

# MATH2070: LAB 8: Higher Order Interpolation and Mesh Generation

Introduction	Exercise 1
Parametric Interpolation	Exercise 2
Cubic Hermite Interpolation	Exercise 3
Two-dimensional Hermite interpolation and mesh generation	Exercise 4
Varying parameters	Exercise 5
Matching patches	Exercise 6
	Exercise 7
	Exercise 8

## 1 Introduction

Piecewise linear interpolation has many good properties. In particular, if the data come from a continuously differentiable function  $f(x)$  and if the data points are suitably spread throughout the closed interval, then the interpolant converges to the function. The resulting interpolant consists of straight line segments and so has flats and kinks in it, making it inappropriate for many applications.

In this lab we will look at one example of piecewise polynomial interpolation that has curved pieces with continuous derivatives from patch to patch. You have seen spline interpolation, but we will be looking at piecewise Hermite cubic interpolation. We will apply this method first to parametric curves and then to two-dimensional patches. The two-dimensional case is useful because of applications to mesh generation for solving partial differential equations.

This lab will take three sessions. If you print this lab, you may prefer to use the pdf version.

## 2 Parametric Interpolation

You are familiar with a *parameterized* curve, in which a special variable, typically called  $t$ , is used to draw objects for which the relationship between  $x$  and  $y$  is not functional. One good definition of a circle is:

$$\begin{aligned}x &= \cos(t) \\y &= \sin(t)\end{aligned}$$

for  $0 \leq t \leq 2\pi$ . Writing the circle this way involves regarding both  $x$  and  $y$  as functions of the independent variable  $t$ .

Suppose we wanted to do interpolation of some complicated curve? If we have a drawing of the curve, we could simply mark points along the curve that are roughly evenly spaced, and make tables of `tdata`, `xdata` and `ydata`, where the values of `tdata` could be 1, 2, 3, ...

Now if we want to compute intermediate points on the curve, that's really saying, for a given value of `tdata`, what would the corresponding values of `xdata` and `ydata` be? This means doing two interpolation problems, one for `xdata` as a function of `tdata` and the second for `ydata` as a function of `tdata`.

You may recall that a cardioid is a roughly heart-shaped closed curve, but it has no sharp point at the bottom. You can find a nice description of cardioid curves with various ways of defining them at <http://mathworld.wolfram.com/Cardioid.html> One cardioid can be given by the equations

$$\begin{aligned}r &= r_0 + \cos t \\x &= r \cos t \\y &= r \sin t.\end{aligned}\tag{1}$$

**Exercise 1:** Try the following method for plotting a cardioid. Use `eval_plin.m` from last lab, or you can download `eval_plin.m` and `bracket.m` from the web page. Copy the following code to a script m-file named `exer1.m`

```
% generate the data points for a cardioid
ndata = 21;
tdata = linspace ( 0, 2*pi, ndata );
r = 0.5 + cos ( tdata ); % equation for a cardioid
xdata = r .* cos ( tdata );
ydata = r .* sin ( tdata );

% interpolate them
tval = linspace ( 0, 2*pi, 10*ndata );
xval = eval_plin ( tdata, xdata, tval );
yval = eval_plin ( tdata, ydata, tval );

% plot them with correct aspect ratio
plot ( xval, yval )
axis equal
```

You do not need to send me this plot.

You can see that the plot of the cardioid in Exercise 1 is not very smooth. This is because the line segments are straight lines that meet at corners. In the following section you will see one way of interpolating curves using curved sections whose derivatives are continuous so that there are no corners.

### 3 Cubic Hermite Interpolation

Hermite interpolation is discussed in Atkinson, Section 3.6.

If we were trying to design, say, the shape of the sheet metal pattern for a car door, kinks and corners would not be acceptable. If that's a problem, then perhaps we could try to make sure that our interpolant is smoother and simpler by matching *both* the function value and the derivative at each point. (*This assumes you can get the derivative value.*)

Suppose we have points  $x_k$ ,  $k = 1, 2, \dots, N$  and a differentiable function  $f(x)$ , so that the values  $y_k = f(x_k)$  and  $y'_k = f'(x_k)$  can be regarded as data points.

Now consider the situation in the  $k^{\text{th}}$  interval,  $[x_k, x_{k+1}]$ . There is just one cubic polynomial on this interval that has the appropriate function and derivative values at the two endpoints. You may recall the Lagrange interpolation polynomials from Lab 6. Their claim to fame is that on the interval  $[x_k, x_{k+1}]$  there are two special linear polynomials: one that is 1 at  $x_k$  and zero at  $x_{k+1}$ , and the other is 1 at  $x_{k+1}$  and zero at  $x_k$ . This feature makes it easy to construct an arbitrary linear polynomial that matches given values at the endpoints. We want to match *both* the function *and* derivative values at the endpoints, so we will need cubic Hermite polynomials.

Consider the four Hermite polynomials

$$\begin{aligned} h_1(x) &= \frac{(x - x_{k+1})^2(3x_k - x_{k+1} - 2x)}{(x_k - x_{k+1})^3} \\ h_2(x) &= \frac{(x - x_k)(x - x_{k+1})^2}{(x_k - x_{k+1})^2} \\ h_3(x) &= \frac{(x - x_k)^2(3x_{k+1} - x_k - 2x)}{(x_{k+1} - x_k)^3} \\ h_4(x) &= \frac{(x - x_{k+1})(x - x_k)^2}{(x_{k+1} - x_k)^2} \end{aligned}$$

These four cubic polynomials satisfy the following equalities, that are similar to Equations (3.6.11) in Atkinson.

	$x_k$	$x_{k+1}$
$h_1(x)$	1	0
$h'_1(x)$	0	0
$h_2(x)$	0	0
$h'_2(x)$	1	0
$h_3(x)$	0	1
$h'_3(x)$	0	0
$h_4(x)$	0	0
$h'_4(x)$	0	1

(2)

(This table means that, for example,  $h_2(x_k) = 0$ ,  $h'_2(x_k) = 1$ ,  $h_2(x_{k+1}) = 0$ , and  $h'_2(x_{k+1}) = 0$ .) If you know Mathematica, Maple or the symbolic toolbox in Matlab, you can check these equalities easily enough. For the purpose of this lab, the easiest one to check is  $h_2$ , so check it by hand. (Hint: Do not multiply the factors out! You can check by staring at the formula long enough.)

Given these four polynomials, the unique polynomial with the values

$$\begin{aligned} y_k &= f(x_k) \\ y'_k &= f'(x_k) \\ y_{k+1} &= f(x_{k+1}) \\ y'_{k+1} &= f'(x_{k+1}) \end{aligned}$$

can be written as

$$p(x) = y_k h_1(x) + y'_k h_2(x) + y_{k+1} h_3(x) + y'_{k+1} h_4(x). \tag{3}$$

We are now in a position to write a piecewise Hermite interpolation routine. This routine will be similar to `eval_plin.m`.

**Exercise 2:**

- (a) Begin a Matlab function m-file called `eval_herm.m` with the signature
 

```
function yval = eval_herm ( xdata, ydata, ypdata, xval )
```

 and insert comments. the values in `ypdata` are the derivative values at the points `xdata`.
- (b) Use the `bracket` function to find the interval in which `xval` lies. As in `eval_plin.m`, this can be done in a single line.
- (c) Evaluate the four Hermite functions,  $h_1(x)$ ,  $h_2(x)$ ,  $h_3(x)$ , and  $h_4(x)$  at `xval`. Use componentwise operations if you can or use loops.
- (d) Evaluate `yval` using the expression (3) for the Hermite interpolating polynomial.

- (e) Check your work by using `eval_herm` to compute the values at four interior test points of your choice for each of the polynomials  $y = 1$ ,  $y = x$ ,  $y = x^2$ , and  $y = x^3$  interpolated on the interval `xdata=[0,2]`. Recall that four points uniquely determine a cubic polynomial, so if you get agreement within roundoff at four points, you know your code is correct. **If your code is not correct, you can debug in the following way. You only need to do the following steps if you are debugging.**
- i. Try the four polynomials  $y = 1$ ,  $y = x$ ,  $y = x^2$ , and  $y = x^3$  on the interval `xdata=[0,1]` one at a time. If these all work but `eval_herm` does not, odds are you have an error in the denominators of one or more of the  $h_k(x)$ .
  - ii. If you cannot get agreement on `[0,1]`, use `eval_herm` to reproduce each of the  $h_k(x)$  for `xdata=[0,1]`. The table (2) above provides `ydata` and `ypdata` values. Check your results at a minimum of four points. If these all agree but `eval_herm` still does not work correctly, then your implementation of (3) is incorrect.
  - iii. If you have completed the previous two steps and fixed all errors you found, but you still cannot interpolate the given monomials on `[0,2]`, repeat the previous steps using the interval `[0,2]`. When you are finished you *must* have found your errors.

Now, we will use `eval_herm.m` to see how piecewise Hermite converges.

### Exercise 3:

- (a) First, modify your Matlab m-file `runge.m` so that it returns both the Runge function  $y = 1/(1+x^2)$  and its derivative  $y'$ . I will leave it to you to compute the derivative expression, but be sure you are correct. The signature of the function should be

```
function [y,yprime]= runge ( x )
% comments
```

and it should have appropriate comments. Do not forget to use componentwise syntax.

- (b) Copy your `test_piece_interpolate.m` function from last lab, or download my version. Rename it to `test_piece_hermite.m` and modify it to use `eval_herm.m`.
- (c) Using equally spaced data points from -5 to 5 for `xdata` and the `runge.m` function to compute `ydata` and `ypdata`. Estimate the maximum interpolation error by computing the maximum absolute difference between the exact and interpolated values at 4001 evenly spaced points. Fill in the following table:

Runge function, Piecewise Hermite Cubic			
<code>ndata = 5</code>	<code>Err( 5) =</code>	<code>-----</code>	
<code>ndata = 11</code>	<code>Err( 11) =</code>	<code>-----</code>	<code>Err( 5)/Err( 11) =</code>
<code>ndata = 21</code>	<code>Err( 21) =</code>	<code>-----</code>	<code>Err( 11)/Err( 21) =</code>
<code>ndata = 41</code>	<code>Err( 41) =</code>	<code>-----</code>	<code>Err( 21)/Err( 41) =</code>
<code>ndata = 81</code>	<code>Err( 81) =</code>	<code>-----</code>	<code>Err( 41)/Err( 81) =</code>
<code>ndata =161</code>	<code>Err(161) =</code>	<code>-----</code>	<code>Err( 81)/Err(161) =</code>
<code>ndata =321</code>	<code>Err(321) =</code>	<code>-----</code>	<code>Err(161)/Err(321) =</code>
<code>ndata =641</code>	<code>Err(641) =</code>	<code>-----</code>	<code>Err(321)/Err(641) =</code>

- (d) Estimate the rate of convergence by examining the ratios `Err(41)/Err(81)`, *etc.*, and estimating the nearest power of two. You should observe the theoretical convergence rate, which is larger than 2.

Now, you are in a position to apply Hermite interpolation to generate a very smooth cardioid.

### Exercise 4:

- (a) Copy your script m-file `exer1.m` from Exercise 1 and rename it to `exer4.m`.
- (b) Differentiate the equations (1) defining the cardioid and add expressions for `xpdata` and `ypdata` to `exer4.m`.
- (c) Replace `eval_plin` with `eval_herm`.
- (d) Plot the cardioid. Please include this plot with your summary file. If this plot does not appear smooth or if you can notice the boundaries of the pieces that make it up, you probably have an error in your derivatives.

## 4 Two-dimensional Hermite interpolation and mesh generation

In order to solve a partial differential equation, the region over which it is to be solved must be broken into small pieces. These pieces are often called “mesh elements,” and the process of generating them for arbitrary regions is called “mesh generation.” You can see some examples of complicated meshes at <http://www.geuz.org/gmsh/gallery/spirale.gif> The process of building the outline of a complicated assembly and then using it to generate a mesh is very labor-intensive and can take weeks. The discussion in this lab is most relevant to generation of quadrilateral and hexahedral meshes, not triangular or tetrahedral meshes.

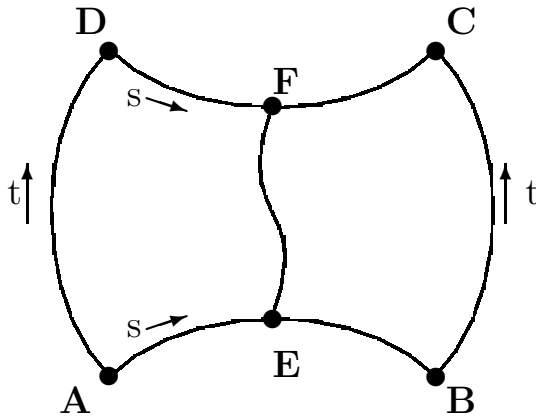
The lowest-level activity in generating meshes in many commercial packages is defining “Coons Patches.” These are two- or three-dimensional geometrical entities that can, on the one hand, be combined together smoothly, and, on the other hand, easily divided into mesh elements. These Coons Patches are generally built using bicubic Hermite polynomials in two dimensions, and tricubic Hermite polynomials in three dimensions. We will be focussing on two dimensions in this lab.

Coons patches are also used in computer graphics, but I am not familiar enough with this application to discuss it.

One approach to generating two-dimensional meshes is to take a drawing of the object and cover it with reasonably large patches with four curved sides. These patches generally are chosen to match up (to a reasonable approximation) with the boundaries of the object and fit against each other and are no smaller than necessary for these tasks. Once all the object in the drawing is completely covered with patches, the patches themselves are broken into smaller mesh pieces. It is important that the mesh lines do not undergo abrupt changes across patch boundaries and it is also important that the user be able to modify the density of mesh lines and to concentrate them near, for example, one boundary. Commercial programs such as Patran and Ansys use this approach.

A two-dimensional bicubic Hermite patch is a smooth map from the unit square  $(s, t) \in [0, 1] \times [0, 1]$  to a region  $(x(s, t), y(s, t)) \in \mathbb{R}^2$ . This mapping is a cubic polynomial in  $s$  for each fixed  $t$  and a cubic polynomial in  $t$  for each fixed  $s$ . A similar definition holds for a two-dimensional patch on a surface in  $\mathbb{R}^3$ , with an additional function  $z(s, t)$ .

Consider the following sketch.



The four outer curves, AB, BC, DC, and AD will form the boundary of the patch. The curves AB and DC are parameterized using a Hermite cubic in  $s$  (with positive direction shown in the figure) and the curves BC and AD are parameterized using a Hermite cubic in  $t$  (with positive direction shown in the figure). Obviously, the coordinates of each of the points A, B, C, and D will be needed. Furthermore, since these are Hermite cubics, the derivatives of  $x(s, t)$  and  $y(s, t)$  with respect to both  $s$  and  $t$  will be needed at each of the four corners. Finally, the cross-derivatives  $\partial^2 x / \partial s \partial t$  will be needed at each of the four corners. These sixteen data are needed to construct the Coons patch.

Among these sixteen data, there is enough information to construct the boundary curve AB parametrically using Hermite polynomials for  $x(s, 0)$  and  $y(s, 0)$ , where  $s$  is the independent variable increasing from point A to point B. This interpolation can be done using the `eval_herm.m` function you wrote earlier. Similarly, the curve DC can be constructed using Hermite polynomials to give  $x(s, 1)$  and  $y(s, 1)$ . Lines of constant  $s$ , such as the line EF, can be constructed using the curves AB and DC. The result will be the parametric coordinate functions  $x(s, t)$  and  $y(s, t)$  for all  $(s, t) \in [0, 1] \times [0, 1]$ . This is the approach taken in the following exercise.

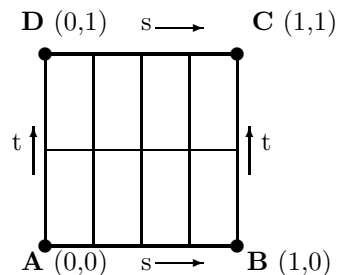
In the next exercise you will write a script m-file that fills a square with smaller squares. This is a special case of mesh generation. In doing this exercise, we will *not* use any special facts about squares, but will be writing as if the lines were curved. In subsequent exercises, we will see what meshes with curved lines look like.

The four corners, A, B, C, and D, each require data for  $x$  and  $y$  regarded as functions of  $s$  and  $t$ . The data are

$x$	$y$
$\frac{\partial x}{\partial s}$	$\frac{\partial y}{\partial s}$
$\frac{\partial x}{\partial t}$	$\frac{\partial y}{\partial t}$
$\frac{\partial^2 x}{\partial s \partial t}$	$\frac{\partial^2 y}{\partial s \partial t}$

To see to what these quantities refer, consider the point A. The value of  $x$  at point A means its abscissa. The quantity  $\frac{\partial x}{\partial s}$  means the derivative of  $x$  with respect to the parameter  $s$  and is the answer to the question, How rapidly does  $x$  change as  $s$  increases from point A. The quantity  $\frac{\partial x}{\partial t}$  is similar, except with respect to the parameter  $t$ . Finally,  $\frac{\partial^2 x}{\partial s \partial t}$  is the cross derivative. These cross derivatives can be difficult to calculate “in your head” but reasonable meshes can often be generated by taking both  $\frac{\partial^2 x}{\partial s \partial t}$  and  $\frac{\partial^2 y}{\partial s \partial t}$  to be zero.

**Exercise 5:** Generate a script m-file named `exer5.m` with the following commands, replacing the ??? with correct expressions. Running this script should produce a square divided into four rectangles horizontally and two rectangles vertically, as shown here



Please include a copy of your plot with the files you send to me.

**Hints:**

- Read through the entire file before making any changes.
- To see what quantities such as  $dxdsB$  should be, answer the question, “As you move along the line from A to B, how is the coordinate  $x$  varying as you get to B?”
- If you are having trouble getting it to look right, first plot the outline (the lines with  $t = 0$ ,  $t = 1$ ,  $s = 0$ , and  $s = 1$ ). Then look at the corner points one at a time. You likely have some of them right and others wrong. Fix the wrong ones, one at a time. Then add the lines with varying  $s$ , and finally add the lines with varying  $t$ .
- If you are still having trouble getting it to look right, look at the corner points one at a time. You likely have some of them right and others wrong. Fix the wrong ones, one at a time.

`% Generate a mesh based on Hermite bicubic interpolation`

`% values for point A`

```
xA      = 0;      yA      = 0;
dxdsA   = 1;      dydsA   = 0;
dxdtA   = 0;      dydtA   = 1;
d2xdsdtA = 0;     d2ydsdtA = 0;
```

`% values for point B`

```
xB      = 1;      yB      = 0;
dxdsB   = ???     dydsB   = ???
dxdtB   = 0;      dydtB   = 1;
d2xdsdtB = 0;     d2ydsdtB = 0;
```

`% values for point C`

```
xC      = 1;      yC      = 1;
dxdsC   = 1;      dydsC   = 0;
dxdtC   = ???     dydtC   = ???
d2xdsdtC = 0;     d2ydsdtC = 0;
```

`% values for point D`

```
xD      = 0;      yD      = 1;
dxdsD   = 1;      dydsD   = 0;
dxdtD   = 0;      dydtD   = 1;
d2xdsdtD = ???     d2ydsdtD = ???
```

```
% Start off with 5 points horizontally,
% and 3 points vertically
```

```

s=linspace(0,1,5);
t=linspace(0,1,3);

% interpolate x along bottom and top, function of s
xAB =eval_herm([0,1],[xA,xB],[dxdsA,dxdsB],s);
dxdtAB=eval_herm([0,1],[dxdtA,dxdtB],[d2xdsdtA,d2xdsdtB],s);
xDC =???
dxdtDC=???

% interpolate y along bottom and top, function of s
yAB =???
dydtAB=???
yDC =???
dydtDC=eval_herm([0,1],[dydtD,dydtC],[d2ydsdtD,d2ydsdtC],s);

% interpolate s-interpolations in t-direction
% if variables x and y already exist, they might have
% the wrong dimensions. Get rid of them before reusing them.
clear x y
for k=1:length(s)
    x(k,:)=eval_herm([0,1],[xAB(k),xDC(k)],[dxdtAB(k),dxdtDC(k)],t);
    y(k,:)=???
end

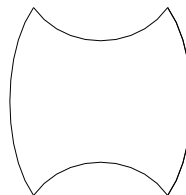
% plot all lines
axis('square');
plot(x(:,1),y(:,1),'b')
hold on
for k=2:length(t)
    plot(x(:,k),y(:,k),'b')
end
for k=1:length(s)
    plot(x(k,:),y(k,:),'b')
end
hold off

```

Just because you can correctly generate a straight-line mesh does not mean your code is correct. In the next exercise, you are going to continue debugging your Matlab programming by testing with two more difficult shapes.

### Exercise 6:

- (a) Consider the following sketch:



Regard each of the four curves as quarter-circles, so the slopes at each corner are  $\frac{dy}{dx} = \pm 1$ . (Do not confuse this with  $\frac{dx}{ds}$  or  $\frac{dy}{ds}$ .) The four corner points are at the corners of the same unit square

you used in Exercise 5. Copy the file `exer5.m` to another file `exer6a.m` and use it to generate an  $21 \times 21$  mesh on this figure. Your mesh should clearly show four-fold symmetry (left-right and top-bottom), and the mesh intervals along the perimeter should be roughly uniform. Please include a plot of your mesh with the files you send to me.

- (b) Copy the file `exer5.m` to a file `exer6b.m` (or you can use “Save as” in the File menu). Change the points so that point C is (1,2) and point B is (3,0). Also, change the number of points for  $s$  to 20 and the number of points for  $t$  to 23. Choose the slopes of the lines so that the *boundary* lines are straight with mesh points uniformly distributed along them. For example, both values  $dxdsD$  and  $dydsD$  should be 1 because both  $x$  and  $y$  increase by 1 as  $s$  goes from 0 to 1 starting at point D and ending at point C. In addition, choose  $d2xdsdt=-2$  and  $d2ydsdt=1$  at all four boundary points. Your mesh lines should all be straight lines. Please include a plot of your mesh with the files you send to me.

## 5 Varying parameters

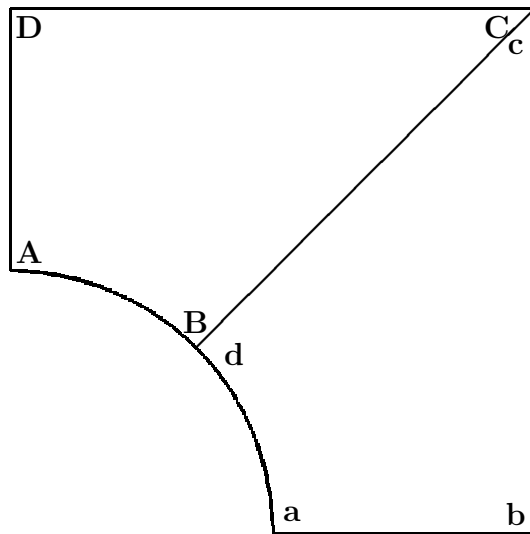
Why should we have used *parametric* interpolation? Once you have the outline of a patch specified, it is possible to change the distribution of the internal mesh lines by changing the parameters ( $s$  and  $t$ ). So long as they vary smoothly and map onto the square  $[0, 1] \times [0, 1]$ , they are essentially arbitrary.

### Exercise 7:

- (a) Copy your `exer5.m` to `exer7.m` and change it to generate 10 rectangular elements horizontally and 15 vertically.
- (b) Modify  $\partial x/\partial s$  at points A and D to be 2.0 and at points B and C to be 0.25. Leave all other values alone. You should see that the mesh elements become much thinner as they get closer to the right side. This is particularly valuable in aeronautical calculations because the “boundary layer” is remarkably thin near an aircraft’s skin and mesh elements must be very thin there. Please send me this plot with your work.

## 6 Matching patches

So far, you have not seen any good reason for using Hermite cubic interpolation over any other method. There really is a good reason for this choice, namely, that two patches can be placed adjacent to each other and, if the derivatives at the endpoints of the common side are given the same values, the mesh will be smooth across the boundary. In the following exercise, there are two patches that touch along one edge. Consider the following sketch.



Point	Coordinates
A	(0, 1)
B	$(\sqrt{2}/2, \sqrt{2}/2)$
C	(2, 2)
D	(0, 2)
a	(1, 0)
b	(2, 0)
c	(2, 2)
d	$(\sqrt{2}/2, \sqrt{2}/2)$

**Exercise 8:** In this exercise, you will generate a mesh in for the two-patch region in the sketch by generating meshes for the two patches separately and seeing that they naturally match up. The inner boundary in the sketch is circular, but we will be using a Hermite cubic approximation, which is pretty good. If more accuracy were needed for this circular boundary, more patches could be used.

(a) Copy the following commands for patch ABCD to a file `exer8.m`.

```
% Generate a mesh for the patch ABCD
sqrt2on2=sqrt(2)/2;

% values for point A
xA      = 0;      yA      = 1;
dxdsA   = 1;      dydsA   = 0;
dxdtA   = 0;      dydtA   = 1;
d2xdsdtA = 0;    d2ydsdtA = 0;

% values for point B
xB      = sqrt2on2; yB      = sqrt2on2;
dxdsB   = sqrt2on2; dydsB   = -sqrt2on2;
dxdtB   = sqrt2on2; dydtB   = sqrt2on2;
d2xdsdtB = 0;    d2ydsdtB = 0;

% values for point C
```

```

xC      = 2;      yC      = 2;
dxdsC   = 2;      dydsC   = 0;
dxdtC   = sqrt(2); dydtC   = sqrt(2);
d2xdsdtC = 0;      d2ydsdtC = 0;

% values for point D
xD      = 0;      yD      = 2;
dxdsD   = 2;      dydsD   = 0;
dxdtD   = 0;      dydtD   = 1;
d2xdsdtD = 0;      d2ydsdtD = 0;

```

- (b) Using your earlier exercises as a model, add appropriate commands to complete `exer8.m` so that it generates and plots a  $20 \times 30$  mesh on patch ABCD.
- (c) Add a second set of commands to `exer8.m` so that it generates a  $20 \times 30$  mesh on patch abcd. Make sure that the number of points along edge BC is the same as the number of points along edge dc, so that the mesh lines are continuous. Your mesh should be symmetric about the line BC (dc). Please include the plot of the combined meshes on these two patches when you send your work to me.
- (d) Hermite cubics will generate smooth mesh lines if the outline of the region is smooth. The outer boundary DCcb has a corner. Make a copy of `exer8.m`, call it `exer8d.m`, and “round the corner off” by moving the points C and c from (2,2) to (1.8,1.8), and making the boundary lines at point C (and c) have slope = -1 (parametric slopes =  $\pm\sqrt{2}/2$ ). Please include this plot when you send me your work.