Laboratory of Advanced Numerical Analysis

Marco Caliari

a.a. 2012–2013

These lecture notes are strongly under development. They are partially based on the work done by Dr. Manolo Venturin in the past years.

Contents

Chapter 1

Wavelets

1.1 The Haar wavelet

We implement in Matlab®

$$s(t) = \sum_k s_k \phi(t-k) \tag{1.1}$$

where

$$\phi(t) = \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

We assume the sum in (1.1) is finite, therefore

$$\begin{aligned} s(t) &= \sum_{k=k_{\min}}^{k_{\max}} s_k \phi(t-k) = \sum_{j=1}^{k_{\max}-k_{\min}+1} s_{k_{\min}-1+j} \phi(t-(k_{\min}-1+j)) = \\ &= \sum_{j=1}^{k_{\max}-k_{\min}+1} r_j \phi(t-(k_{\min}-1+j)) \end{aligned} \tag{1.2}$$

1.1.1 The trend

The *trend* is defined as

$$T(t) = \sum_h \frac{s_{2h} + s_{2h+1}}{2} \phi(t/2-h)$$

Given the previous $k_{\min}$ and $k_{\max}$, we have to find the indexes involved in the sum above. We need $2h_{\min} \ge k_{\min}$ and $2h_{\max}+1 \le k_{\max}$. If $k_{\min}$ is odd, we

get $h_{\min} = \lceil k_{\min}/2 \rceil = (k_{\min}+1)/2$. But in the definition of the trend, the coefficient relative to $h_{\min}-1$ is

$$\frac{s_{2(h_{\min}-1)} + s_{2(h_{\min}-1)+1}}{2} = \frac{s_{k_{\min}-1} + s_{k_{\min}}}{2} = \frac{s_{k_{\min}}}{2} \neq 0$$

Therefore, we cannot neglect the index $h_{\min}-1$ and we cannot take $h_{\min} = \lfloor k_{\min}/2 \rfloor$, otherwise we have to access index 0 in the vector `r`. A similar consideration holds for $h_{\max}$. The simplest way to overcome this problem is to modify the values $k_{\min}$ and $k_{\max}$ in the following way: if $k_{\min}$ is odd, then $k_{\min} = k_{\min}-1$ and $s_{k_{\min}} = 0$; if $k_{\max}$ is even, then $k_{\max} = k_{\max}+1$, $s_{k_{\max}} = 0$. Finally, $h_{\min} = k_{\min}/2$ and $h_{\max} = (k_{\max}-1)/2$ and

$$\begin{aligned}
T(t) &= \sum_{h=h_{\min}}^{h_{\max}} \frac{s_{2h} + s_{2h+1}}{2}\phi(t/2-h) = \\
&= \sum_{i=1}^{h_{\max}-h_{\min}+1} \frac{s_{2(h_{\min}-1+i)} + s_{2(h_{\min}-1+i)+1}}{2}\phi(t/2-(h_{\min}-1+i)) = \\
&= \sum_{i=1}^{h_{\max}-h_{\min}+1} \frac{s_{k_{\min}-1+(2i-1)} + s_{k_{\min}-1+2i}}{2}\phi(t/2-(h_{\min}-1+i)) = \\
&= \sum_{i=1}^{h_{\max}-h_{\min}+1} \frac{r_{2i-1} + r_{2i}}{2}\phi(t/2-(h_{\min}-1+i)) = \\
&= \sum_{i=1}^{h_{\max}-h_{\min}+1} u_i\phi(t/2-(h_{\min}-1+i))
\end{aligned}$$

Therefore, $T(t)$ is a signal defined by $h_{\min}$ and the vector u, with breaks at 2x integers.

1.1.2 The detail

The *detail* is defined as $D(t) = s(t) - T(t)$. It can be proven that

$$D(t) = \sum_{h} \frac{s_{2h} - s_{2h+1}}{2}\psi(t/2-h) = \sum_{i=1}^{h_{\max}-h_{\min}+1} v_i\psi(t/2-(h_{\min}-1+i)) \quad (1.3)$$

where $v_i = (r_{2i-1} - r_{2i})/2$ and $\psi(t) = \phi(2t) - \phi(2t-1)$.

1.2 Haar decomposition

The signal $s(t)$ can be decomposed into l levels as

$$s(t) = T_1(t) + D_1(t) = (T_2(t) + D_2(t)) + D_1(t) = \ldots = \\ = (T_l(t) + D_l(t)) + D_{l-1}(t) \ldots + D_2(t) + D_1(t)$$

Notice that it is easy to write the signal as the sum, but in general not to compute the amplitudes of the signal given its decomposition.

In order to decompose a signal, we could make use of a structure `S` with fields:

- `type`: signals are of type `trend` (they need ϕ functions to be computed) or `detail` (they need ψ functions to be computed);
- `breaks`: an integer number l such that breaks are at 2^{l-1}x integers;
- `kmin`: smallest index;
- `amplitudes`: array containing the coefficients.

1.3 Images

We will consider only gray-scale images. A gray-scale image is a matrix of order $m \times n$ (m is the height of the image in pixels and n the width) with integer entries in $[0, 255]$ (0 is black and 255 is white). The command to load an image and check the result is

```
>> a = imread('arena.png');
>> size(a)

ans =

         512        1024

>> colormap(gray(256));
>> image(a)
```

In order to apply our tools we need m and n powers of 2 (see `imresize`).

Notice that `a` is a matrix of unsigned 8-bit integers (`uint8`). When computing the trend and the detail, we will use `double` (standard) numbers.

The trend can be stored into a $m/2 \times n/2$ matrix. The detail into three $m/2 \times n/2$ matrices. In fact, a square

$$\begin{array}{|c|c|}\hline a-\mu & b-\mu \\ \hline c-\mu & d-\mu \\ \hline \end{array}$$

where μ is the average of a, b, c, d, can be represented by

$$A \cdot \begin{array}{|c|c|}\hline 1 & -1 \\ \hline 1 & -1 \\ \hline \end{array} + B \cdot \begin{array}{|c|c|}\hline 1 & 1 \\ \hline -1 & -1 \\ \hline \end{array} + C \cdot \begin{array}{|c|c|}\hline 1 & -1 \\ \hline -1 & 1 \\ \hline \end{array}$$

where

$$A = \frac{a-b+c-d}{4} \quad (x \text{ details})$$
$$B = \frac{a+b-c-d}{4} \quad (y \text{ details})$$
$$C = \frac{a+b-c-d}{4} \quad (xy \text{ details})$$

1.3.1 Colormaps

Another good color-map for gray-scale images is `pink`. Since the detail images contain negative values, the images appear quite dark. In order to brighten them it is possible to use

```
colormap(gray(256).^(1-beta))
```

where `beta` is a number between 0 and 1. Conversely, to darken an image you can use

```
colormap(gray(256).^1/(1-beta))
```

See also the command `brighten`.

1.4 Exercises

`mkhaar.m`
`haarplot.m`

1. Write a function which computes $s(t)$, $T(t)$ and $D(t)$, given $k_{\min}$, an array r of coefficients of length $k_{\max} - k_{\min} + 1$ and an array t of independent variables. Test it to plot $s(t)$, $T(t)$ and $D(t)$ for

$$\begin{aligned} \left[s_2, s_3, s_5, s_8\right] &= \left[1, 2, -1, -4\right] \\ \left[s_1, s_3, s_5, s_8, s_{10}\right] &= \left[1, 2, -1, -4, -2\right] \\ \left[s_{-2}, s_1, s_5\right] &= \left[1, 2, -1\right] \\ \left[s_{-1}, s_3, s_5, s_8, s_9\right] &= \left[1, 2, -1, -4, -5\right] \end{aligned}$$

2? Compute the detail $D(t)$ as $s(t) - T(t)$ and as in formula (1.3) and check that they coincide. `haar.m`

3 Write a function which decomposes a signal. The decomposition should be stored into a structure `dec` with fields `dec.T` (the cell array of the trends) and `dec.D` (the cell array of the details). Then write a function which decomposes a signal up to level l. `haar.m` `haardec.m`

4 Load an image $m \times n$ (powers of 2) with `imread` and compute its trend and detail. The trend image should have size $m/2 \times n/2$. Show that the original image is the sum of its trend and its detail. `haarcomp.m` `imagetrenddetail.m` `zoomtrend.m`

5 Show the decomposition of the detail into A, B and C. `showdec.m` `ABC2detail.m`

Chapter 2

Splines

2.1 Piecewise polynomials

Let us consider a sequence of nodes $x_1 < x_2 < \ldots < x_n$ and a piecewise polynomial which on the interval $[x_i, x_{i+1}]$ is

$$a_{i,1}(x - x_i)^{j-1} + a_{i,2}(x - x_i)^{j-2} + \ldots + a_{i,j-1}(x - x_i) + a_{i,j}$$

Matlab® handles piecewise polynomials through the `pp` structures. They contain the following fields:

- `form`: `pp`;
- `breaks`: interpolation nodes $\{x_i\}$, as a row vector;
- `coefs`: coefficients stored in a matrix $A = (a_{i,j})$ of size $(n-1) \times j$;
- `pieces`: number of intervals, $n - 1$;
- `order`: j;
- `dim`: number of sets of values related to the breaks.

Given the interpolation nodes and coefficient matrix it is possible to assemble the structure `pp` through the command `mkpp`. Given a structure `pp` it is possible to extract the nodes and the coefficient matrix through the command `unmkpp`.

2.2 Splines of degree 1

Splines of degree 1 are quite easy to implement in this way: in fact

$$a_{i,1} = \frac{y_{i+1} - y_i}{h_i}$$
$$a_{i,2} = y_i$$

with $h_i = x_{i+1} - x_i$ and it is enough to use the command `mkpp`. Once the structure `pp` is created, it is possible to evaluate the piecewise polynomial on a arbitrary set of points through the command `ppval`.

2.2.1 Compression

It is possible to compress a set of values $\{y_i\}$ corresponding to $\{x_i\}$, that is to extract a subset of $\{x_i\}$ for which its linear interpolant differs less than a prescribed tolerance from the original values $\{y_i\}$. We proceed in this way: x_1 has always to be kept and we take the linear interpolant between x_1 and x_3 and measure the error between its values at internal nodes (actually only x_2) and the corresponding y's: if it is below to a prescribed tolerance, replace x_3 with x_4 and repeat. As soon as the linear interpolant between x_1 and x_m fails the criterion, x_{m-1} has to be kept. We restart the procedure from x_{m-1}. We stop as soon as we reach the last original node.

2.3 Quadratic splines

Let us consider a quadratic spline $S(x)$ defined at the *sites* (i.e., interpolation points) $\{x_i\}$. In order to construct its coefficients, we can consider its first derivative in the interval $[x_i, x_{i+1}]$

$$S'_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{h_i}(x - x_i) + m_i, \quad i = 1, 2, \ldots, n-1$$

where $m_i = S'(x_i)$ are the unknowns. By integration we get

$$S_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{2h_i}(x - x_i)^2 + m_i(x - x_i) + a_i$$

We require the interpolation property $S_{[x_i,x_{i+1}]}(x_j) = y_j$, $j = i, i+1$

$$y_i = a_i$$
$$y_{i+1} = (m_{i+1} - m_i)\frac{h_i}{2} + m_i h_i + y_i$$

and we get the linear system

$$\frac{m_i}{2} + \frac{m_{i+1}}{2} = \frac{y_{i+1} - y_i}{h_i}, \quad i = 1, 2, \ldots, n-1$$

The missing condition is usually prescribed on m_1 or m_n. Due to this asymmetry in the conditions, sometimes quadratic splines show an oscillatory behavior.

2.3.1 A note about periodic quadratic splines

Let us try to construct a periodic quadratic spline. It means that $y_1 = y_n$ ans we impose $m_1 = m_n$. The arising system is

$$\begin{bmatrix} 1 & 0 & \ldots & 0 & -1 \\ 1 & 1 & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 1 & 1 & 0 \\ 0 & \ldots & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ \vdots \\ m_{n-1} \\ m_n \end{bmatrix} = 2 \begin{bmatrix} 0 \\ \frac{y_2 - y_1}{h_1} \\ \vdots \\ \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \\ \frac{y_n - y_{n-1}}{h_{n-1}} \end{bmatrix}$$

If n is odd, the rank of the matrix is $n-1$ (the first row is the sum, with alternate sign, of the following). Therefore, a solution exists only if the rank of the complete system is $n-1$, too. In that case, given a solution $\{m_i\}$, $\{m_i + (-1)^i k\}$ is a solution, too.

2.3.2 Quadratic splines with knots

Another way to proceed is the following: given the set $\{x_i\}$, we consider the intervals given by the *knots* $\{t_j\}$, where

$$\begin{aligned} t_1 &= x_1 - \frac{h_1}{2} \\ t_j &= \frac{x_{j-1} + x_j}{2}, \quad j = 2, 3, \ldots, n \\ t_{n+1} &= x_n + \frac{h_{n-1}}{2} \end{aligned}$$

and the quadratic spline $S(t)$ defined by

$$\begin{aligned} S'_{[t_j, t_{j+1}]}(t) &= \frac{m_{j+1} - m_j}{k_j}(t - t_j) + m_j, && j = 1, 2, \ldots, n \\ S_{[t_j, t_{j+1}]}(t) &= \frac{m_{j+1} - m_j}{2k_j}(t - t_j)^2 + m_j(t - t_j) + a_j, && j = 1, 2, \ldots, n \end{aligned}$$

First of all, with this choice we have automatically the continuity of first derivatives at internal knots. Moreover, we require interpolation conditions at $\{x_i\}$

$$S_{[t_j,t_{j+1}]}(x_j) = \frac{m_{j+1}-m_j}{8}k_j + m_j\frac{k_j}{2} + a_j = y_j, \quad j = 1, 2, \ldots, n$$

from which

$$a_j = y_j - \frac{m_{j+1}-m_j}{8}k_j - m_j\frac{k_j}{2} = y_j - \frac{3}{8}k_j m_j - \frac{1}{8}k_j m_{j+1}$$

and finally continuity at internal knots $S_{[t_{j-1},t_j]}(t_j) = S_{[t_j,t_{j+1}]}(t_j)$, $j = 2, 3, \ldots, n$ from which

$$\frac{k_{j-1}}{8}m_{j-1} + \frac{3}{8}(k_{j-1}+k_j)m_j + \frac{k_j}{8}m_{j+1} = y_j - y_{j-1}, \quad j = 2, 3, \ldots, n$$

Two other conditions are missing, and we can prescribed the values of m_1 *and* m_{n+1}, or the value of the second derivative at x_1 and x_n, or the continuity of second derivative at the second and the second-last knot (we will call such a quadratic spline *not-a-knot quadratic spline*), or the periodicity of first and second derivative, or finally we can prescribe the values at t_1 and t_{n+1}. For instance, if we consider the not-a-knot quadratic spline, from $S''_{[t_1,t_2]} = S''_{[t_2,t_3]}$ we get $-k_2m_1 + (k_1+k_2)m_2 - k_1m_3 = 0$ and similarly $-k_nm_{n-1} + (k_{n-1} + k_n)m_n - k_{n-1}m_{n+1} = 0$.

2.4 Cubic splines

Let us consider a cubic spline $S(x)$ defined at the sites $\{x_i\}$. In order to construct its coefficients, we can consider its second derivative in the interval $[x_i, x_{i+1}]$

$$S''_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1}-m_i}{h_i}(x-x_i) + m_i, \quad i = 1, 2, \ldots, n-1$$

where $m_i = S''(x_i)$ are the unknowns. By integration we get

$$S'_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1}-m_i}{2h_i}(x-x_i)^2 + m_i(x-x_i) + a_i$$
$$S_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1}-m_i}{6h_i}(x-x_i)^3 + \frac{m_i}{2}(x-x_i)^2 + a_i(x-x_i) + b_i$$

We require the interpolation property $S_{[x_i,x_{i+1}]}(x_j) = y_j$, $j = i, i+1$

$$\begin{aligned} b_i &= y_i \\ a_i &= \frac{y_{i+1}-y_i}{h_i} - (m_{i+1}-m_i)\frac{h_i}{6} - m_i\frac{h_i}{2} = \\ &= \frac{y_{i+1}-y_i}{h_i} - m_{i+1}\frac{h_i}{6} - m_i\frac{h_i}{3} \end{aligned}$$

and finally we require the continuity of the first derivative $S'_{[x_{i-1},x_i]}(x_i) = S'_{[x_i,x_{i+1}]}(x_i)$, $i = 2, 3, \ldots, n-1$, from which

$$\frac{h_{i-1}}{6}m_{i-1} + \frac{h_{i-1}+h_i}{3}m_i + \frac{h_i}{6}m_{i+1} = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \tag{2.1}$$

Two other conditions are missing and they depend on the type of the cubic spline. System (2.1) is tridiagonal with dominant diagonal: Thomas' algorithm can be used to solve it, with cost $\mathcal{O}(n)$.

2.4.1 Cubic spline interpolation of a curve

Given a parametric curve $\gamma(t) = (x(t), y(t))$, we may want to interpolate it at some sites $\{t_i\}$ by splines. It is just a double interpolation of the functions $(t, x(t))$ and $(t, y(t))$.

2.4.2 Cubic splines in Matlab®

We use the command `csape`.

2.4.3 Smoothing cubic splines

The Matlab® command `csaps` computes the `pp` form of the cubic spline $S(x)$ minimizing

$$(1-p)\sum_{i=1}^{m} w_i \left|y_i - S(x_i)\right|^2 + p\int_{x_1}^{x_m} S''(x)\mathrm{d}x$$

where $\{x_i\}_i$ is the set of sites and $\{y_i\}$ the set of values, possibly affected by a noise. The weights $\{w_i\}$, if not specified, are chosen equal to 1. If $p = 0$ the result is the least-squares straight line and if $p = 1$ the result is the natural cubic spline interpolating the values. If p is not supplied, a default value is chosen.

2.4.4 Thin-plate smoothing splines

In Matlab®, for a set $\{(x_i, y_i)\}_{i=1}^n$, two-dimensional thin-plate smoothing splines are defined as the function

$$S(x,y) = \sum_{i=1}^{n} c_i \phi(\sqrt{(x-x_i)^2 + (y-y_i)^2}) + c_{n+1}x + c_{n+2}y + c_{n+3} \qquad (2.2)$$

where

$$\phi(r) = r^2 \log(r^2)$$

which minimizes

$$(1-p)\sum_{i=1}^{n} |z_i - S(x_i,y_i)|^2 + p \iint \left|\frac{\partial^2 S}{\partial x^2}(x,y)\right|^2 + 2\left|\frac{\partial^2 S}{\partial x \partial y}(x,y)\right| + \left|\frac{\partial^2 S}{\partial y^2}(x,y)\right| \mathrm{d}x\mathrm{d}y$$

They are computed by the command `tpaps` (p an optional parameter) which returns a `st` form (which can be evaluated by `stval`).

A note on thin-plate spline evaluation

The result of `tpaps(xy,z)`, where `xy` is a $2 \times n$ matrix (`xy(1,i)` $= x_i$, `xy(2,i)` $= y_i$ and `z` a vector of values (`z(i)` $= z_i$) is a structure containing the fields

- `form`: `st-tp00`
- `centers`: the set $\{(x_i, y_i)\}_{i=1}^n$;
- `coefs`: the set $\{c_i\}_{i=1}^{n+3}$;
- `ncenters`: n;
- `number`: $n+3$;
- `dim`: number of sets of values
- `interv`: the cell array

```
{[min(x(1,:)),max(x(1,:))],[min(x(2,:)),max(x(2,:))]}
```

If we want evaluate (2.2) at a grid of values `[X,Y] = ndgrid(xx,yy)`, either we use `stval(st,{xx,yy})` (the easy way) or we use the following code

```
XY = [X(:)';Y(:)'];
DM = zeros(size(xy,2),size(XY,2));
for i = 1:2
  [xyd,XYd] = ndgrid(xy(i,:),XY(i,:));
  DM = DM+(XYd-xyd).^2;
end
% DM is the squared distance matrix
[centers,coefs] = stbrk(st); % get centers and coefs
n = size(centers,2);
vals = coefs(1:n)*phi(DM); % phi part, phi(r) = r*log(r)
vals = vals+coefs(n+1)*XY(1,:)+coefs(n+2)*XY(2,:)+...
       coefs(n+3); % linear part
```

In order to plot the result we can `tpss.m`

```
mesh(X,Y,reshape(vals,size(X)))
```

The distance matrix may be quite large (it is a full matrix): it could be more convenient to evaluate the thin-plate spline on sub-matrices.

2.5 Exercises

1. Write a function `spline1` which given a set of nodes `x` and a set of values `y` computes the structure `pp` associated to spline of degree 1. If an optional argument `xx` is given, it should evaluate the computed structure at `xx`. `spline1.m`

2. Write a function `compress` which given a set of nodes `x`, a set of values `y` and a tolerance `tol` perform a compression as described above. The function should also give the maximum interpolation error. `compress.m`

3. Complete system (2.1) with the missing conditions for any type of cubic spline. Keep the system tridiagonal.

4. Consider n couples of equispaced sites on the quarter of circle of equation $y = \sqrt{1-x^2}$, $x \geq 0$, from $(x_1, y_1) = (0, 1)$ to $(x_n, y_n) = (1, 0)$. `curve.m`

   - draw the cubic natural spline $S(x)$ through the sites;
   - draw the cubic piecewise interpolant $P(x)$ given by `pchip`
   - verify that

$$\int_0^1 |S''(x)|^2 \mathrm{d}x \leq \int_0^1 |P''(x)|^2 \mathrm{d}x$$

- draw the natural spline which interpolates the parametric curve $\gamma(t) = (\cos(\pi/2 - t), \sin(\pi/2 - t))$, $0 \leq t \leq \pi/2$ at n equispaced sites $\{t_i\}$.

Do the points above for $n = 4, 5, \ldots, 10$.

`weightedspline.m`

5? We consider

$$S_{[x_i,x_{i+1}]}(x) = a_i \frac{(x - x_i)^3}{h_i^2} + b_i \frac{(x - x_i)^2}{h_i} + m_i(x - x_i) + y_i$$

Clearly we have

$$S_{[x_i,x_{i+1}]}(x_i) = y_i$$
$$S'_{[x_i,x_{i+1}]}(x_i) = m_i$$

(a) find a_i and b_i such that

$$S_{[x_i,x_{i+1}]}(x_{i+1}) = y_{i+1}$$
$$S'_{[x_i,x_{i+1}]}(x_{i+1}) = m_{i+1}$$

(b) compute $S''_{[x_i,x_{i+1}]}(x)$;

(c) impose the optimality condition

$$w_{i-1} S''_{[x_{i-1},x_i]}(x_i) - w_i S''_{[x_i,x_{i+1}]}(x_i) = 0, \quad i = 1, 2, \ldots, n$$

where $w_0 = w_n = 0$ and get the tridiagonal sparse linear system for $\{m_i\}_{i=1}^n$;

(d) implement a function `weightedspline` taking `x` and `y` as input arguments and, optionally, the weights `w` which computes the `pp` structure associated with $S(x)$ above; if the weights are not given, the default weights are

$$w_i = \frac{1}{\left(1 + \left(\frac{y_{i+1} - y_i}{h_i}\right)^2\right)^3}, \quad i = 1, 2, \ldots, n - 1$$

(e) compare weighted cubic splines and natural cubic splines on the following sets of sites

```
m = 10;
h = 1/10; % h<1/2
x = linspace(0,1,m);
y = zeros(size(x));
y(abs(x-1/2)<h) = 1;
```

with different values of `m` and `h`.

6. Consider the function $f(x) = \cos(x)$ in the interval $[0, 2\pi]$. Show the behaviour of `orderspline.m`

   (a) $\|f - S_h\|_\infty$ for $h \to 0$, where S_h is the spline interpolating at a set of sites with step-size h; consider the four types of splines.

   (b) $\|f^{(i)} - S_h^{(i)}\|_\infty$ for $h \to 0$, for $i = 1, 2, 3$.

   Repeat the exercise for $f(x) = \cos(x) + \frac{(x-2\pi)^2}{2}$. Finally, numerically detect the order of approximation by knot-a-not cubic splines of $f(x) = |x|^{\frac{7}{2}}$ in $[-\pi, \pi]$.

7. Consider the function $f(x) = \arctan(x)$ in the interval $[0, 30]$. Show the behavior of a quadratic spline where the value of $f'(0)$ or $f'(30)$ is prescribed and of a not-a-knot quadratic spline on a set of 11 equispaced sites. `quadraticspline.m` `spline2.m`

8. Consider the function $f(x) = \frac{1}{1+x^2}$ on the interval $[-5, 5]$. Take a set of uniformly distributed sites $\{x_i\}_{i=1}^{11}$ and add a random uniform noise to the corresponding values $y_i = f(x_i)$. Try the command `csaps` with different values of p. In particular `sspline.m`

   (a) show that for $p = 0$ the result corresponds to the least squares straight line;

   (b) show that for $p = 1$ the result corresponds to the natural spline;

   (c) try a value $0.9 < p < 1$;

   (d) get the default value used for p;

   (e) compare the result with a global polynomial fitting obtained by the command `polyfit` (and evaluated by the command `polyval`) with an appropriate degree.

Chapter 3

B-splines

3.1 Definition

Given the knots $\{t_j\}_{j=1}^n$ with $t_j \leq t_{j+1}$, we define the B-spline of order 1 (degree 0) for $j = 1, 2, \ldots, n-1$

$$B_{j,1}(t) = \begin{cases} 1, & t \in [t_j, t_{j+1}) \\ 0, & \text{otherwise} \end{cases}$$

and, recursively, the B-spline of order k (degree $k-1$) for $j = 1, 2, \ldots, n-k$

$$B_{j,k}(t) = a_{j,k}(t) + b_{j,k}(t)$$

where

$$a_{j,k}(t) = \begin{cases} \frac{t-t_j}{t_{j+k-1}-t_j} B_{j,k-1}(t), & t_j \neq t_{j+k-1} \\ 0, & t_j = t_{j+k-1} \end{cases}$$

$$b_{j,k}(t) = \begin{cases} \frac{t_{j+k}-t}{t_{j+k}-t_{j+1}} B_{j+1,k-1}(t), & t_{j+k} \neq t_{j+1} \\ 0, & t_{j+k} = t_{j+1} \end{cases}$$

We also set the following *continuity* condition

$$B_{j,k}(t) = 1, \quad \text{for } j = n-k \text{ and } t = t_n$$

The definition above can be easily, though not efficiently, implemented in a recursive way. The more efficient iterative way is `BSplinerec.m`

$$B_{i,h}(t) = \frac{t-t_i}{t_{i+h-1}-t_i} B_{i,h-1}(t) + \frac{t_{i+h}-t}{t_{i+h}-t_{i+1}} B_{i+1,h-1}(t), \quad h = 2, 3, \ldots, k, \quad i = 1, 2, \ldots, n-h$$

In this way, all the B-splines $B_{j,k}(t)$ for $j = 1, 2, \ldots, n-k$ are computed (this can be useful for B-splines curves). `BSplineEval.m`

3.2 Subdivision of spline curves

3.2.1 Chaikin's algorithm

Given a sequence of *control points* $\{b_j\}_{j=1}^n$, we can *refine* them through the iterations

$$
\begin{array}{lccccccccccccccc}
b_j^0: & b_1 & & & & b_2 & & & & b_3 & & & \dots & & b_n & & \\
 & b_1 & & b_1 & & b_2 & & b_2 & & b_3 & & b_3 & \dots & & b_n & & b_n \\
b_j^1: & & b_1 & & \frac{b_1+b_2}{2} & & b_2 & & \frac{b_2+b_3}{2} & & b_3 & & \dots & & & b_n & \\
b_j^2: & & & \frac{3b_1+b_2}{4} & & \frac{b_1+3b_2}{4} & & \frac{3b_2+b_3}{4} & & \frac{b_2+3b_3}{4} & & & \dots & & \frac{b_{n-1}+3b_n}{4} & & \\
b_j^3: & \dots & & & & & & & & & & & & & & &
\end{array}
$$

The first and last points possible require a special treatment. The sequence $\{b_j^2\}$ is the original Chaikin's sequence. Moreover, we can take the sequence $\{b_j^2\}$ and apply again and again the same refinement. We will call this procedure Chaikin's refinement algorithm of the second order.

3.2.2 Chaikin's algorithm in two variables

A note on the command `meshgrid`

In Matlab® it possible to define and plot a function of two variables with the following commands

```
x = linspace(0,1,11);
y = linspace(2,4,21);
[X,Y] = meshgrid(x,y);
f = @(x,y) x.^2.*y.^3;
Z = f(X,Y);
mesh(X,Y,Z)
```

If we look at `X`, we see that it is equivalent to `repmat(x,lenth(y),1)` and `Y` is equivalent to `repmat(y',1,length(x))`. Therefore, `Z(i,j)` corresponds to $f(x_j, y_i)$. In other words, the sequence `[X(:),Y(:)]` corresponds to the points in the grid numbered from bottom to top and from left to right. If we replace the command `meshgrid` with `ndgrid`, then `Z(i,j)` corresponds to $f(x_i, y_j)$ and the sequence `[X(:),Y(:)]` corresponds to the points in the grid numbered from left to right and from bottom to top.

3.2.3 Computation of $N_k(t-j)$

Given the infinite knot-sequence $\hat{t}_j = j$, $j \in \mathbb{Z}$, we consider

$$N_k(t-j) = \hat{B}_{j,k}(t)$$

where $\hat{B}_{j,k}(t)$ is the B-splines defined on the infinite knot-sequence $\{\hat{t}_j\}$. Let us consider first the case $k > 1$. For $j = 0$ it is

$$N_k(t) = \hat{B}_{0,k}(t)$$

which has support in $[t_0, t_k] = [0, k]$ with $N_k(t_k) = 0$. The sequence of knots $\{t_j\}_{j=1}^{k+1}$

$$0, 1, \ldots, k-1, k$$

is not enough, because $\hat{B}_{0,k}(\hat{t}_k) = B_{1,k}(t_{k+1}) = 1$ (by the continuity conditions). Therefore, the sequence $\{t_j\}_{j=1}^{k+2}$

$$0, 1, \ldots, k-1, k, k+1$$

has to be considered instead. Finally, $N_k(t-j)$ is computed as $B_{1,k}(t-j)$ on the finite sequence of knots $\{t_j\}_{j=1}^{k+2}$. For the case $k = 1$, the corresponding sequence of knots $\{t_j\}_{j=1}^{3}$

$$0, 1, 2$$

gives $N_1(t) = \hat{B}_{0,1}(t) = B_{1,1}(t)$ which has support in $[t_0, t_1] = [0, 1]$, with $N_1(1) = 0$.

3.2.4 Subdivision of periodic spline curves

First of all, observe that the support of $N_k(t-j)$ is $[j, j+k]$. Let us consider the curve

$$s(t) = \sum_j b_j N_k(t-j) \tag{3.1}$$

where $b_{i+n} = b_i$, $i \in \mathbb{Z}$, $n > 2$. It follows

$$\begin{aligned} s(t+n) &= \sum_j b_j N_k(t+n-j) = \sum_j b_j N_k(t-(j-n)) = \\ &= \sum_i b_{i+n} N_k(t-i) = \sum_i b_i N_k(t-i) = s(t) \end{aligned}$$

that is $s(t)$ is periodic with period n. Let us consider the interval $[k, n+k]$: for $k > 1$, the first N_k not completely zero on it is $N_k(t-1)$ and the last is $N_k(t-(n+k-1))$. For the particular case $k = 1$, the first not completely zero on $[0, n]$ is $N_1(t-1)$ and the last is $N_1(t-(n+k))$. Therefore

$$s(t) = \sum_{j=1}^{n+k-\min(k-1,1)} b_j N_k(t-j), \quad t \in [k, n+k] \tag{3.2}$$

It is possible to prove that the curve $s(t)$ is the limit of Chaikin's refine algorithm of order $k-1$ on the original sequence of control points $\{b_j\}_{j=1}^{n}$.

3.3 Interpolation by B-splines

Given a set of sites $\{x_i\}_{i=1}^n$ and a set of knots $\{t_j\}_{j=1}^{n+k}$ satisfying the Schoenberg–Whitney condition

$$x_i \in (t_i, t_{i+k})$$

with equality only at the extreme knots, each of which occurs with exact multiplicity k, then it is possible find the curve

$$s(t) = \sum_{j=1}^{n} b_j B_{j,k}(t), \quad s(x_i) = y_i$$

where the set $\{y_i\}_{i=1}^n$ is given. Clearly, the set of coefficients $\{b_j\}$ are the solution of

$$Ac = y, \quad A = (a_{i,j}) = B_{j,k}(x_i)$$

3.4 B-spline curves

We define the *B-spline curve* of order k as

$$S(t) = \sum_{i=1}^{n+1} b_i B_i^k(t), \quad t \in [0,1] \tag{3.3}$$

where $B_i^k(t)$ is the i-th, $1 \le i \le n+1$, B-spline $B_{i,k}(t)$ on the $n+1+k$ knots

$$\underbrace{0,\ldots,0}_{k-1 \text{ times}}, \frac{0}{n-k+2}, \frac{1}{n-k+2}, \ldots, \frac{n-k+2}{n-k+2}, \underbrace{1,\ldots,1}_{k-1 \text{ times}}$$

and $b_i \in \mathbb{R}^d$. Clearly, for $k = 1$ we recover the control points and for $k = 2$ the *control polygon*, i.e., the (possibly open) polygon of vertexes $\{b_i\}_{i=1}^{n+1}$. The maximum value for k is of course $n + 1$, in which case the B-spline curve coincides with the Bézier curve of degree n. It interpolates b_1 and b_{n+1}.

3.4.1 A (not so) efficient implementation

If `t` is a row vector of evaluation points, it is possible to evaluate all the B-splines through the iterative scheme by following commands

```
B = zeros(n+k,length(t));
for i = 1:n+k
  B(i,knots(i) <= t & t < knots(i+1)) = 1;
```

```
end
for h = 2:k
  for i = 1:n+1+k-h
    v1 = knots(i+h-1) == knots(i);
    v2 = knots(i+h) == knots(i+1);
    B(i,:) = (1-v1)*(t-knots(i))/(knots(i+h-1)-knots(i)+v1).*B(i,:)+...
             (1-v2)*(knots(i+h)-t)/(knots(i+h)-knots(i+1)+v2).*B(i+1,:);
  end
end
B = B(1:n+1,:);
B(n+1,t == knots(n+1+k)) = 1;
```

where each row of the matrix `b` corresponds to a B-spline. Then, if `b` is the matrix $d \times (n+1)$ of the control points, the B-spline curve evaluated at `t` is `b*B`. The only weak part of this algorithm is that usually `B` is a matrix with a lot of zero entries (since B-spline has local support). A more clever implementation should compute the rows of `B` only where they are different from zero.

3.4.2 Periodic B-spline curves

We have already seen a periodic curve defined by B-splines (see (3.2)). We want to rewrite it in terms of B-spline curve. We consider the cases $k > 1$. First of all, we write that formula for $n+1$ periodic control points

$$s(z) = \sum_{j=1}^{n+k} b_j N_k(z-j), \quad z \in [k, n+1+k]$$

where $b_{n+1+i} = b_i$, $i \in \mathbb{Z}$. We now set

$$S(t) = s(z), \quad t = \frac{z-k}{n+1}$$

We have

$$S(t) = \sum_{j=1}^{n+k} b_j N_k((n+1)t+k-j), \quad t \in [0,1]$$

with

$$N_k((n+1)t+k-j) = B_{1,k}((n+1)t+k-j)$$

$B_{1,k}$ the B-spline defined on the knots $[t_1, t_2, \ldots, t_k, t_{k+1}, t_{k+2}] = [0, 1, \ldots, k-1, k, k+1]$. We apply its definition

$$\begin{aligned} B_{1,k}((n+1)t+k-j) &= \frac{((n+1)t+k-j)-0}{(k-1)-0} B_{1,k-1}((n+1)t+k-j) + \\ &+ \frac{k-((n+1)t+k-j)}{k-1} B_{2,k-1}((n+1)t+k-j) = \\ &= \frac{t-\frac{j-k}{n+1}}{\frac{j-1}{n+1}-\frac{j-k}{n+1}} B_{1,k-1}((n+1)t+k-j) + \\ &+ \frac{\frac{j}{n+1}-t}{\frac{j}{n+1}-\frac{j+1-k}{n+1}} B_{2,k-1}((n+1)t+k-j) \end{aligned}$$

If we now define the B-splines B_j^k on the knots $t^j = (j-k)/(n+1)$, $j = 1, 2, \ldots, n+2k$, we have $B_{i,k}((n+1)t+k-j) = B_{j+i-1}^k(t)$, $j = 1, 2, \ldots, n+k$. In particular, $N_k((n+1)t+k-i) = B_{1,k}((n+1)t+k-i) = B_i^k(t)$. Therefore

$$S(t) = \sum_{i=1}^{n+k} b_i B_i^k(t), \quad t \in [0,1]$$

From the implementation point of view, we observe that

$$B_i^k(t) = B_k^k\left(t - \frac{i-k}{n+1}\right) = B_k^k\left(\frac{(n+1)t-i+k}{n+1}\right)$$

`periodicB.m` and B_k^k requires only the knots $t^j = (j-k)/(n+1)$, $j = 1, 2, \ldots, 2k+1$.

`periodicBScurve.m`

An application to fonts

> TrueType fonts use Bézier splines composed of quadratic Bézier curves. Modern imaging systems like PostScript, Asymptote, Metafont, and SVG use Bézier splines composed of cubic Bézier curves for drawing curved shapes. OpenType fonts can use either types, depending on the flavor of the font [Wikipedia].

Suppose

$$S(t) = \sum_{i=1}^{n+1} b_i B_i^k(t)$$

is a non-periodic B-spline curve of order $k = 3$ (degree 2) representing a font, with $b_1 = b_{n+1}$. If we consider the piecewise Bézier curve of degree 2 with control points $(b_i + b_{i+1})/2$, b_{i+1}, $(b_{i+1} + b_{i+2})/2$, $i = 1, 2, \ldots, n$, where

$b_{n+2} = b_2$, then it "almost" coincides with the B-spline curve. The differences are in the first and last control point. In order to have an exact match, we need the B-spline curve to be periodic. We can repeat with $k = 4$. The piecewise Bézier curve of degree 3 to be considered has control points $((b_i/3+2b_{i+1}/3)+(2b_{i+1}/3+b_{i+2}/3))/2$, $2b_{i+1}/3+b_{i+2}/3$, $b_{i+1}/3+2b_{i+2}/3$, and $((b_{i+1}/3 + 2b_{i+2}/3) + (2b_{i+2}/3 + b_{i+3}/3))/2$, $i = 1, 2, \ldots, n$, where $b_{n+3} = b_3$.

`font.m`

`periodicfont.m`

`periodicfont3.m`

3.5 B-spline in Matlab®

Matlab® handles B-splines through the `B-` structures. They contain the following fields:

- `form`: `B-`;
- `knots`: vector of knots $\{t_j\}$;
- `coefs`: vector of coefficients $\{b_j\}$;
- `number`: number of coefficients;
- `order`: B-spline order k;
- `dim`: number of sets of values related to the knots.

About the field `dim`, it corresponds to the number of rows of `coefs`, that is 1 for a B-spline function, 2 for a two-dimensional B-spline curve and 3 for a three-dimensional B-spline curve. It is possible to assemble the structure `B-` through the command `spmak` and to evaluate at a set of points through the command `spval`.

For instance, given $n+1$ coefficients $\{b_j\}_{j=1}^{n+1}$, the two-dimensional B-spline curve of order k defined in (3.3) can be evaluated through the commands

```
n = size(b,2)-1; % b of size 2 x (n+1)
knots = linspace(0,1,n+1-k+2);
knots = [zeros(1,k-1),knots,ones(1,k-1)];
Bform = spmak(knots,b);
t = linspace(0,1);
s = spval(Bform,t);
plot(s(1,:),s(2,:))
```

3.6 Knot insertion

Given the B-spline curve

$$S(t) = \sum_{i=1}^{n+1} b_i B_i^k(t)$$

with knots $\{t_j\}_{j=1}^{n+1+k}$ we can perform a *knot insertion* of knot $\bar{t}$ in the following way:

knotinsertion.m

- find h such that $\bar{t} \in [t_h, t_{h+1})$ (if $\bar{t} = 1$, set $h = n + 1$)
- define the new set of $n + 2$ control points

$$b_1, b_2, \ldots, b_{h-k+1}, \bar{b}_{h-k+2}, \bar{b}_{h-k+3}, \ldots, \bar{b}_h, b_h, b_{h+1}, \ldots, b_{n+1}$$

  where

$$\bar{b}_i = (1-a_i)b_{i-1} + a_i b_i, \quad a_i = \frac{\bar{t} - t_i}{t_{i+k-1} - t_i}, \quad i = h-k+2, h-k+3, \ldots, h$$

  with $a_i = 0$ if $t_{i+k-1} = t_i$.

- define the new sequence of $n + 2 + k$ knots

$$t_1, t_2, \ldots, t_h, \bar{t}, t_{h+1}, t_{h+2}, \ldots, t_{n+1+k}$$

3.7 Exercises

BSplineEval.m

1. Write a function computing the B-spline $B_{i,k}(t)$.

curves2d.m
curves3d.m

2. Show the properties of subdivision of periodic spline curves.

bspline1.m
bspline2.m
bspline3.m

3. Reproduce the behaviour of splines of degree 1, not-a-knot quadratic splines and not-a-knot cubic splines by B-splines.

orderbspline.m

4. Plot the B-splines of order 4 for the knots $[0, 0, 0, 0, 1, 2, 3, 3, 3, 4, 5, 6, 6, 6, 6]$. Using few sites, say 9, construct a cubic spline interpolation of $f(x) = |x|^{1/2}$ in the interval $[-1, 1]$, using the command `spapi`.

badknotssites.m

5. Schoenberg–Withney conditions are not enough to guarantee interpolation. The interpolation matrix, though non-singular, can become ill-conditioned. Consider the following example:

```
function B = badknotssites(m)
% m has to be odd
x = linspace(-1,1,m)';
xx = linspace(-1,1,1001)';
y = abs(x).^(1/2);
t = linspace(-1,0,(m-3)/2)';
t = [t;0];
t = [t;linspace(0,1,(m-3)/2)'];
t = [t(1)*ones(3,1);t;t(m-2)*ones(3,1)];
B = spapi(t,x,y);
```

Plot the result with different values of `m`, from 11 to 501. Numerically verify that Schoenberg–Withney conditions are always satisfied.

6. Given the control points $b_1 = (0,0)$, $b_2 = (0,1)$, $b_3 = (1,1)$, $b_4 = (1,0)$, $b_5 = (1/2,0)$ and $b_6 = (1/2,1/2)$, show the B-spline curves of any order from $k = 1$ to $k = n+1$. `BScurve.m`

7? Take the control points `B.m` `BScurve0.m` `BScurve1.m` `BScurve2.m` `BScurve3.m` `BScurve4.m`

```
b =  [0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9;...
     0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0];
```

and plot the B-spline curve of order 3. Now take the control points

```
b1 =  [0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9;...
      0,1,1,0,0,1,1,0,0,1.5,1,0,0,1,1,0,0,1,1,0];
```

and plot the B-spline curve of order 3. Do the same with the Bézier curves. Show that the difference between the two Bézier curves is not local (for instance, compute their distance in $t \in [0.8, 1]$). `locality.m`

Chapter 4

Bézier curves

4.1 Bernstein's polynomials

We define Bernstein's polynomial of degree n as

$$B_i^n(t) = \binom{n}{i-1} t^{i-1}(1-t)^{n-i+1}, \quad i = 1, 2, \ldots, n+1$$

where usually $t \in [0, 1]$. Notice that $B_1^0 \equiv 1$. It is possible to show that

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t), \quad n > 0, \quad i = 1, 2, \ldots, n+1 \tag{4.1}$$

where $B_0^n \equiv 0$ for $n > 0$ and $B_i^n \equiv 0$ if $i > n+1$. A *Bézier* curve of degree n is

$$B(t) = \sum_{i=1}^{n+1} b_i B_i^n(t)$$

where $b_i \in \mathbb{R}^d$ are called *control points*.

4.2 Evaluation of a Bézier curve

Given a Bézier curve of degree n and control points $\{b_i\}_{i=1}^{n+1}$, let us define $b_i^0(t) \equiv b_i$ and

$$b_i^r(t) = (1-t)b_i^{r-1}(t) + tb_{i+1}^{r-1}(t), \quad r = 1, 2, \ldots, n, \quad i = 1, 2, \ldots, n-r+1$$

(De Casteljau's algorithm). We have

$$b_1^n(t) = \sum_{i=1}^{n+1} b_i B_i^n(t) \tag{4.2}$$

There is a nice result about this algorithm: given $t_0 \in [0,1]$, if we define

$$\begin{aligned} c_{i+1} &= b_1^i(t_0), & i &= 0,1,\ldots,n \\ d_i &= b_i^{n+1-i}(t_0), & i &= 1,2,\ldots,n+1 \end{aligned}$$

then the curve

$$\sum_{i=1}^{n+1} b_i B_i^n(t)$$

is the union of the two disjoint curves

$$\sum_{i=1}^{n+1} c_i B_i^n(t), \quad \sum_{i=1}^{n+1} d_i B_i^n(t),$$

4.3 Affine invariance

If an affine map

$$\Phi(x) = Ax + b$$

where $x, b \in \mathbb{R}^d$, $A \in \mathbb{R}^{d\times d}$ is applied to a Bézier curve, then the result is equivalent to the Bézier curve of the affine images of its control points, that is

$$\Phi(B(t)) = \sum_{i=1}^{n+1} \Phi(b_i) B_i^n(t) \tag{4.3}$$

4.4 Degree elevation

Given a Bézier curve of degree n with control points $\{b_i\}_{i=1}^{n+1}$ we can make a degree elevation considering the Bézier curve of degree $n+1$ with control points given by

$$\begin{aligned} \hat{b}_1 &= b_1 \\ \hat{b}_i &= \frac{i-1}{n+1} b_{i-1} + \frac{n-i+2}{n+1} b_i, \quad i = 2,3,\ldots,n+1 \\ \hat{b}_{n+2} &= b_{n+1} \end{aligned}$$

4.5 Exercises

`Bern.m`

1. Implement the recurrence relation (4.1) for Bernstein's polynomials with a function `Bern(i,k,t)` and verify it coincides with the definition.

2? Given the function `decasteljau.m`, verify equivalence (4.2). `decasteljau.m`

3 Given the control points $b_1 = (0, 0)$, $b_2 = (0, 1)$, $b_3 = (1, 1)$, $b_4 = (1, 0)$, $b_5 = (1/2, 0)$ and $b_6 = (1/2, 1/2)$, verify equivalence (4.3) with a rotation, a translation and a scaling on the corresponding Bézier curve. `bezier.m`

4 Try the degree elevation on the Bézier curve defined by the control points above. `degreeelevation.m`

Chapter 5

Fast Gauss transform

5.1 On Hermite's polynomials

Let us define the *Hermite polynomials* of degree α

$$H_\alpha(x) = (-1)^\alpha \mathrm{e}^{x^2} \frac{\mathrm{d}^\alpha}{\mathrm{d}x^\alpha} \mathrm{e}^{-x^2}$$

It can be proved that they satisfy the following recurrence relation

$$\begin{aligned} H_0(x) = 1,\ H_1(x) = 2x \\ H_{\alpha+1}(x) = 2xH_\alpha(x) - 2\alpha H_{\alpha-1}(x), \quad \alpha \geq 1 \end{aligned}$$

We define the *Hermite function* $h_\alpha(x) = H_\alpha(x)\mathrm{e}^{-x^2}$.

5.2 Fast Gauss sum

In order to compute

$$G(y_j) = \sum_{i=1}^{N} q_i \exp\left(-\frac{(x_i - y_j)^2}{\delta}\right), \quad j = 1, 2, \ldots, M \tag{5.1}$$

we can use the Matlab® commands

```
[X,Y] = ndgrid(x,y);
E = exp(-(X-Y).^2/delta);
G = q(:)'*E;
```

In the d-variate case, we have

$$\begin{aligned} G(y_j) &= \sum_{i=1}^{N} q_i \exp\left(-\frac{|x_i - y_j|^2}{\delta}\right) = \\ &= \sum_{i=1}^{N} q_i \exp\left(-\frac{(x_{i,1} - y_{j,1})^2}{\delta}\right) \cdot \ldots \cdot \exp\left(-\frac{(x_{i,d} - y_{j,d})^2}{\delta}\right) \end{aligned} \tag{5.2}$$

Therefore, the direct implementation is

```
d = size(x,1); % x a dxN array
E = ones(size(x,2),size(y,2));
for k = 1:d
  [X,Y] = ndgrid(x(k,:),y(k,:));
  E = E.*exp(-(X-Y).^2/delta);
end
G = q(:)'*E;
```

For the fast implementation, we can subdivide the source points $\{x_i\}_i$ into subintervals of length $\sqrt{\delta}$, with centers $\{\bar{x}_m\}$. Therefore

$$\begin{aligned} G(y_j) &= \sum_m \sum_{|x_i - \bar{x}_m| \le \sqrt{\delta}/2} q_i \exp\left(-\frac{(x_i - y_j)^2}{\delta}\right) = \\ &= \sum_m \sum_{\beta=0}^{\infty} \frac{1}{\beta!} \left(\frac{\bar{y} - y_j}{\sqrt{\delta}}\right)^{\beta} \sum_{\alpha=0}^{\infty} \frac{h_{\alpha+\beta}\left(\frac{\bar{y}-\bar{x}_m}{\sqrt{\delta}}\right)}{\alpha!} \sum_{|x_i - \bar{x}_m| \le \sqrt{\delta}/2} q_i \left(\frac{x_i - \bar{x}_m}{\sqrt{\delta}}\right)^{\alpha} \end{aligned}$$

FGT.m

for a given $\bar{y}$. If $|\bar{y} - y_j| < \sqrt{\delta}/2$ then

$$G(y_j) \approx \sum_m \sum_{\beta=0}^{k} \frac{1}{\beta!} \left(\frac{\bar{y} - y_j}{\sqrt{\delta}}\right)^{\beta} \sum_{\alpha=0}^{k} \frac{h_{\alpha+\beta}\left(\frac{\bar{y}-\bar{x}_m}{\sqrt{\delta}}\right)}{\alpha!} \sum_{|x_i - \bar{x}_m| < \sqrt{\delta}/2} q_i \left(\frac{x_i - \bar{x}_m}{\sqrt{\delta}}\right)^{\alpha}$$

The `FIGTree` library

The Matlab® command

```
figtree(x,h,q,y,epsilon)
```

(from the library FIGTree) approximates the Gauss transform

$$G(y_j) = \sum_{i=1}^{N} q_i \exp\left(-\frac{|x_i - y_j|^2}{h^2}\right), \quad j = 1, 2, \ldots, M$$

by $\hat{G}(y_j)$, with $\left|\hat{G}(y_j) - G(y_j)\right| \leq \varepsilon \cdot \sum_i q_i$. Here `x` is a $d \times N$, `y` a $d \times M$ array and `q` a $N \times L$ array: in fact, it is possible to compute L Gauss transforms at once which correspond to different sets of weights $\{q_i\}$.

5.3 Applications

5.3.1 Heat equation

The solution of

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,x) = \Delta u(t,x), & x \in \mathbb{R}^d,\ t > 0 \\ u(t,x) = u_0(x), & \lim\limits_{|x|\to\infty} u_0(x) = 0 \end{cases}$$

is

$$u(t,x) = \frac{1}{(4\pi t)^{d/2}} \int_{\mathbb{R}^d} u_0(\xi) \exp\left(-\frac{|x-\xi|^2}{4t}\right) \mathrm{d}\xi$$

The unbounded domain $\mathbb{R}^d$ can be truncated to $(-a,a)^d$ and the trapezoidal rule applied on a uniform grid with $h = 2a/(N+1)$

$$u(t,\xi_j) \approx \frac{h^d}{(4\pi t)^{d/2}} \sum_{i=1}^{N} u_0(\xi_i) \exp\left(-\frac{(\xi_j - \xi_i)^2}{4t}\right)$$

with ξ_i inner grid points.

Finite differences for the two-dimensional heat equation

Let us consider a uniform grid of inner points $\xi_k = (x_i, y_j)$, $k = (j-1)N + i$, $i, j = 1, 2, \ldots, N$, in a square $[-a,a]^2$, with $x_i = -a + hi$, $y_j = -a + hj$, $h = 2a/(N+1)$. The standard matrix A of one-dimensional finite differences (central, second order, homogeneous Dirichlet boundary conditions) is

```
A = toeplitz(sparse([1,1],[1,2],[-2,1]/h^2,1,N));
```

The matrix for two-dimensional finite differences is then $A_2 = I \otimes A + A \otimes I$

```
kron(speye(N),A)+kron(A,speye(N))
```

With this discretization in space, the heat equation becomes

$$\begin{cases} u'(t) = A_2 u(t) \\ u(0) = u_0(t) \end{cases}$$

and can be solved with a method for ODEs. For instance, with forward Euler, the code is

```
x = linspace(0,1,N+2);
x = x(2:N+1);
h = 1/(N+1);
[X,Y] = ndgrid(x);
A = toeplitz(sparse([1,1],[1,2],[-2,1]/h^2,1,N));
A2 = kron(speye(N),A)+kron(A,speye(N));
u0 = sin(pi*X).*sin(pi*Y);
k = 0.002; % < 2/max(abs(eig(A2)))
u = u0(:);
for n = 1:10
  u = u+k*A2*u;
end
mesh(X,Y,reshape(u,size(X)))
```

5.4 Exercises

1? Solve the one-dimensional heat equation in the interval $[-10, 10]$ up to the final time $t^\star = 1$ using the (fast) Gauss transform. Take as initial value the function

$$u_0(x) = \begin{cases} 1 & |x| \leq 1 \\ 0 & |x| > 1 \end{cases}$$

Compare the result with a method of lines. Repeat the exercise in two dimensions.

heat.m

Chapter 6

RBF

6.1 Interpolation by Radial Basis Functions

We consider the interpolation of a given function $f\colon \Omega \subseteq \mathbb{R}^d \to$ by *radial basis functions* $\varphi(r)$ at sites $\{x_i\}_{i=1}^n \in \Omega$

$$f(x) \approx \mathcal{P}_f(x) = \sum_{i=1}^{n} c_i \varphi(|x - x_i|)$$

By the interpolation condition $\mathcal{P}_f(x_j) = f(x_j)$ we get the linear system

$$\sum_{i=1}^{n} c_i \varphi(|x_j - x_i|) = f(x_j), \quad j = 1, 2, \ldots, n$$

that is

$$A_{\mathrm{I}} c = f$$

with $A_{\mathrm{I}} = (a_{ij}) = (\varphi(|x_j - x_i|))$, $c = [c_1, c_2, \ldots, c_n]^{\mathrm{t}}$ and $f_i = f(x_i)$. Given the $d \times n$ array `x` and a function `phi.m`, the code to recover the coefficients is quite standard

```
d = size(x,1);
DM = zeros(size(x,2));
for k = 1:d
  [xj,xi] = ndgrid(x(k,:));
  DM = DM+(xj-xi).^2;
end
AI = phi(epsilon*sqrt(DM));
c = AI\f;
```

If multiple interpolations with the same sites have to be performed, the matrix A_{I} can be factorized (usually with Cholesky's algorithm) once and for all.

In order to evaluate the RBF at a set of evaluation points $\{y_j\}_{j=1}^m$, we can construct the evaluation matrix

$$A_{\mathrm{E}} = (a_{ji}) = (\varphi(|y_j - x_i|))$$

and then

$$\mathcal{P}_f(y) = \begin{bmatrix} \mathcal{P}_f(y_1) \\ \mathcal{P}_f(y_2) \\ \vdots \\ \mathcal{P}_f(y_m) \end{bmatrix} = A_{\mathrm{E}} c$$

The code is

```
d = size(x,1);
DM = zeros(size(y,2),size(x,2));
for k = 1:d
  [yj,xi] = ndgrid(y(k,:),x(k,:));
  DM = DM+(yj-xi).^2;
end
Pf = phi(epsilon*sqrt(DM))*c;
```

If the number m of evaluation points is large, it may be convenient to split the evaluation matrix.

6.1.1 Gaussian RBF

The function $\varphi(r)$ is

$$\varphi(r) = \mathrm{e}^{-(\varepsilon r)^2}$$

where ε is the *shape parameter*. Given the coefficients, in this case it is possible to apply the Fast Gauss Transform in order to evaluate $\mathcal{P}_f(y_j)$

```
figtree(x,1/epsilon^2,c,y,1e-10)
```

6.1.2 Multiquadric

The function $\varphi(r)$ is

$$\varphi(r) = \sqrt{1 + (\varepsilon r)^2}$$

6.1.3 Inverse multiquadric

The function $\varphi(r)$ is

$$\varphi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$$

6.2 Franke's function

Probably the most used two-dimensional test function for interpolation and approximation is Franke's function

$$f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2 + (9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{9y+1}{10}\right) + \\ + \frac{1}{2}\exp\left(-\frac{(9x-7)^2 + (9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right)$$

in the unit square $[0,1]^2$.

6.3 Exercises

1? Measure the maximum interpolation error over a grid of 50×50 points, using $n \times n$ interpolation points ($n = 5, 10, \ldots, 35$), different values for the shape parameter ε ($\varepsilon = 0.5, 1.5, \ldots, 4.5$) for any of the known RBFs for the interpolation of Franke's function. Why is it sometimes not possible to correctly solve the arising linear system?

Chapter 7

Triangulations and some elements

7.1 Delaunay triangulation

Two-dimensional Delaunay's triangulation in Matlab® is created by

```
dt = DelaunayTri(x,y)
```

where `x` and `y` are (column) vectors. The result `dt` is a structure containing the fields `Constraints`, `X` (corresponding to `[x(:),y(:)]`) and `Triangulation`, an $\ell \times 3$ integer array: triangle l, $l = 1, 2, \ldots, \ell$ is made of `X(dt.Triangulation(l,1),:)`, `X(dt.Triangulation(l,2),:)` and `X(dt.Triangulation(l,3),:)`.

Given a point (p, q) it is possible to know which triangle it belongs to by the command

```
pointLocation(dt,p,q)
```

The result is the number of the triangle in `dt.Triangulation` or `NaN` if outside the triangulation.

Given a triangulation `dt`, it is possible to interpolate a function `fun` through the command

```
F = TriScatteredInterp(dt,v,method)
```

where `v=fun(dt.X(:,1),dt.X(:,2))` and `method` is `'natural'`, `'linear'` or `'nearest'`. Finally, it is possible to evaluate `F` as a normal bivariate function.

7.2 (Hsieh–)Clough–Tocher element

Given a triangle of points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ and a function f defined on it, it is possible to define a piecewise polynomial of degree 3 by prescribing the following conditions:

- interpolation of f and ∇f at the vertexes ($6 \cdot 3$ conditions)
- normal derivatives at the midpoint of each edge (3 conditions)
- $\mathcal{C}^1$ continuity at the barycenter ($3 \cdot 2$ conditions)
- continuity of normal derivatives at the midpoint of the seams (3 conditions)

The result of a Hsieh–Clough–Tocher interpolation could be a structure `hct` containing the fields

`HCT_sys.m`
`triangle.m`

- `x`: an array 3×3 with `x(i,j)` the value of the j-th abscissa of the i-th subtriangle;
- `y`: equivalent to `x`;
- `coefs`: the coefficients $\{a_{ik}\}_{k=1}^{10}$ of the cubic polynomials defined on the i-th subtriangle in the form

$$a_{i1} + a_{i2}x + a_{i3}y + a_{i4}x^2 + a_{i5}xy + a_{i6}y^2 + a_{i7}x^3 + a_{i8}x^2y + a_{i9}xy^2 + a_{i10}y^3$$

7.3 Reduced (Hsieh–)Clough–Tocher element

In this reduced model we prescribe the normal derivative at each edge to be a linear function in x and y. Let us consider an edge with vertexes $A = (x_A, y_A)$ and $B = (x_B, y_B)$. The normal derivative of the corresponding cubic polynomial $p(x, y) = a_1 + a_2x + a_3y + \ldots + a_{10}y^3$ evaluated at a point $[x(t), y(t)] = tA + (1 - t)B$ is

$$\partial_x p(x(t), y(t))(y_A - y_B) + \partial_y p(x(t), y(t))(x_B - x_A)$$

(a normalization factor is missing). We want the quadratic terms in t of this expression to be zero. We find

$$\begin{aligned} &3a_7(x_A - x_B)^2(y_A - y_B) + a_8(x_A - x_B)(2(y_A - y_B)^2 - (x_A - x_B)^2) + \\ &a_9(y_B - y_A)(2(x_B - x_A)^2 - (y_A - y_B)^2) + 3a_{10}(y_A - y_B)^2(x_B - x_A) = 0 \end{aligned}$$

These three equations (for $A = 1, 2, 3$ and $B = 2, 3, 1$) have to replace the equations that impose the normal derivatives in the (full) HCT element.

`red_HCT_sys.m`

7.3.1 Estimate of first derivatives

It may be possible that only f is available. In this case, we have to estimate the first derivatives at the vertexes and normal derivatives at the midpoint of each edge. The easiest way is the following: for each point (x, y), we compute the plane interpolating the values of f on each triangle containing (x, y) as vertex. If the vertexes are (x_1, y_1), (x_2, y_2) and (x_3, y_3), then the plane has equation

$$z = c_1 + c_2 x + c_3 y$$

with

$$\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} f(x_1, y_1) \\ f(x_2, y_2) \\ f(x_3, y_3) \end{bmatrix}$$

We approximate $\nabla f(x, y)$ by taking the average of the gradient of the planes passing through (x, y). Once the gradient at the vertexes is approximated, we can compute the normal derivatives at the midpoint of each edge as

$$\nabla f\left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right) \cdot \vec{n}_3 \approx \frac{-\frac{\tilde{f}_x(x_1,y_1)+\tilde{f}_x(x_2,y_2)}{2}(y_2-y_1) + \frac{\tilde{f}_y(x_1,y_1)+\tilde{f}_y(x_2,y_2)}{2}(x_2-x_1)}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}}$$

$$\nabla f\left(\frac{x_2+x_3}{2}, \frac{y_2+y_3}{2}\right) \cdot \vec{n}_1 \approx \frac{-\frac{\tilde{f}_x(x_2,y_2)+\tilde{f}_x(x_3,y_3)}{2}(y_3-y_2) + \frac{\tilde{f}_y(x_2,y_2)+\tilde{f}_y(x_3,y_3)}{2}(x_3-x_2)}{\sqrt{(x_3-x_2)^2+(y_3-y_2)^2}}$$

$$\nabla f\left(\frac{x_1+x_3}{2}, \frac{y_1+y_3}{2}\right) \cdot \vec{n}_2 \approx \frac{-\frac{\tilde{f}_x(x_1,y_1)+\tilde{f}_x(x_3,y_3)}{2}(y_1-y_3) + \frac{\tilde{f}_y(x_1,y_1)+\tilde{f}_y(x_3,y_3)}{2}(x_1-x_3)}{\sqrt{(x_1-x_3)^2+(y_1-y_3)^2}}$$

This approach is very simple and it can be used for quite regular meshes. More sophisticated approaches may use local least square fitting.

7.3.2 Determining whether a point is inside a triangle

Once we know that a point $P = (p, q)$ is inside a triangle, we have to find in which subtriangle it is contained. We use in practice the barycentric coordinates. Given the three points of a triangle A, B, C, P is inside (or at most on an edge) if

$$P = A + u(C - A) + v(B - A)$$

with $u, v \geq 0$ and $u + v \leq 1$. Given A, B, C and P we have to compute u and v. If we call $v_0 = C - A$, $v_1 = B - A$ and $v_2 = P - A$, we have

$$u = \frac{(v_1 \cdot v_1)(v_2 \cdot v_0) - (v_1 \cdot v_0)(v_2 \cdot v_1)}{(v_0 \cdot v_0)(v_1 \cdot v_1) - (v_0 \cdot v_1)(v_1 \cdot v_0)}$$

$$v = \frac{(v_0 \cdot v_0)(v_2 \cdot v_1) - (v_0 \cdot v_1)(v_2 \cdot v_0)}{(v_0 \cdot v_0)(v_1 \cdot v_1) - (v_0 \cdot v_1)(v_1 \cdot v_0)}$$

Since these values are computed up to machine precision, the tests $u \geq 0$, $v \geq 0$ and $u + v \leq 1$ have to be slightly modified.

7.4 Exercises

`interpolation.m`
`triangle.m`
`whichsubtriangle.m`

1? Given some points on the unit square $[0, 1]^2$

- build a Delaunay triangulation;
- compute the HCT interpolant of f (using ∇f) for each triangle;
- evaluate the piecewise interpolant at a regular grid of 50×50 points;
- measure the interpolation error when f is a cubic bivariate polynomial and Franke's function;
- compute the HCT interpolant using an estimate of ∇f and measure the interpolation errors;
- compare the results with `TriScatteredInterp`.

2 Modify the function `HCT_sys.m` in order to compute the reduced HCT element. Try the previous exercise with this element.