TANA09 Datatekniska beräkningar

Laboration 3. Interpolation

Namn:_____

Personnummer:_____

Epost:_____

Namn:_____

Personnummer:_____

Epost:_____

Godkänd den:_____ Sign:_____

Retur:_____

1 Introduction

2 Polynomial Interpolation

In the first part of this exercise we will study polynomial interpolation using MATLABs standard routines för polynomials.

An interpolating polynomial is computed using `polyfit`. Given a table of points (t_k, y_k), $k = 1, \ldots, n$, stored in two vectors `t` and `y`, we fit a polynomial of degree $n-1$ to the table using the command

```
>> p=polyfit(t,y,n-1);
```

In the case when the degree of the polynomial is $n-1$, where n is the number of points in the table we get an interpolating polynomial, i.e. $p(t_k) = y_k$. If we select a different degree for the polynomial we instead get a least squares fit $\min \|p(t_k) - y_k\|_2$. We can compute the value of the polynomial for t values stored in a vector `tt` by typing

```
>> yy=polyval(p,tt); plot(tt,yy);
```

Use the functions `polyfit` and `polyval` to solve the next few exercises.

Exercise 2.1 Let $f(t) = \mathrm{e}^{-t/5} + \sin(t)$. We will approximate $f(1.3)$ using linear interpolation and the following table:

t_k	1	1.5	2
$f(t_k)$	1.6602	1.7383	1.5796

How many points $(t_k, f(t_k))$ do we need to create the linear interpolating polynomial $P_1(t)$? Which points should we use?

Answer:_____

Compute the value of the polynomial in the point $t = 1.3$, i.e. $P_1(1.3)$.

Answer:_____

The MATLAB routine `polyfit` does not provide an estimate of the truncation error. Instead we make use of the fact that $P_2(t) \approx P_1(t) + R_T(t)$. Thus we compute a polynomial $P_2(t)$ that interpolates $f(t)$ in all three points in the above table, and estimate the truncation error when approximating $f(t)$ by the straight line $P_1(t)$ by $R_T(t) \approx P_2(t) - P_1(t)$.

Exercise 2.2 Do an error analysis and estimate the error $|f(1.3) - P_1(1.3)|$. The values given in the table are assumed to be correctly rounded. Present your calculations. Also compare your estimate with the actual error.

$|R_T| \leq$

$|R_{XF}| \leq$

$|R_B| \leq$

$|R_{TOT}| \leq$

Answer:

2.1 Runge's phenomena

In this exercise we shall study what happens when we try to interpolate the function

$$f(x) = \frac{1}{1 + 25x^2}$$

using a high degree polynomial. The program `RungesPhenomena.m` can be used for this investigation. By typing, e.g.,

```
>> RungesPhenomena(6);
```

and an interpolating polynomial of degree $n = 6$, that interpolates $f(x)$ in equidistant points in the interval $[-1, 1]$ is computed and plotted.

Exercise 2.3 Try to interpolate the function $f(x)$ using polynomials of degree $n = 4, 6, 8$, etc. Do we get a better, or worse, approximation if a higher degree polynomial is used?

Answer:

Exercise 2.4 Which conclusion should we draw from this experiment?

Svar:

3 Spline interpolation

In MATLAB the routine `csape` can be used to compute interpolating cubic splines. In order to compute the cubic spline, with *natural boundary conditions*, that interpolates the table (t_k, y_k), $k = 1, \ldots, n$, stored as two vectors `t` och `y`, we write

```
>> pn=csape(t,y,'variational');
```

If we want to use *correct boundary conditions* we instead write

```
>> pr=csape(t,y,'complete',[yprima,yprimb]);
```

where `yprima` and `yprimb` are numerical values for the derivatives $y'(t_1)$ and $y'(t_n)$ respectively.

In order to compute the function values we can use `fnval` or `ppval`. As an example we can plot the spline on an interval $[a, b]$ by typing

```
>> tt=a:0.01:b;
>> yy=fnval(pn,tt);
>> plot(tt,yy);
```

3.1 A mechanical Arm

If an industrial robot is used to move a tool often the path is specified as a cubic spline. In our case the arm should follow a cubic spline $s(t)$ that interpolates the values in the following table:

t	0	2	4	5	6
y	0	0	1	0	0

Exercise 3.1 Create two vectors `t` and `y` that contains the values from the table. Use `csape` to generate the **natural** cubic spline $s(t)$ that interpolates the given function values. Plot the spline curve and and also mark the points (t_k, y_k) from the table with `*` in the graph.

Finally use `fnval` to compute the values of the spline function in a couple of additional points. Give numeric values for $s(1)$ and $s(3)$?

Answer:_____

Exercise 3.2 From the table we see that appropriate boundary conditions for the cubic spline are $s'(0) = 0$ and $s'(6) = 0$. Again use `csape` and compute the cubic spline that interpolates the above table, but with **correct** boundary conditions. Plot the new spline curve in the same graph as the previous curve. Use *hold on*.

Again use `fnval` and evauate the spline curve for $t = 1$ and $t = 3$. Give numeric values for $s(1)$ and $s(3)$?

Answer:_____

The spline function changes as a result of changing the boundary conditions. Where is the changes most visible? Near the end points or in the middle of the interval?

Answer:_____

Report the exercises 3.1 and 3.2 by printing the graph when you are done.

3.2 The Truncation Error

From the theory we know that the truncation error when using an interpolating cubic spline is

$$R_T = f(x) - s_h(x) \approx c \cdot h^p,$$

where c is a constant, and p is an integer. In this exercise we will experimentally determine the *order of accuracy* p, when using *correct boundary conditions*.

We will use `csape` to interpolate the function

$$f(x) = \frac{4}{3}x^4 - \frac{4}{3}x^3 + \frac{1}{2}x^2.$$

with a cubic spline. Since the function is known we can compute the error as a function of the step size h. By investigating how the error depends on the step size h we can compute the order of accuracy, i.e. the parameter p above.

The MATLAB program `SplineError.m` computes the interpolating cubic spline and also plots both the spline and the error curve $|s_h(x) - f(x)|$, $0 < x < 1$. Study the code. The values for a couple of parameters are left out. You need to fill in the correct values or the program will not work properly.

Exercise 3.3 (Preparation) Calculate $f'(0)$ and $f'(1)$.

Answer:_____

Exercise 3.4 Make the nessecary changes in the program `SplineError.m` and enter the numerical values for $f'(0)$ and $f'(1)$ respectively. Finally use the program to interpolate $f(x)$ using a cubic splines with different step sizes h. Use $h_1 = 1/4$, $h_2 = 1/8$. Write down the maximum error for $0 < x < 1$ in the table below.

h	$1/4$	$1/8$
$\lvert R_T(h) \rvert$		

Exercise 3.5 (**Preparation**) We will now make use of the fact that $R_T(h) \approx c \cdot h^p$ and derive a formula for the order of accuracy p. Insert the expression for the truncation error in the formula below and simplify:

$$\frac{|R_T(h_1)|}{|R_T(h_2)|} \approx$$

Exercise 3.6 Compute the following

$2^p =$_____ Experimental value: $p =$_____ (**integer!**)

Exercise 3.7 Write down the theoretical error estimate for cubic spline interpolation. For which type of boundary conditions is the error estimate derived? Does the theoretical result agree with the practical experiment?

Svar:_____

4 Parametric Splines for Distance Calculations

Most mapping applications will tell you the distance you need to travel before you reach your destination. In this exercise we will investigate one possible method for calculating the distance between two points on the map. The method is practical in the sense that it requires very little additional information stored on the map and is computationally efficient.

The basic outline of the method is to start by collecting a set of points (x_k, y_k) along the travel path. The points are used to find an interpolating cubic spline such that

$$s(t_k) = (s_x(t_k), s_y(t_k))^T = (x_k, y_k)^T,$$

and $s(0)$ is the point of origin and $s(1)$ is the destination. The points need to be selected in such a way that the resulting spline curve is a good approximation of the travel path. The distance we need to travel is calculated by calculating the length of the resulting spline curve.

Exercise 4.1 The function `DisplayMap()` shows an image containing a map over Linköping. On the map the locations of the church in Slaka and the Sibylla kiosk at T1 are marked. The goal of the exercise is to measure the distance we need to travel to get from the church to the hamburger kiosk.

The function `AddPointsToMap` lets you add more and more points to the map. First decide which roads to travel and then add points on the map. Add points more closely together if the road turns sharply. Initially just add a couple of points. If you already have a couple of points marked you can add, e.g. 5 additional points by typing,

```
>> Points = [ Points ; AddPointsToMap(5) ];
```

Note that the points have to be added in order.

Exercise 4.2 Suppose two vectors `x` and `y` containing points (x_k, y_k) on the map are available. Write a function that computes a parametric spline curve $s(t) = (s_x(t), s_y(t))^T$ that interpolates the points $(x_k, y_k)^T$. The function should be used as follows,

```
>> [ sx , sy]=ParametricCurve( x , y );
```

Use the Matlab function `csaps` to create the two component functions `sx` and `sy`. The parameter range should be $0 \leq t \leq 1$.

Besides computing the spline curve it is a good idea to also plot the curve in a new, or existing, graphics window. To do the plot assume that it is enogh to evaluate the spline curve for 10 points between each coordinate marked on the map. This means you should create a vector with t-values,

```
>>  n=10*length(x); tt=(0:n)/n;
```

and evaluate the spline functions `sx` and `sy` for the parameter values stored in the vector `tt` using `ppval`.

Go back and add points along the path again until you have a spline curve that follows the road accurately. How many points did you add?

Answer:

Exercise 4.3 Implement a function `CalculateLength(sx,sy,N)`. Use the folllowing method: Pick a number N and create a parameter vector

```
>> tt=(0:N)/N;
```

evaluate points $(s_x(t_k), s_y(t_k))$ on the curve. The length of the curve is approximated by

$$L_N = \sum_{k=1}^{N} \| \begin{pmatrix} s_x(t_k) - s_x(t_{k-1}) \\ s_y(t_k) - s_y(t_{k-1}) \end{pmatrix} \|_2.$$

The scale of the map is such that a length $L = 96 \approx 1\,km$. Use your function to calculate the distance between Slaka and the Sibylla kiosk. Use $N = 100$. Print and handin the map with the spline curve added.

Answer:

Exercise 4.4 This type of integration method is supposed to have an error $C \cdot (\Delta t)^2$. Verify this by computing the curve length using $N = 100, 200$, and 400. Use $N = 5000$ to get a good approximation of the *exact* distance.

N	100	200	400
$\lvert L_N - L_e \rvert$			

Do the results confirm that the error is proportional to $(\Delta t)^2$?

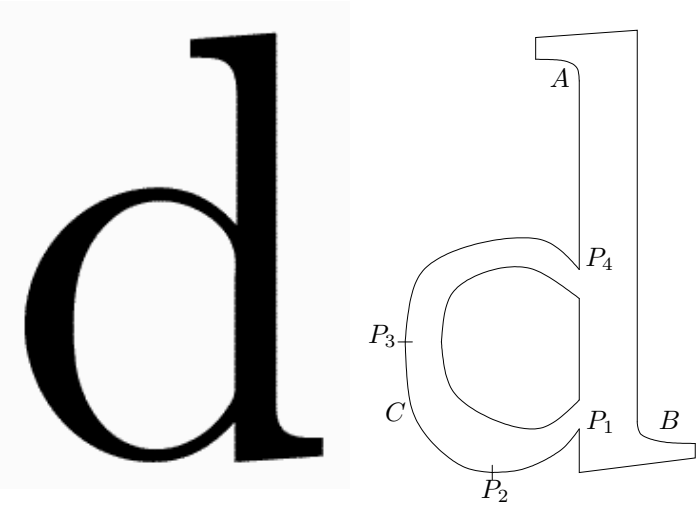
Answer:

Remark This method is realistic to use for a wide range of applications where it is important to plot the path, and estimnate the distance, travelled. It is most efficient for cases where the roads are mostly straight with low curvature. The points needed can be stored for each road segment.

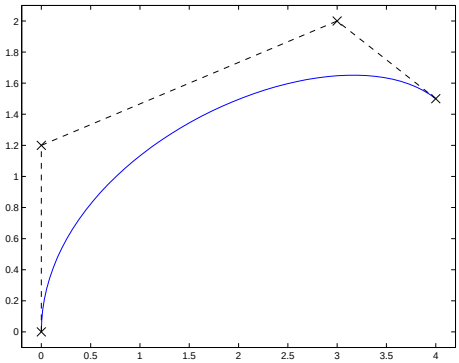
Since the curve $s(x)$ is a piecewise polynomial we can compute derivatives analytically. Hence we can calculate the tangent vector $s'(t)$ for each polynomial segment. The length of the tangent vector $\|s'(t)\|_2^2$ is also a polynomial. Hence, with a little analytical work, we can actually calculate the length of the curve by integrating a polynomial function. The MATLAB function `fnder` computes the needed derivatives. The needed products can be computed using convolutions, i.e. `conv`, and finally the arc length integral using `fnint`. The details are beyond the scope of a 2 hour exercise though.

5 Beziér Curves and Font Specification

In several applications one needs to specify the shape of an object. One such application is type setting text for printing. The individual letters making up the text have standardized shapes. It is desirable to have a font that is independent of the resolution used for the printing. Hence the shapes are described in such a way that we can easily generate pixel images showing the letters in any resolution. Designing a font package means having an artist draw the desired shape of each letter. Each curve segment is then described mathematically in such a way that images can be easily generated. The full procedure is best described in *The Metafont book*, by Donald E. Knuth (author of the TeX typesetting language). In this exercise we will use the standard technique for specifying curves in font packages and design the letter d.



Figur 1: The letter d generated from a vectorized font. The outline of the letter is made up from Beziér curves and line segments. The interpolation points used to create the outer arc are displayed.



Exercise 5.1 A cubic Beziér curve patch is defined using 4 *control points* $P_1, \ldots, P_4$. In the local coordinate t the definition is

$$p(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t) t^2 p_3 + t^3 P_4, \qquad 0 \leq t \leq 1.$$

Write a MATLAB function `DisplayBezierPatch`, with the 4 control points and a parameter N as input. The function should create a vector `t=(0:N)'/N` and evaluate the Beziér curve for those values of the parameter t.

Finally the function should plot both the curve and the control points. For the following exercises it is useful to make the result look like the above example. The shape of the curve can be changed by moving the control points.

Exercise 5.2 The MATLAB program `DrawFont` plots the line segments in the letter. In the next few exercises we will add the missing smooth curve segments. Open the function `DrawFont` in the Matlab editor. First we will add code to draw the curve segment A. Denote by P_1 and P_2 the starting and ending points for the curve segment. We need two additional control points to specify the Beziér curve.

At start the curve tangent of the curve should be vertical. What condition on the control point guarantee that this is the case?

Answer:

At the end point the tangent of the curve segment should be horizontal. Write down the coordinates for the 4 points that define the Beziér curve segment. Add the code to plot the curve segment. Is the result good?

Answer:

Exercise 5.3 (optional) Similarily find two control points to use, together with the two points P_1 and P_2, to draw the curve segment B as a Beziér curve. Write down the coordinates for the points you use.

Answer:

Exercise 5.4 Now we will create the Beziér curves that make up the outer arc C in the letter. The curve should start at a point P_1 and end in a point P_4. Since we want the curve to touch the line $y=0$ we pick an interpolation point $P_2 = (x_2, 0)$. Experiment a little to find a suitable value x_2 and write down the interpolation point. Also we want the curve to have a vertical tanget at some point. This is the next interpolation point P_3. Write down your choice for P_3 as well.

Answer:

For each of the Beziér curves $P_1 \mapsto P_2$, $P_2 \mapsto P_3$, and $P_3 \mapsto P_4$ we need two additional control points. Make sure to pick something that looks good, and also so that the tangent is horizontal at P_2 and vertical at P_3. Add the code to draw the curve segments in the script. Does the result look nice?

Answer:

Exercise 5.5 (optional) Finally we will add the curve segments that make up the inner arc in the letter. It is enough to use two Beziér curves so specify an interpolation point P_2 where the tangent of the curve is vertical. The inner curve will be made up of Beziér curves $P_1 \mapsto P_2$, and $P_2 \mapsto P_3$. Pick good control points and add the code to draw the curves. Write down the coordinates for the interpolation points you use. Is the result good?

Answer:

Report by plotting the handing in the graph with the curves defining your letter. It may look better if you use `axis image` before the plot.

Exercise 5.6 (optional) Alternate font styles, such as bold face or italic fonts, are generated by mathematical transformations on the points used to describe the basic curve shapes. An italic style font is tilted slightly. This is achived by a linear transformation $(x, y) \mapsto (x + \alpha y, y)$. In Matlab the interpolation points can be transformed by, e.g.,

```
>> Points = [ 1,0.07 ; 0 1]*Points;
```

which tilts the letter slightly. The code for doing this transformation is prepared into the script `DrawFont`. You need to make the same transformation on all the additional interpolation and control points you created during the exercise. The result should be a nice italic style letter d. Did it work?

Answer:

Remarks Beziér curves are the standard tool for designing smooth curves in many branches of computer graphics. The methods are easily extended to making mathematical representations of smooth surfaces. Most graphics systems, e.g. `OpenGL` or the `Postscript` language, has support for drawing various types of splines.