.1.

3.6 Task 6

The implementation of the splitting functions can be found in appendix A, section 7.3 and 7.4. Resulting plots can be found in appendix B, section 8.10. Program listing can be found in appendix C, section 9.1.

3.7 Task 7

The implementation of the rectangle function can be found in appendix A, section 7.5. Resulting plots can be found in appendix B, section 8.11. Program listing can be found in appendix C, section 9.1.

3.8 Task 8

An implementation of the intersection detecting function can be found in appendix A, section 7.6. Resulting plots can be found in appendix B, section 8.12. Program listing can be found in appendix C, section 9.1.

3.9 Task 9

Resulting plots can be found in appendix B, section 8.13 and 8.14. Program listing can be found in appendix C, section 9.1.

3.10 Task 10

Resulting plots can be found in appendix B, section 8.15 and 8.16. Program listing can be found in appendix C, section 9.1. A lower case 'f' was chosen due to the longhand capital 'f' using multiple lines.

4 Analysis

Understanding the way the splitting function works was difficult was somewhat time-consuming.

The performance of the intersection method is not optimal as every splitted set of control points of the first curve were compared against every splitted set of control points of the second curve. The resulting performance is bad as large number of comparisons need to be made. However the performance was somewhat improved by jumping to the next iteration of the function as soon as a collision was detected

5 Lessons learned

A few new Matlab functions were learned as noted in the numerical considerations section. The most useful being cell-matrices, that were used as an array of matrices.

Hands on experience with building curves to a certain specification.

I learned an iterative method of calculating intersections using rectangles and splitting.

6 Acknowledgements

Angelica Gabasio, Mikael Nilsson and Mikael Sahlström.
Numerical Analysis, Timothy Sauer. Pearson Education, 2005.

7 Appendix: A

7.1 de Casteljau

```
function [point, points] = decasteljau(t0, points)
% Soruce: Slides, lecture #10
%
% Input:
% t0     Compute a point for t = t0, on the Bézier curve
% points The control points for the Bézier curve
%
% Output:
% point  Point on the current Bézier curve for the given t0
% prevPoint The innermost controlpoints used to calculate a point on the current Bézier curve

    dim = size(points);
    n = dim(1);
    dimen = 2;
    for j=1:n+1
        for i=1:n-j
            points(i:i,1:dimen) = points(i:i,1:dimen)*(1-t0)+ points(i+1:i+1,1:dimen)*t0;
        end
    end
    point = points(1:1,1:dimen);
end
```

7.2 Wrap

```
function [x, y] = wrap(nbrPoints, points)
% Wrap the decasteljau algorithm for easier syntax and better readability
%
% Input:
% nbrPoints   Desired number of points of the resulting Bezier curve
% points      Control points used to create Bezier curve
%
% Output:
% x           Vector of x-coordinates of the points on the Bezier curve
% y           Vector of y-coordinates of the points on the Bezier curve

n=nbrPoints;
t = linspace(0,1,n+1);
bezier = zeros(n+1,2);
for i = 1:n+1
    [point, ~] = decasteljau(t(i),points);
    bezier(i,1:2) = point;
end
x=bezier(:,1);
y=bezier(:,2);

end
```

7.3 Split lower

```
function [x, y, controlPoints] = splitlower(points, tsplit)
% Wrap the decasteljau algorithm for easier syntax and better readability
% and return Bezier curve of t<=tsplit
%
% Input:
% nbrPoints   Desired number of points of the resulting Bezier curve
% points      Control points used to create Bezier curve
% tsplit      Only return parts of the bezier curve generated with t<tsplit
%
% Output:
% x           Vector of x-coordinates of the innermost points used to calculate the Bézier curve
% y           Vector of y-coordinates of the innermost points used to calculate the Bézier curve
% controlPoints      Matrix of the innermost points used to calculate the Bézier curve

[~, controlPoints] = decasteljau(tsplit,points);

x=controlPoints(:,1);
```

```
y=controlPoints(:,2);

end
```

7.4 Split upper

```
function [x, y, controlPoints] = splitupper(points, tsplit)
% Wrap the decasteljau algorithm for easier syntax and better readability
% and return Bezier curve of t>=tsplit
%
% Input:
% nbrPoints   Desired number of points of the resulting Bezier curve
% points      Control points used to create Bezier curve
% tsplit      Only return parts of the bezier curve generated with t>tsplit
%
% Output:
% x           Vector of x-coordinates of the innermost points used to calculate the Bézier curve
% y           Vector of y-coordinates of the innermost points used to calculate the Bézier curve
% controlPoints      Matrix of the innermost points used to calculate the Bézier curve

[~, controlPoints] = decasteljau(1-tsplit,flipud(points));

x=controlPoints(:,1);
y=controlPoints(:,2);

end
```

7.5 Box

```
function [boxVar, posVect] = box(points)
% Creates a rectangle parallell with the x and y-axis around the convex
% hull of a set of points.
%
% Input:
% points      Points to surround with a rectangle
%
% Output:
% boxVar      The points of the smallest possible rectangle surrounding the convex hull of points
% posVect     Position vector of the rectangle

x = points(:,1);
y = points(:,2);
k = convhull(x,y);

xMin = min(x(k));
yMin = min(y(k));
xMax = max(x(k));
yMax = max(y(k));

boxVar(1:1,1:2) = [xMin yMin;];
boxVar(2:2,1:2) = [xMin yMax;];
boxVar(3:3,1:2) = [xMax yMax;];
boxVar(4:4,1:2) = [xMax yMin;];
boxVar(5:5,1:2) = [xMin yMin;];

posVect = [xMin yMin (xMax-xMin) (yMax-yMin)];

end
```

7.6 Intersection

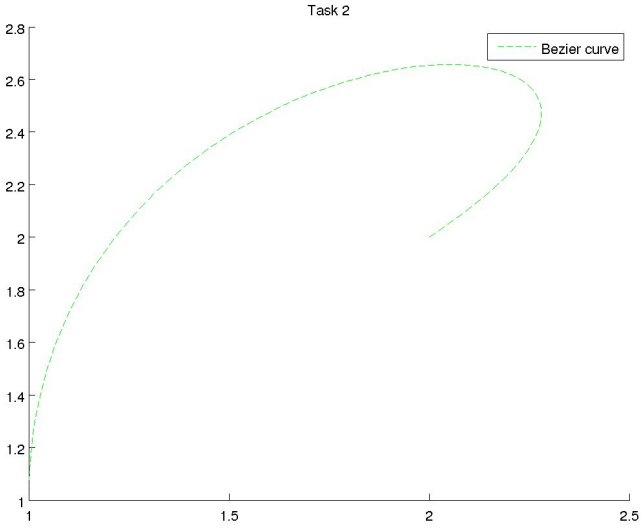
```
function [boolean] = intersection(points1, points2, maxIterations)
% Calculate if two curves intersect
%
% Input:
% points1     The control points of bezier curve 1
% points2     The control points of bezier curve 2
% maxIterations Maximum number of iterations to run this algorithm
%
% Output:
```

```
% boolean     Do the curves intersect. 0 = no

[~,posVect1] = box(points1);
[~,posVect2] = box(points2);
boolean = rectint(posVect1,posVect2);
if boolean == 0
    return
end
cp1{1} = points1;
cp2{1} = points2;
if 1 < maxIterations
    boolean = innerIntersect(cp1, cp2, 2, maxIterations);
end
end



function [boolean] = innerIntersect(cp1, cp2, currentIteration, maxIterations)
% Inner recurvsive function splitting the control points into sub control points and
% checking the rectangles around the sub control points for intersection
%
% Input:
% cp1         Subcontrolpoints for the first curve
% cp2         Subcontrolpoints for the second curve
% currentIteration Current iteration number
% maxIterations Maximum number of iterations to run this algorithm
%
% Output:
% boolean     Do the curves intersect. 0 = no

cp1dim = size(cp1);
nbrCp1 = cp1dim(2);
cp2dim = size(cp2);
nbrCp2 = cp2dim(2);
% Split curves into sub control points
newCp1{1} =[0 0];
for i=1:nbrCp1
    disp('Splitting curve 1')
    tempCp = cp1{i};
    [~, ~, tempCpL] = splitlower(tempCp,0.5);
    newCp1{(2*i -1)} = tempCpL;
    [~, ~, tempCpU] = splitupper(tempCp,0.5);
    newCp1{(2*i)} = tempCpU;
end
newCp2{1} =[0 0];
for i=1:nbrCp2
    disp('Splitting curve 2')
    tempCp = cp2{i};
    [~, ~, tempCpL] = splitlower(tempCp,0.5);
    newCp2{(2*i -1)} = tempCpL;
    [~, ~, tempCpU] = splitupper(tempCp,0.5);
    newCp2{(2*i)} = tempCpU;
end

% Look for intersection in rectangles of sub control points
nbrNewCp1 = nbrCp1*2;
nbrNewCp2 = nbrCp2*2;
boolean = 0;
for i=1:nbrNewCp1

    tempCp1 = newCp1{i};
    for j=1:nbrNewCp2
       disp(sprintf('Iter: %d; Checking subcurve %d against %d',currentIteration,i,j))
       tempCp2 = newCp2{j};
       [~,posVect1] = box(tempCp1);
       [~,posVect2] = box(tempCp2);
       boolean = rectint(posVect1,posVect2);
       if boolean ~= 0
           break
       end
    end
    if boolean ~= 0
       break
```

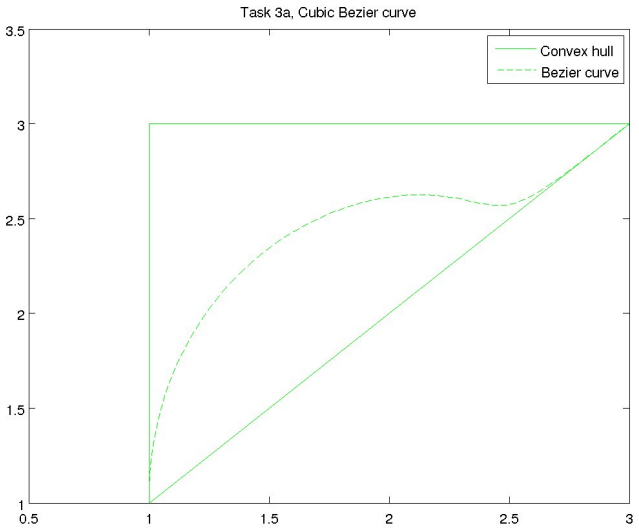
```
    end
end

if boolean == 0
    disp('innerIntersect: boolean == 0')
    return
end
if currentIteration < maxIterations
    disp(sprintf('innerIntersect: currentIteration < maxIterations  -- %d < %d', currentIteration,maxIterations))
    boolean = innerIntersect(newCp1, newCp2, currentIteration+1, maxIterations);
else
    disp(sprintf('innerIntersect: currentIteration >= maxIterations  -- %d >= %d', currentIteration,maxIterations))
    return
end



end
```

8 Appendix: B

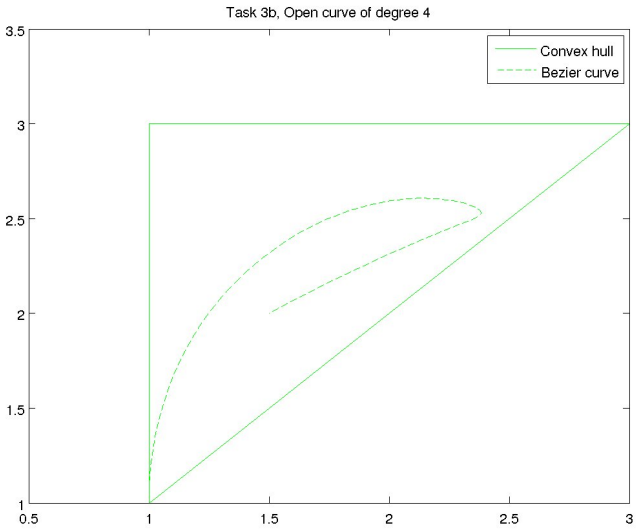
8.1 Task 2



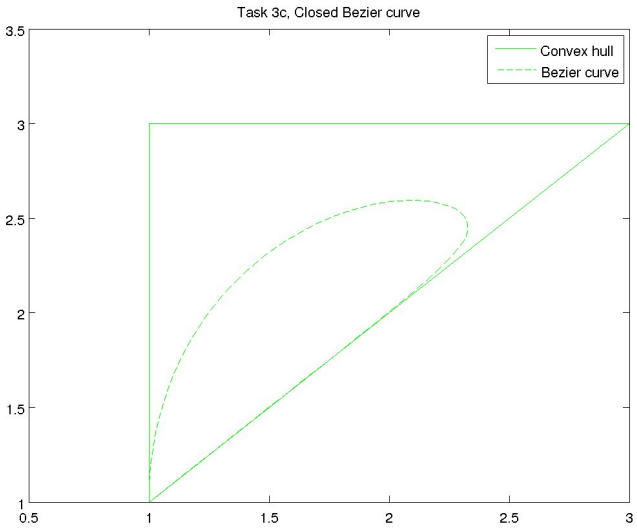
8.2 Task 3a



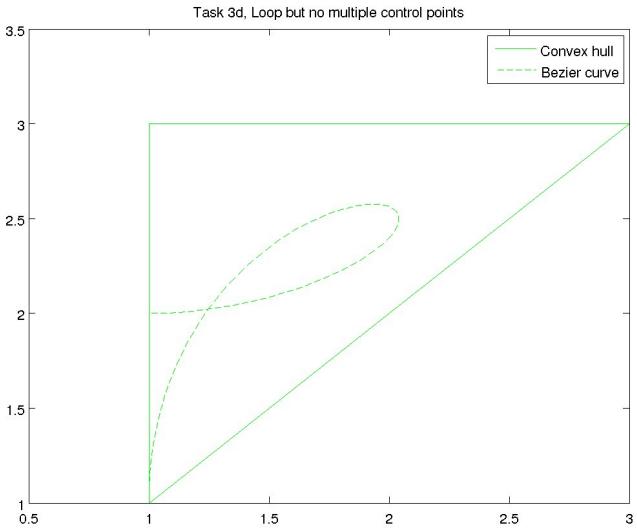
8.3 Task 3b



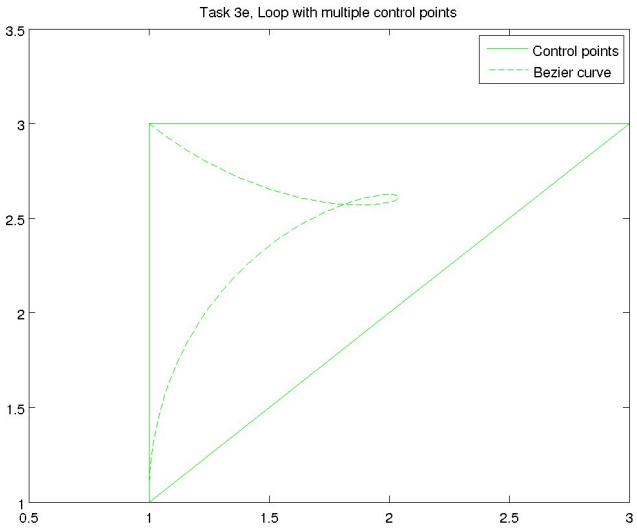
8.4 Task 3c



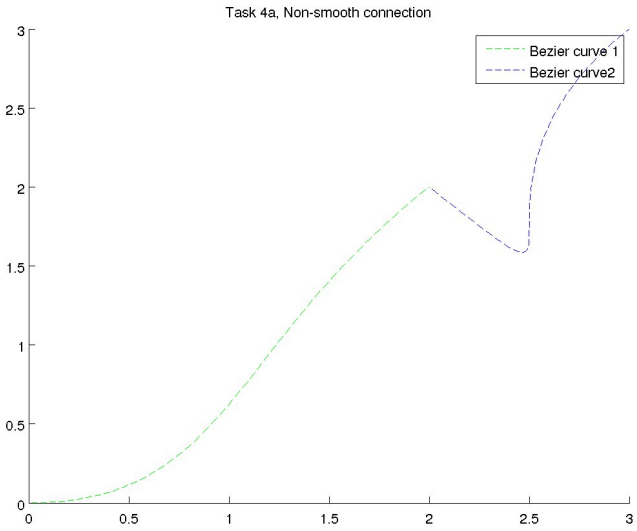
8.5 Task 3d



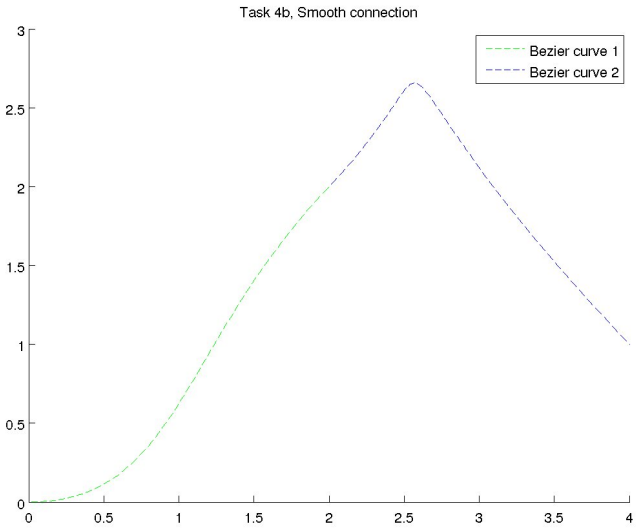
8.6 Task 3e



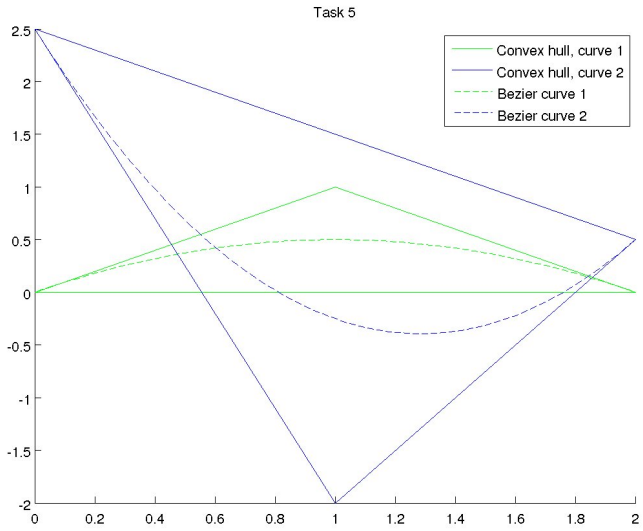
8.7 Task 4a



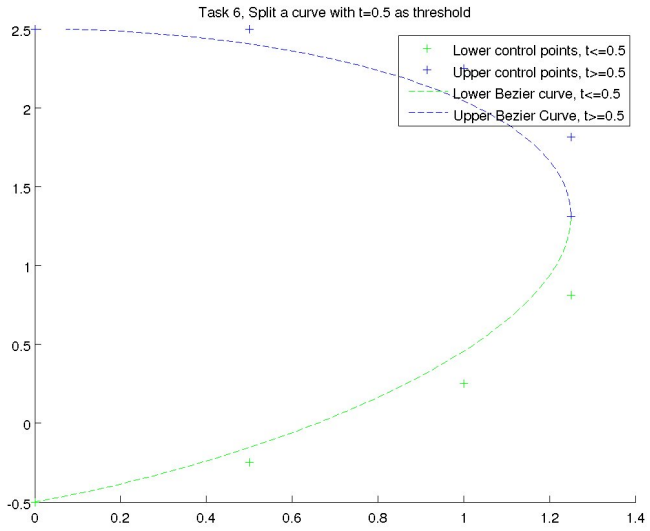
8.8 Task 4b



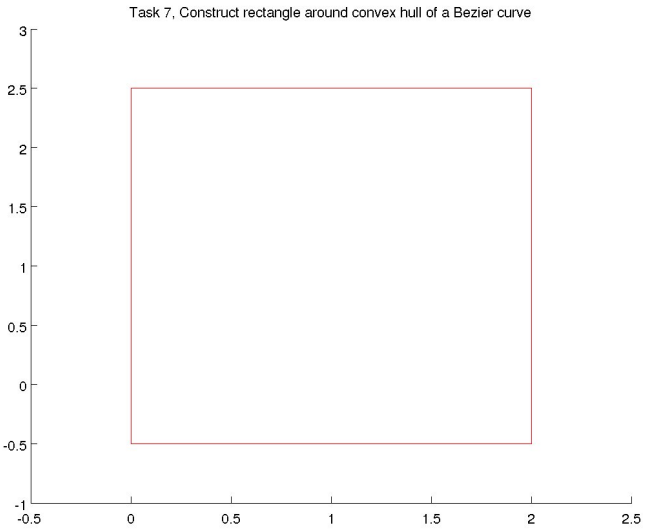
8.9 Task 5



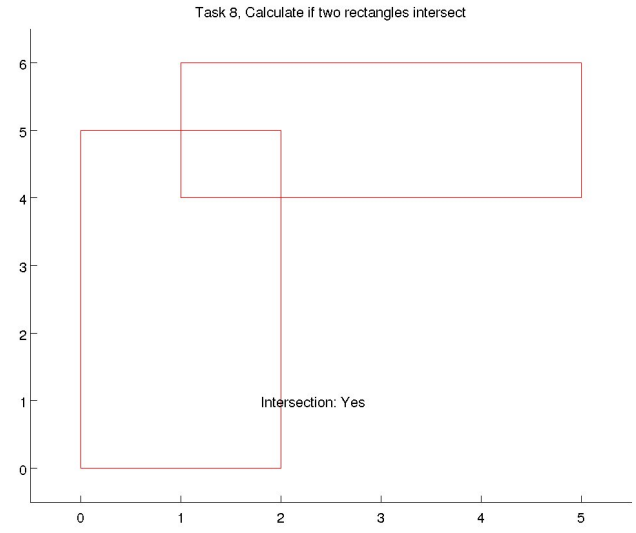
8.10 Task 6



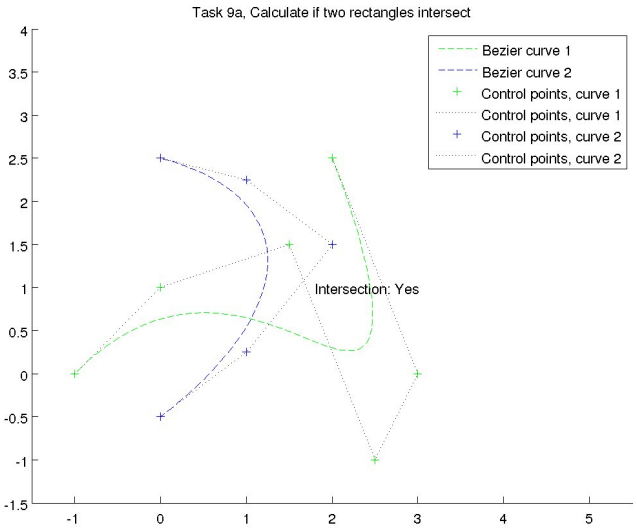
8.11 Task 7



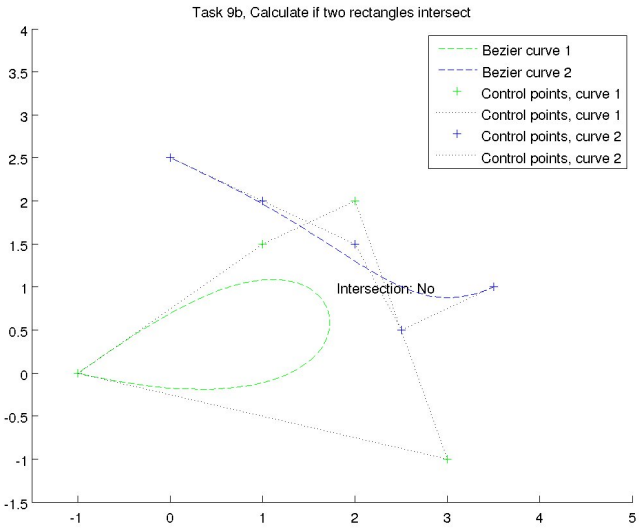
8.12 Task 8



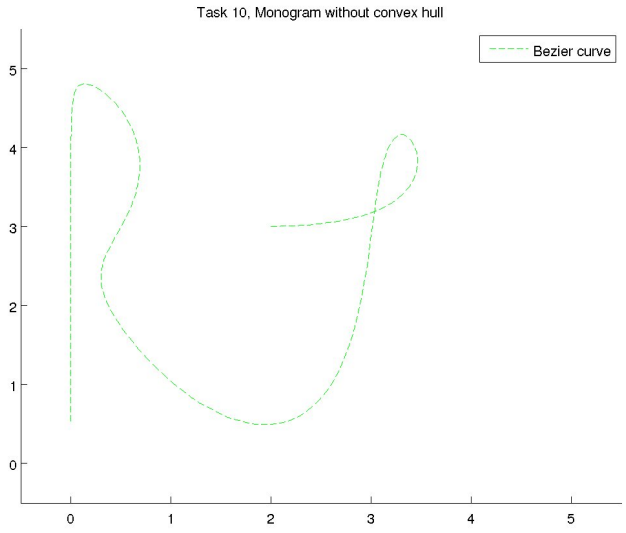
8.13 Task 9a



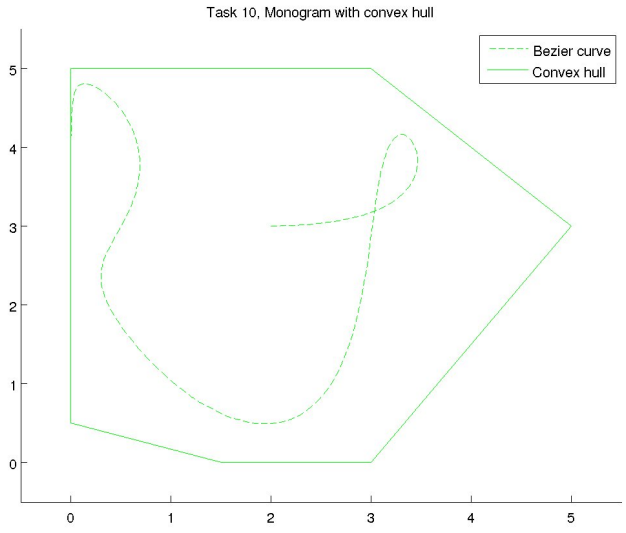
8.14 Task 9b



8.15 Task 10a



8.16 Task 10b



9 Appendix: C

9.1 Program listing

```
% Reset
clear all
close all
hold off


% Task 1
points = [1 1; 1 3; 3 3; 2 2];
bezier = decasteljau(0.1,points);


% Task 2
hold on
title('Task 2');
[x,y]=wrap(100,points);
plot(x,y,'g--');
legend('Bezier curve');
saveas(gcf(),'task2.png');


% Task 3 a A cubic Bezier curve
figure();
points = [1 1; 1 3; 3 3; 2 2; 3 3;];
x = points(:,1);
y = points(:,2);
k = convhull(x,y);

plot(x(k),y(k),'g');
hold on
title('Task 3a, Cubic Bezier curve');
[x,y]=wrap(100,points);
plot(x,y,'g--');

axis([0.5, 3, 1, 3.5])
legend('Convex hull','Bezier curve');
saveas(gcf(),'task3a.png');


% Task 3 b An open curve of degree 4
figure();
points = [1 1; 1 3; 3 3; 2 2; 3 3; 1.5 2;];
x = points(:,1);
y = points(:,2);
k = convhull(x,y);

plot(x(k),y(k),'g');
hold on
title('Task 3b, Open curve of degree 4');

[x,y]=wrap(100,points);
plot(x,y,'g--');
axis([0.5, 3, 1, 3.5])
legend('Convex hull','Bezier curve');
saveas(gcf(),'task3b.png');


% Task 3 c Closed Bezier curve
figure;
points = [1 1; 1 3; 3 3; 2 2; 3 3; 1 1;];
x = points(:,1);
y = points(:,2);
k = convhull(x,y);

plot(x(k),y(k),'g');
hold on
title('Task 3c, Closed Bezier curve');

[x,y]=wrap(100,points);
plot(x,y,'g--');
```

```
axis([0.5, 3, 1, 3.5])
legend('Convex hull','Bezier curve');
saveas(gcf(),'task3c.png');


% Task 3 d Loop but no multiple control points
figure;
points = [1 1; 1 3; 3 3; 2 2; 1 2;];
x = points(:,1);
y = points(:,2);
k = convhull(x,y);

plot(x(k),y(k),'g');
hold on
title('Task 3d, Loop but no multiple control points');

[x,y]=wrap(100,points);
plot(x,y,'g--');
axis([0.5, 3, 1, 3.5])
legend('Convex hull','Bezier curve');
saveas(gcf(),'task3d.png');

% Task 3 e Loop with multiple control points
figure;
points = [1 1; 1 3; 3 3; 2 2; 1 3;];
x = points(:,1);
y = points(:,2);
k = convhull(x,y);

plot(x(k),y(k),'g');
hold on
title('Task 3e, Loop with multiple control points');

[x,y]=wrap(100,points);
plot(x,y,'g--');
axis([0.5, 3, 1, 3.5])
legend('Control points','Bezier curve');
saveas(gcf(),'task3e.png');


% Task 4 a do not connect in a smooth way
figure;
hold on;
title('Task 4a, Non-smooth connection');

points1 = [0 0; 1 0; 1 1; 2 2;];
points2 = [2 2; 3 1; 2 2; 3 3];

[x1,y1]=wrap(100,points1);
[x2,y2]=wrap(100,points2);
plot(x1,y1,'g--');
plot(x2,y2,'b--');
lgnd = legend('Bezier curve 1','Bezier curve2');
set(lgnd,'color','none');
saveas(gcf(),'task4a.png');


% Task 4 b connect in a smooth way
figure;
hold on;
title('Task 4b, Smooth connection');
points1 = [0 0; 1 0; 1 1; 2 2;];
points2 = [2 2; 3 3; 2 3; 4 1;];

[x1,y1]=wrap(100,points1);
[x2,y2]=wrap(100,points2);
plot(x1,y1,'g--');
plot(x2,y2,'b--');
lgnd=legend('Bezier curve 1','Bezier curve 2');
set(lgnd,'color','none');
saveas(gcf(),'task4b.png');


% Task 5 Plot the Bezier curves with control points
```

```
%[ (0,0) , (1,1) , (2,0) ] and
%[ (0,2.5) , (1 -2) , (2, 0.5) ] and their convex hulls.
figure;
hold on;
title('Task 5');
points1 = [0 0; 1 1; 2 0;];
points2 = [0 2.5; 1 -2; 2 0.5;];

x1 = points1(:,1);
y1 = points1(:,2);
k1 = convhull(x1,y1);
x2 = points2(:,1);
y2 = points2(:,2);
k2 = convhull(x2,y2);
plot(x1(k1),y1(k1),'g');
plot(x2(k2),y2(k2),'b');

[x1,y1]=wrap(100,points1);
[x2,y2]=wrap(100,points2);
plot(x1,y1,'g--');
plot(x2,y2,'b--');
lgnd=legend('Convex hull, curve 1','Convex hull, curve 2','Bezier curve 1','Bezier curve 2');
set(lgnd,'color','none');
saveas(gcf(),'task5.png');


% Task 6 Split curve into left and right curves with t=0.5 as a threshold
figure;
hold on
title('Task 6, Split a curve with t=0.5 as threshold');
points = [0 2.5; 1 2.5; 2 1.5; 1 0; 0 -0.5;];
t = 0.5;

[~,~,controlPoints1] = splitlower(points,t);
plot(controlPoints1(:,1),controlPoints1(:,2),'+g');

[~,~,controlPoints2] = splitupper(points,t);
plot(controlPoints2(:,1),controlPoints2(:,2),'+b');

[x1,y1]=wrap(100,controlPoints1);
[x2,y2]=wrap(100,controlPoints2);
plot(x1,y1,'g--');
plot(x2,y2,'b--');

lgnd=legend('Lower control points, t<=0.5','Upper control points, t>=0.5','Lower Bezier curve, t<=0.5','Upper Bezier Curve, t>=
set(lgnd,'color','none');
saveas(gcf(),'task6.png');


% Task 7 construct the smallest rectangle parallel to the x
% and y axes that contains the convex hull of a given Bezier
figure;
hold on
title('Task 7, Construct rectangle around convex hull of a Bezier curve');
points = [0 2.5; 1 2.5; 2 1.5; 1 0; 0 -0.5;];

[box1, ~] = box(points);
plot(box1(:,1),box1(:,2),'r');
axis([-0.5, 2.5, -1, 3])
saveas(gcf(),'task7.png');


% Task 8 calculate the intersection of two rectangles
figure;
hold on
title('Task 8, Calculate if two rectangles intersect');
points1 = [0 0; 2 0; 2 5; 0 5;];
points2 = [1 4; 5 4; 5 6; 1 6;];

[box1,posVect1] = box(points1);
[box2,posVect2] = box(points2);
plot(box1(:,1),box1(:,2),'r');
plot(box2(:,1),box2(:,2),'r');
boolean = rectint(posVect1, posVect2);
```

```
if (boolean > 0)
    txt = 'Intersection: Yes';
else
    txt = 'Intersection: No';
end
text(1.8,1,txt);
axis([-0.5, 5.5, -0.5, 6.5])
saveas(gcf(),'task8.png');


% Task 9a calculate if the Bezier curves intersect
figure;
hold on
title('Task 9a, Calculate if two rectangles intersect');
points1 = [-1 0; 0 1; 1.5 1.5; 2.5 -1; 3 0; 2 2.5;];
points2 = [0 2.5; 1 2.25; 2 1.5; 1 0.25; 0 -0.5;];
boolean = intersection(points1, points2, 3);

if (boolean > 0)
    txt = 'Intersection: Yes';
else
    txt = 'Intersection: No';
end
text(1.8,1,txt);

[x1,y1]=wrap(100,points1);
[x2,y2]=wrap(100,points2);
plot(x1,y1,'g--');
plot(x2,y2,'b--');

[~,~,controlPoints1] = splitlower(points1,0);
plot(controlPoints1(:,1),controlPoints1(:,2),'+g');
plot(controlPoints1(:,1),controlPoints1(:,2),'k:');
[~,~,controlPoints1] = splitlower(points2,0);
plot(controlPoints1(:,1),controlPoints1(:,2),'+b');
plot(controlPoints1(:,1),controlPoints1(:,2),'k:');

axis([-1.5, 5.5, -1.5, 4])
lgnd=legend('Bezier curve 1','Bezier curve 2','Control points, curve 1','Control points, curve 1','Control points, curve 2','Co
set(lgnd,'color','none');
saveas(gcf(),'task9a.png');


% Task 9b calculate if the Bezier curves intersect
figure;
hold on
title('Task 9b, Calculate if two rectangles intersect');
points1 = [-1 0; 1 1.5; 2 2; 3 -1; -1 0;];
points2 = [0 2.5; 1 2; 2 1.5; 2.5 0.5; 3.5 1;];

boolean = intersection(points1, points2, 3);

if (boolean > 0)
    txt = 'Intersection: Yes';
else
    txt = 'Intersection: No';
end
text(1.8,1,txt);

[x1,y1]=wrap(100,points1);
[x2,y2]=wrap(100,points2);
plot(x1,y1,'g--');
plot(x2,y2,'b--');

[~,~,controlPoints1] = splitlower(points1,0);
plot(controlPoints1(:,1),controlPoints1(:,2),'+g');
plot(controlPoints1(:,1),controlPoints1(:,2),'k:');
[~,~,controlPoints1] = splitlower(points2,0);
plot(controlPoints1(:,1),controlPoints1(:,2),'+b');
plot(controlPoints1(:,1),controlPoints1(:,2),'k:');

axis([-1.5, 5, -1.5, 4])
lgnd=legend('Bezier curve 1','Bezier curve 2','Control points, curve 1','Control points, curve 1','Control points, curve 2','Co
```

```
set(lgnd,'color','none');
saveas(gcf(),'task9b.png');


% Task 10a
figure;
hold on;
title('Task 10, Monogram without convex hull');
points = [0 0.5; 0 5; 0 5; 0 5; 0 5; 0 5; 0 5; 4 3.75; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 1 0.25;
    1.5 0; 1.5 0; 1.5 0; 3 0; 3 0; 3 0; 3 3; 3 3; 3 4.5; 3 5; 3 5; 5 3; 2 3;];

[x,y] = wrap(200,points);
plot(x,y,'g--');

axis([-0.5, 5.5, -0.5, 5.5])
lgnd=legend('Bezier curve');
set(lgnd,'color','none');
saveas(gcf(),'task10a.png');


% Task 10b
figure;
hold on;
title('Task 10, Monogram with convex hull');
points = [0 0.5; 0 5; 0 5; 0 5; 0 5; 0 5; 0 5; 4 3.75; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 0 2.5; 1 0.25;
    1.5 0; 1.5 0; 1.5 0; 3 0; 3 0; 3 0; 3 3; 3 3; 3 4.5; 3 5; 3 5; 5 3; 2 3;];

[x,y] = wrap(200,points);
plot(x,y,'g--');

x = points(:,1);
y = points(:,2);
k = convhull(x,y);
plot(x(k),y(k),'g');

axis([-0.5, 5.5, -0.5, 5.5])
lgnd=legend('Bezier curve', 'Convex hull');
set(lgnd,'color','none');
saveas(gcf(),'task10b.png');
```