A Package for Representing $\mathcal{C}^1$ Interpolating Surfaces: Application to the Lagoon of Venice's Bed

M. Morandi Cecchi
Dipartimento di Matematica Pura e Applicata, Via Belzoni 7, 35131 Padova, Italy
E-mail: mcecchi@math.unipd.it

S. De Marchi
Dipartimento di Matematica e Informatica, Via delle Scienze 206, 33100 Udine, Italy
E-mail: demarchi@dimi.uniud.it

D. Fasoli
Verona Software S.r.l., Via Trento 18, 37124 Verona Italy
E-mail: vrsoft@sis.it

The paper deals with the description of a method and the accompaining software, the package LABSUP, for representing $\mathcal{C}^1$ interpolating surfaces. The application to the lagoon of Venice's bed is also proposed. The surfaces are built over the Delaunay triangulation and the polynomial patches used for the representation can be chosen among the Q_{18} element, the Clough-Tocher or the Powell-Sabin finite elements or simply using global Bézier methods. The first three patches, requires the knowledge of the gradients at the nodes, or at least a *suitable estimation* of them. Therefore, interesting in itself, is the *derivative estimation process* based on the minimization of the energy functional associated with the interpolant. For the representation of the lagoon of Venice's bed we only used the *reduced* Clough-Tocher finite element, due to the high number of points involved for which one needs to compute the Delaunay triangulation and simply the partial derivatives of first order. A brief description of the software modules together with some graphical results of *parts* of the lagoon of Venice's bed are also presented.

Keywords: Interpolation of scattered data, algorithms for functional minimization, computer graphics and computational geometry.

AMS Subject classification: 65D05, 65Y25, 65D15.

1. Introduction

The representation of complicated surfaces with smooth functions is one of the most interesting problems to solve for people working in the field of visualization, modeling or graphics in general. The representation of Venice's lagoon bed, i.e. the application of the present work, has its relevance for two main reasons.

1. It represents an interesting bidimensional *scattered data* problem for testing the performance of some visualization methods and rendering algorithms.

2. It allows us to display the bed of the lagoon in order to get some details and to study some phenomena acting on the morphology of the bed itself, such as the depth changing, the tide action, pollution and so on.

In bidimensional scattered data problems one is asked to search for an interpolating or approximating surface to the given data. Usually, the surface is built as a set of polynomial patches of small degree. The requirement that the joints between elements be C^1 gives rise to a surface that will be globally C^1. The supplied data must include values of gradients in order to get smooth interpolating surfaces. In scattered data problems these quantities are normally *not known* a priori, hence the necessity to generate or to estimate them. This is the most important task for many methods producing smooth interpolating surfaces to scattered data [1,2].

In the present paper we face the problem of representing Venice's lagoon bed by a C^1 piecewise polynomial interpolating surface. The chosen polynomial patches are the *cubic Clough-Tocher* elements (cf. [11,15]). Derivative estimates are supplied by the minimization on a suitable domain of the energy functional associated with the Clough-Tocher element, expressed in Bernstein-Bézier form. The domain of minimization, that we shall denote as a *cell*, is the set of triangles that share the point at which we estimate the derivatives. The estimation method specializes the global method by Alfeld [2] and optimizes the *local Method II* by Cline and Renka [17].

The paper is organized as follows. In Section 2 we briefly present the C^1 interpolation problem for scattered data. In Section 3 we examine Delaunay triangulations and constrained Delaunay triangulations. Moreover we outline a *deletion algorithm*, that in the present implementation solves the problem of erasing undesirable lines connecting nodes around holes or lying on the boundary of the lagoon. In Section 4 we describe the interpolation scheme chosen and we give details for the derivative estimation procedure. In Section 5, we present the package LABSUP describing the programs that made it up with a practical way to subdivide them depending on the task they perform. In Section 6 we give some practical examples which explain how to use the programs in the package and finally in Section 7 we describe and briefly comment some pictures of the lagoon of Venice's bed as produced with the package.

2. The problem of C^1 interpolation of scattered data

The problem can be formulated in a simple mathematical way as follows. Let $\{(x_i, y_i, z_i)\}_{1\leq i\leq N}$ be a set of data points and let $(z_{i,x}, z_{i,y})$, $i = 1, \ldots, N$ be the corresponding gradients. Let H be a bounded subset of the Euclidean plane containing the points (x_i, y_i). We search for a function $f \in C^1(H)$ satisfying the interpolation conditions:

$$f(x_i, y_i) = z_i \, , \tag{1}$$

$$f_x(x_i, y_i) = z_{i,x} \, , \tag{2}$$

$$f_y(x_i, y_i) = z_{i,y} \, , \tag{3}$$

for all $i \in \{1, \ldots, N\}$.

A solution to this problem is sought by performing the following steps.

1. Construction of a triangulation τ of H, that is a subdivision of H into triangles (called *macro-triangles*), whose vertices are exactly the points (x_i, y_i), for which two different triangles intersect or in a common vertex or in an edge.

2. Subdivision of each triangle in *subtriangles*, called *micro-triangles.*

3. Representation of the solution on each triangle as a bivariate polynomial of degree $\leq d$. At this stage we shall take into account the interpolation of the given points and their gradients, in order to guarantee C^1 continuity between adjacent micro and macro-triangles.

In most cases of real interest the initial data are only the values z_i and not the corresponding gradients $(z_{i,x}, z_{i,y})$. Therefore, we have to provide in some way the missing information by simply making use of the data (x_i, y_i, z_i). A C^1 solution which only interpolates the data z_i would obviously be meaningless. In fact, the great number of degrees of freedom present in triangulations with several elements, would produce many different interpolants all satisfying the C^1 continuity [12]. Therefore, the problem is a *Hermite interpolation problem* where the unknowns are the values of the gradients at the interpolation points.

3. Constrained Delaunay triangulation

Let us assume that the domain H is the convex hull of the points (x_i, y_i), $i = 1, \ldots, N$ on which we have built a *Delaunay triangulation* τ . As well known, this triangulation satisfies the *max-min property*, that is it maximizes the smallest angle among all the angles of the triangles of τ (cf. [19]). Thus, in the interior of τ long thin triangles are avoided. These triangles are indeed not desirable since in the derivative estimation process they produce bad estimates (cf.[12,19]). On the other hand, Delaunay triangulations create such triangles when there are two neighboring nodes (connected by an edge) along the convex hull border and a third node very close to their joining line. For this reason, apart from possible subdivisions of the boundary of the convex hull with many equally spaced nodes, the use of *constrained Delaunay triangulation* would be an interesting choice.

In a Delaunay triangulation a *constraint* is a region "forbidden" to the triangulation process, following a closed polygonal passing through the *constrained nodes*.

The corresponding constrained Delaunay triangulation has been generated using the package *TRIPACK: Constrained Delaunay Triangulation Package* by R. J. Renka, as described in [16]. The Fortran 77 source file `src.f` and the test program `driver.f` can be download at the internet address `http://netlib.bell-labs.com/netlib/toms/` file `751.gz`. These two files are well commented and no problem arises in making examples that use the package. Simple examples of its use are presented at the Section 6.

Once the constraints are located, the rule to respect is simple: *all the unconstrained nodes must precede the constrained ones* (see Example 6.2).

TRIPACK uses the following conventions.

1. The nodes of every constraint must have consecutive indices. Only the nodes defining the *outer border* are ordered in a clockwise way. In this way the area to be triangulated always lies on the right of the oriented constraint, while on its left there will be the constraint.
2. There is only one *exterior constraint*, that is the outer border.
3. Two different constraints cannot share a node.
4. The indices of the first node of every constraint must be stored in a suitable array. Using the notations as in TRIPACK, we shall refer to the total number of constraints as NCC and to the corresponding array as LCC. We also set LCC(NCC+1)=N+1, so that the ith constraint has LCC(i+1)-LCC(i) nodes i=1,...,NCC.

Once the Delaunay triangulation is built, which is always over the convex hull, we must solve the problem of deleting all triangles within constrained regions or connecting boundary nodes as will be shown in Figures 6 and 7 of Section 7.

3.1. Constrained region deletion: outline of the algorithm

The actual version of the algorithm works on the output as produced by TRIPACK by means the subroutine TRPRNT included in the package. An example of this output can be found in the Example 6.2 below. The format of the output produced by the subroutine TRPRNT follows the following syntax.

For each node the first column contains the progressive index, the second and third its x, y Cartesian coordinates. The successive columns contain the indices of the neighboring nodes listed in a counterclockwise way. If a node is on the boundary, the corresponding row ends with a zero. Thus, each triangle is stored three times, one for each row corresponding to its vertices. The first column of each row contains the index of the *base node*, followed by a couple of contiguous indices.

Our representation can be achieved by *preprocessing* the output produced by TRIPACK. The program that performs this step is `rk2tri` (see Sections 5 and 6).

For each node the first column is the progressive node index, while the successive ones are the neighboring node indices listed in a counterclockwise direction. In this context a node is called a *neighboring node of a base node* if it is the end of an edge exiting from the base node. If a node does not belong to any constraint (or if there are no constraints it does not belong to the convex hull border), the row ends with the first neighboring node index (see for example the second line in our representation in Example 6.2).

In what follows we adopt these definitions. A region formed by all the triangles having the same base node as a vertex, is called a *closed cell* otherwise it is said an *open cell*. Moreover we use the index -1 to end each row in the file of the triangulation.
The algorithm applied to remove constrained regions or a triangle performs the following further steps.

1. If the current node does not belong to any constraint and the cell is open it accepts all its row except the ending zero, otherwise it writes its index again at the end of the first neighboring node index. Anyway, it finishes with -1.

2. If the current node belongs to the ith constraint ($1 \leq i \leq \text{NCC}$) and the current row ends with a zero, i.e. the cell is *open*, on each triangle it recursively performs the following steps:

   2.1 For every triangle T belonging to the ith constraint ($1 \leq i \leq \text{NCC}$), starting from the left side, it considers its vertices ordered increasingly. The triangle is accepted if the third vertex lies on the right of the oriented segment joining the two other vertices, otherwise it rejects it. The *rejection* is done in the following way. If the process starts from the left side, it deletes the first neighboring node index from the current row, otherwise it deletes the last one[1].
   Once the search has been completed, the algorithm has to delete the ending zero introduced by the previous representation, and adds the improper index -1 at the end of each row. If the current node does not belong to the ith constraint ($1 \leq i \leq \text{NCC}$) and its corresponding row does not end with a zero (i.e. the cell is closed) it recursively examines all triangles defined by the row applying (2).

   2.2 It continues until the first rejection.

[1] Notice that this test can similarly be performed using the sign of the z component of the cross product of two vectors belonging to the (x, y) plane.

3. Then it rotates to the left of the neighboring node indices as many times as the rejected triangle has indices in positions not contiguous. Finally, it adds an ending zero in order to transform the closed cell in an open one and it continues as described at step (2).
   If all triangles are accepted, we add the first neighboring node index to the end of the row and the index -1.

4. The interpolation scheme

The class of functions we are looking for is that of the n-degree polynomial splines:

$$S_n(\tau) = \left\{ f \in C^1(\tau) \ : \ f_{|T} \in \mathbb{P}_n \ \forall T \in \tau \right\}, \tag{4}$$

or, more generally,

$$S_n(\tau_h) = \left\{ f \in C^1(\tau_h) \ : \ f_{|T} \in \mathbb{P}_n \ \forall T \in \tau_h \right\}, \tag{5}$$

where τ_h is the triangulation obtained by subdividing every triangle of τ in h sub-triangles and $\mathbb{P}_n$ is the space of the bivariate polynomials of degree $\leq n$.

This choice has two main advantages.

- The use of bivariate Bernstein polynomials, i.e.

  $$B^n_{i,j,k}(u, v, w) = \frac{n!}{i!\, j!\, k!} u^i v^j w^k \ , \quad i + j + k = n$$

  as a basis for polynomial splines. The *barycentric coordinates* u, v, w allow to define the restriction to a generic triangle $T \in \tau_h$ as a *non parametric polynomial Bézier patch* (cf. [11]):

  $$b_n(u, v, w) = \sum_{i+j+k=n} d_{i,j,k} B^n_{i,j,k}(u, v, w), \tag{6}$$

  that is a function of the control points $d_{i,j,k}$. In the linear case this net is know as the *Bézier net*.
- The C^1 continuity conditions between two patches $\mathbf{b}$ and $\mathbf{b}'$, are simply related to the *coplanarity* of the four control points defining the two patches.

As proved by Ženišek in [21,22], if we do not split τ into sub-triangles, the problem of C^1 interpolation is *solvable if and only if* $n \geq 5$ and if we know the partial derivatives at the nodes up to the order of 2. The Q_{18} element (cf. [3]), is a fifth degree polynomial identified by 18 coefficients instead of 21. Using the element Q_{18}, the C^1 continuity conditions among adjacent triangles are guaranteed by the property to possess a cubic normal derivative at the macro-triangles' edges, instead of a quartic. However, the derivative estimation process up to the second order is in this case a difficult task. This suggests the use of lower degree polynomial interpolation schemes. However this leads to an unavoidable subdivision of τ into micro-triangles. In particular, the Clough-Tocher element (cf. [5–7,11,15]) uses cubic polynomial patches obtained by subdividing each triangle in three parts by means of the barycentric point. This element is completely determined by the knowledge of the first partial derivatives at the vertices of each triangle and the cross derivatives at some point of each edge (for instance the midpoint). A *reduced* or *condensed* Clough-Tocher element can be obtained if the normal derivatives across edges are linear instead of quadratic: this is the one we used. In this case the C^1 continuity conditions between adjacent macro-triangles are guaranteed by the fact that the element owns linear normal derivative, instead of quadratic, along the macro-triangles' edges.

If we further decrease the degree it is possible to use a quadratic polynomial interpolation scheme based on the Powell-Sabin splitting [14,11,18]. This scheme has two different splittings of the triangles, also known as *A subdivision* and *B subdivision*. In the *A subdivision* each triangle is divided into six sub-triangles by means of the circumcentre and the connections with the vertices and the middle edge points. While, if the triangle has all interior angles lower than a fixed limit (usually 75 degrees) the triangle is subdivided into twelve sub-triangles by means of the barycenter, the connections with the vertices and the middle edge points and the connections among the three middle edge points: this is the *B subdivision*. In both cases, the C^1 continuity conditions between adjacent macro-triangles are guaranteed by the property that the normal derivative along macro-triangles is linear instead of piecewise linear.

4.1. The derivative estimation procedure

The technique employed for the derivative estimation is based on the minimization of the so called *energy functional* ϕ associated with the interpolation scheme. This functional represents the squares of the linearized main curvatures of the interpolant.

The procedure can be described formally as follows. Let N be the number of nodes on the domain D and let s be the number of derivatives required by the scheme for every node. Thus, for each triangle the scheme will require $S = 3 \cdot s$ derivatives and $N \cdot s$ derivatives for the whole domain D. It is possible to define a "standard" way to sort these unknowns so that we can refer to each of them as the ith element of a vector $\mathbf{v} \in \mathbb{R}^{N \cdot s}$. The array $\mathbf{v}$ is ordered in the following way.

- $v_i < v_j$ if v_i refers to a node belonging to D with an index smaller than or equal to the one of v_j.
- If those indices are equal, $v_i < v_j$ if v_i is relative to a derivation order smaller than the one of v_j.

- If those indices are still equal, $v_i < v_j$ if v_i has more derivations with respect to the x variable than v_j.

Let Q_T be the restriction of the interpolation function Q to the triangle T. The functional we want to minimize is:

$$\phi(\mathbf{v}) = \sum_{T\in D} \int_T \left(Q^2_{T,xx} + 2Q^2_{T,xy} + Q^2_{T,yy}\right)\, dx\, dy. \tag{7}$$

It has been proved [17,2,12] that it is possible to write

$$\phi(\mathbf{v}) = \frac{1}{2}\mathbf{v}^T A\mathbf{v} + \mathbf{g}^T\mathbf{v} + c,$$

A being a symmetric positive definite matrix. Therefore, $\phi(\mathbf{v})$ is a quadratic form which has one and only one minimum, $\overline{\mathbf{v}}$. Thus, the optimal estimation for the derivatives at the nodes corresponds to the unique solution of the linear system $A\overline{\mathbf{v}} + \mathbf{g} = 0$. If the domain D extends to the whole triangulation, the process is known as *global estimation* and has been discussed by Alfeld in [2]. However, if we deal with a great number of nodes, the resulting linear system, even if sparse, will be of too large a dimension and therefore will be hardly solvable due to memory space and computational time problems. To this end, we make use of a *local estimation* approach (cf. [17]). Originally applied only to the triangle edges, it allows us to restrict the domain D to the set of the triangles sharing a fixed node (its cell). This method provides an effective alternative to the estimation of the derivatives at a given node. In our study we implemented both the global and the local estimation techniques, proving that as far as execution times are concerned, there is a similar behavior when the number of nodes are less than or equal to 16, while for a large number of nodes the local estimation is faster than the global one (cf. [12]). As for the "quality" of the approximation, there are not significant differences for triangulations which do not have long and thin triangles (cf. [12]).

4.2. Improving the estimate

We have introduced a *variant* of the local derivative estimation process, which in most cases and especially for triangulations without long and thin triangles, leads to an energy $\phi(\mathbf{v})$ much smaller than the original local procedure.

Let $P_i = (x_i, y_i)$ be the node and g_i the corresponding gradient to be estimated. Let P_j, $j = 1,\dots,k$ be the set of its neighboring nodes (connected to P_i by an edge of the triangulation) and $g_{i,j}$ be the gradient at P_i obtained by the minimization of the functional ϕ over the cell whose base node is P_j so that $g_{i,i}$

is the gradient estimated at P_i by the original local procedure. Then, we simply set:

$$g_i = (1-q)\cdot g_{i,i} \, + \, q\cdot \frac{\sum_{j=1}^{k} \frac{1}{d_{i,j}^2} g_{i,j}}{\sum_{j=1}^{k} \frac{1}{d_{i,j}^2}}, \quad q \in [0,1] \tag{8}$$

where $d_{i,j}$ is the Euclidean distance between P_i and P_j.

5. The LABSUP package

Here we briefly describe all the programs that form the package LABSUP. The name LABSUP is the acronym of "**LA**boratory for **B**ivarate **SU**rfaces and **P**atches".
All the programs are written in C language except when the triangulation has some constraints: in this case the driver program which interfaces with the TRIPACK package has to be written in Fortran 77. They can be grouped in five main classes as described in Table 1.
Here we schematize the sequence to follow in using the programs.

1. The first step is the generation of the file `.dat`, that is a two or three dimensional file of points. For this purpose one can use the program `gen_rand` or simply write it by himself. It is worth to note that in the case the file has three dimensional points the z-coordinate is not considered by the algorithm since the triangulation is built over the convex hull of the projections of the data points onto the plane $z = 0$.

2. The second step is the generation of the Delaunay triangulation. This requires the knowledge of possible constraints. If there are constrained regions, one must use the TRIPACK package (by linking the file `src.f` to a driver Fortran program as described in the file `README.1ST` supplied with the software) to get the constrained Delaunay triangulation, then preprocessed it with the program `rk2tri`. Otherwise one simply use the program `do_trian`. Moreover, the program `do_trian` allows to interactively change an existing triangulation providing a shell which permits to add a new node and to view the new modified triangulation.
   At this step we have so generated the triangulation file which has suffix `.tri`. If one desires to have more information about the triangulation, the programs `tri_qlty` and `info_vic` should be used.

3. Derivatives estimation can be accomplished by using one of the programs listed in Table 2 where we also propose a suffix for the generated file.

4. To compute the functional energy for the C^1 interpolating surface one can use the programs listed in Table 1, applied both to global and local derivatives

Triangulation	
`do_trian`	Builds the Delaunay triangulation over the convex hull of the projections of the data points onto the plane $z = 0$.
`rk2tri`	Converts a triangulation file in TRIPACK format in the same format of `do_trian`.
Derivatives	
Global estimators	The domain of minimization is the whole triangulation.
`do_glder`	for the element Q_{18},
`cltglder`	for Clough-Tocher splitting,
`pwsglder`	for Powell-Sabin splitting.
Local estimators	The domain of minimization is a *cell*
`do_der`	for the element Q_{18},
`clt_der`	for Clough-Tocher splitting,
`clt_der2`	applies formula (10) to the Clough-Tocher splitting,
`pws_der`	for Powell-Sabin splitting.
Energy Functional	
`funz_clt`	Computes the energy of Clough-Tocher elements with normal linear derivatives.
`funz_q18`	Computes the energy for Q_{18} elements.
`funz_pws`	Computes the energy of Powell-Sabin elements of A-type and B-type.
Miscellaneous	
`gen_rand`	Generates a file of random 3D-points or user-defined.
`tri_qlty`	Provides information about angles and areas of the triangles.
`info_vic`	Provides the maximum and the average number of nodes for a given cell.
Visualization	
`labsup3d`	It is the program of the package which allows to visualize and to manipulate surfaces by using methods based on the underlying Delaunay triangulation. **Note**: it requires the use of the `OpenGL` or `Mesa` graphics libraries.

Table 1
The programs of the LABSUP package.

estimations files. The result is a real number which measures the smoothness of the resulting surface.

5. The visualization of the surface and/or patches, the underlying triangulation and all the managing process will be done by the interactive program

Name	Patch	suffix
do_glder	Global Q_{18}	.ggd
do_der	Local Q_{18}	.g_d
cltglder	Global Clough-Tocher	.gct
clt_der	Local Clough-Tocher	.ct
clt_der2	Improved Local Clough-Tocher	.ct2
pwsglder	Global Powell-Sabin	.gpw
pws_der	Local Powell-Sabin	.pw

Table 2
Derivative estimation modules

`labsup3d`. For the visualization process it makes use of the *OpenGL* graphics library or the *free OpenGL* library known as *Mesa*. For more details on Mesa Graphics Library we refer to the B. Paul's documentation: **Mesa** - *The free OpenGL work-alike library.* The documentation and the software are available at the address: `http://www.mesa3d.org`.

We conclude this Section by noticing that the source files of all programs and examples are available in a same directory. The accompanying documentation file, `README.1ST`, gives a summary of all the features concerning the files in the package together with a description of the synopses for their use. Moreover for more details on using the graphical interface of `labsup3d`, we supplied a PostScript file `labsup.ps`, which also describes how to use it.

6. Numerical examples

We present here some numerical examples that explain how the package works. For all these examples we provide the corresponding `.dat` files. Again the file `README.1ST` explains how to produce all remaining data files (triangulation files, derivatives files) and how to compute the energy of the surface, to display and manage the surface.

6.1. Example 1: Franke's function and saddle function

We describe the behavior on two well-known test functions: the *Franke function*

$$\begin{aligned} f(x,y) = {} & \frac{3}{4}e^{-\frac{(9x-2)^2+(9y-2)^2}{4}} + \frac{3}{4}e^{-\frac{(9x+1)^2}{49} - \frac{9y+1}{10}} + \\ & \frac{1}{2}e^{-\frac{(9x-7)^2+(9y-3)^2}{4}} - \frac{1}{5}e^{-(9x-4)^2-(9y-7)^2}, \end{aligned} \tag{9}$$

and the *saddle function*

$$s(x,y) = \frac{1.25 + \cos(5.4y)}{6 + 6(3x-1)^2},$$

considered also in [17,13]. The package supplies the files `exam1_f.dat` and

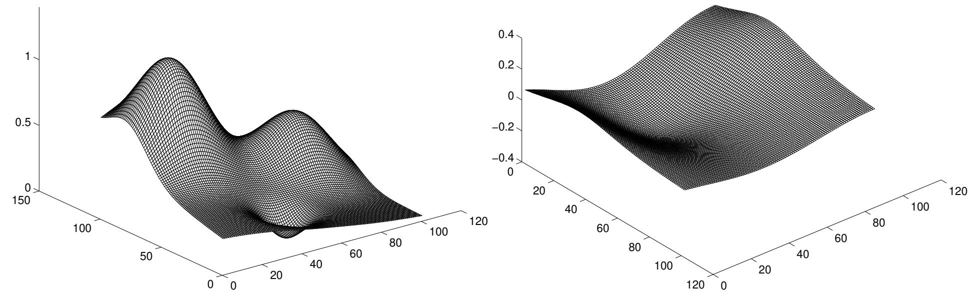

Figure 1. The Franke function and the saddle function

`exam1_s.dat` which consist of 149 points belonging to the unit square with 20 points equally spaced on the boundary and the remaining ones scattered in the interior of the square (generated with the program `gen_rand`). The corresponding triangulation files, `exam1_f.tri` and `exam1_s.tri` supplied with the package, are generated with the program `do_trian`.

For the function f, the minimization of the Clough-Tocher energy functional leads to the energy values computed by using the program `funz_clt` applied to the derivatives files listed in the Table below. These files could be generated as explained in the file `README.1ST` at section 14.

626.040	original local procedure	derivatives file `exam1_f.ct`
363.010	modified local procedure with $q = 0.5$	derivatives file `exam1_f.ct2`
229.892	global procedure	derivatives file `exam1_f.gct`

For the function s, the minimization of the Clough-Tocher energy functional leads to the energy values:

40.010	original local procedure	derivatives file `exam1_s.ct`
13.123	modified local procedure with $q = 0.5$	derivatives file `exam1_s.ct2`
9.532	global procedure	derivatives file `exam1_s.gct`

Note that the value $q = 0.5$ is chosen since it gave the minimum value for the energy. We also notice that formula (8) is a particular instance of a more general one used in [10, Eq. (2.31)], that is

$$g_i = (1-q)\cdot g_{i,i} + q\cdot \frac{\sum_{j=1}^{k} \frac{\alpha e^{-\beta d_{i,j}^2}}{d_{i,j}^{\gamma}} g_{i,j}}{\sum_{j=1}^{k} \frac{\alpha e^{-\beta d_{i,j}^2}}{d_{i,j}^{\gamma}}}, \quad q \in [0,1]\ ; \tag{10}$$

when $\alpha = 1,\ \beta = 0,\ \gamma = 2$.

6.2. *Example 2: a simple constrained region*

The aim of this very simple example is to show the difference between the TRIPACK data structure and our data structure. Let us consider the following set of points in the plane $z = 0$ (stored in the file `exam2.dat`).

indices	abscissae	ordinates
1	0.5	0.25
2	0.25	0.75
3	0.0	0.0
4	0.0	1.0
5	1.0	1.0
6	1.0	0.0
7	0.25	0.5
8	0.75	0.5
9	0.50	0.75

Table 3
Points on the unit square

The TRIPACK syntax requires that the non-constraint nodes must be at the beginning. We choose the points (0.5, 0.25) and (0.25, 0.75) as two unconstrained nodes. The remaining 7 nodes forms two constrained curves: the first one, whose nodes indices are 3 up to 6 is the outer border that is the vertices of the unit square, the second one, whose nodes have indices 7 up to 9 is the inner triangle.

As displayed in Figure 2, this set of points forms a square with a triangular hole. The two further unconstrained nodes are pointed out by small circles.

TRIPACK produces the following output, in which the spaces between successive lines have been eliminated for conciseness purposes (this is file `exam2.rk`).

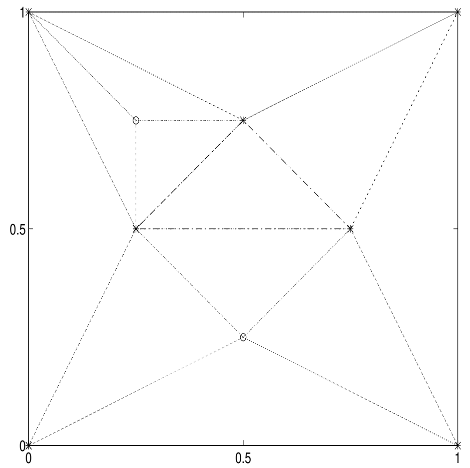

Figure 2. Region with two constraints: the border and the inner triangle and its constrained Delaunay triangulation.

```
                    Adjacency sets,    N =      9

Node     X(Node)        Y(Node)          Neighbors of Node
  1    .500000E+00    .250000E+00    6     8     7     3
  2    .250000E+00    .750000E+00    9     4     7
  3    .000000E+00    .000000E+00    6     1     7     4    0
  4    .000000E+00    .100000E+01    3     7     2     9    5    0
  5    .100000E+01    .100000E+01    4     9     8     6    0
  6    .100000E+01    .000000E+00    5     8     1     3    0
  7    .250000E+00    .500000E+00    8     9     2     4    3    1
  8    .750000E+00    .500000E+00    6     5     9     7    1
  9    .500000E+00    .750000E+00    5     4     2     7    8

NB =    4  Boundary Nodes    NA =     20 Arcs     NT =    12 Triangles
NCC =   2  Constraint Curves
              3     7
```

Some more words about the output produced by TRIPACK. First of all the user may choose if to print the coordinates of the points. `N` is the number of nodes, `NB` is the number of boundary nodes, `NA` is the number of arcs, `NT` is the number of triangles and `NCC` is the number of constraint curves. The numbers below each component of `NCC` are simply the indexes of the first node in the constraint curves.

The corresponding data structure produced by our conversion program `rk2tri`, stored in the file `exam2.tri`, consists of two parts. The first part starts with the uppercase C and contains the list of the indices of the nodes followed by the list of the indices of their neighbors. The second part starts with the uppercase B and lists the indices of the nodes of the outer border that originally were not included in the constraints of the triangulation. In the case when the constraints include the outer border then this part will result empty, as in this case.

```
C
        0       5       7       6       2       5      -1
        1       8       3       6       8      -1
        2       5       0       6       3      -1
        3       2       6       1       8       4      -1
        4       3       8       7       5      -1
        5       4       7       0       2      -1
        6       8       1       3       2       0       7      -1
        7       6       0       5       4       8      -1
        8       7       4       3       1       6      -1
B
```

6.3. Example 3: a constrained surface

We consider the data points in the file `exam3.dat` of our package, which is the evaluation of R. Franke's function on 147 points as follows: 119 random points into the interval [0,1]x[0,1] (generated by `gen_rand` program), 7 points defining an inner constraint hole that is the first constraint curve, and further 21 points defining the outer border which is the second constraint curve. This file is obtained from file `exam1_f.dat` by suppressing lines 89 and 112 from it and making a permutation of the remaining 147 lines in order to have the constraints at the end of the file, as required by the package TRIPACK (used by our package for generating the constrained triangulation).

The above suppressed lines introduce an inner hole and a 'corner-cutting' in the triangulation as shown in Figure 3. Both images in Figure 3 have been obtained with the program `labsup3d` and stored first with the X Window dumping utility `xwd` and then saved in PostScript format with the interactive image display for the X Window `xv`, after a simple changing in the background color. It is worth noting, that the actual version of TRIPACK allow to generate the PostScript file of the triangulation. This can be accomplish by using the subroutine TRPLOT included in the package. The main difference between our triangulation file are again the constraints regions that are pointed out by dashed lines instead of continuous ones. In Figure 4 we show the triangulation of this example as produced with the subroutine TRPLOT, with the local logical variable ANNOT setted to .FALSE. (this does not print the annotations about the window extremes below the plot). The corresponding file `exam3.eps` is generated automatically by the program `exam3` of our package.

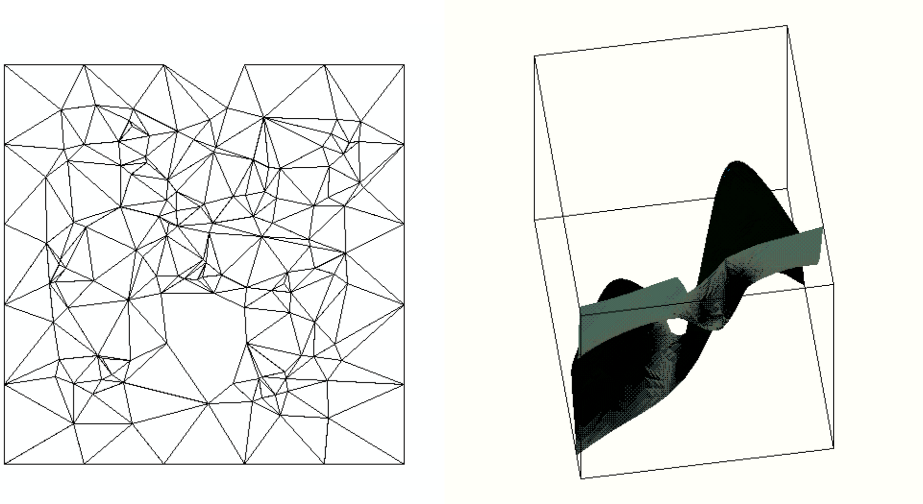

Figure 3. Left side: Delaunay constraint triangulation. Right side: a view from below of the surface by using global derivative estimation based on the element Q_{18}.

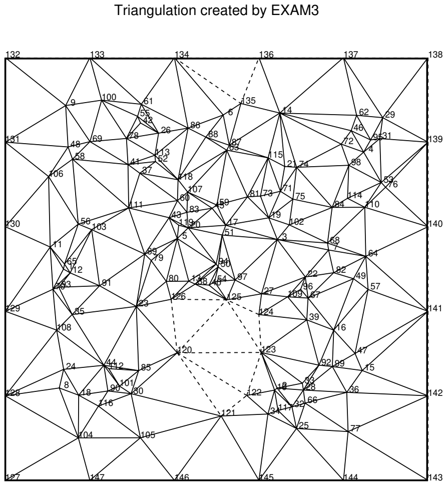

Figure 4. Delaunay constraint triangulation as obtained with the subroutine TRPLOT of the TRIPACK package.

7. The lagoon of Venice's bed: description of the application and graphical results

The initial data consisted of about $200,000$ points divided between *interior* points and *boundary* ones. This amount was reduced to $65,636$ due to redundancy of boundary data points around islands and the border of the whole lagoon. These points have been supplied by the *"Ufficio Idrografico e del Magistrato alle Acque"* of the city of Venice in 1969-1970, and grouped in about 150 maps, in scale 1:5000, corresponding to regions with a particular name. A copy of the map produced in the sixties by the *Italian Military Geographical Institute* in scale 1:250000, is shown in Figure 5 for completeness. Following the requirements outlined in

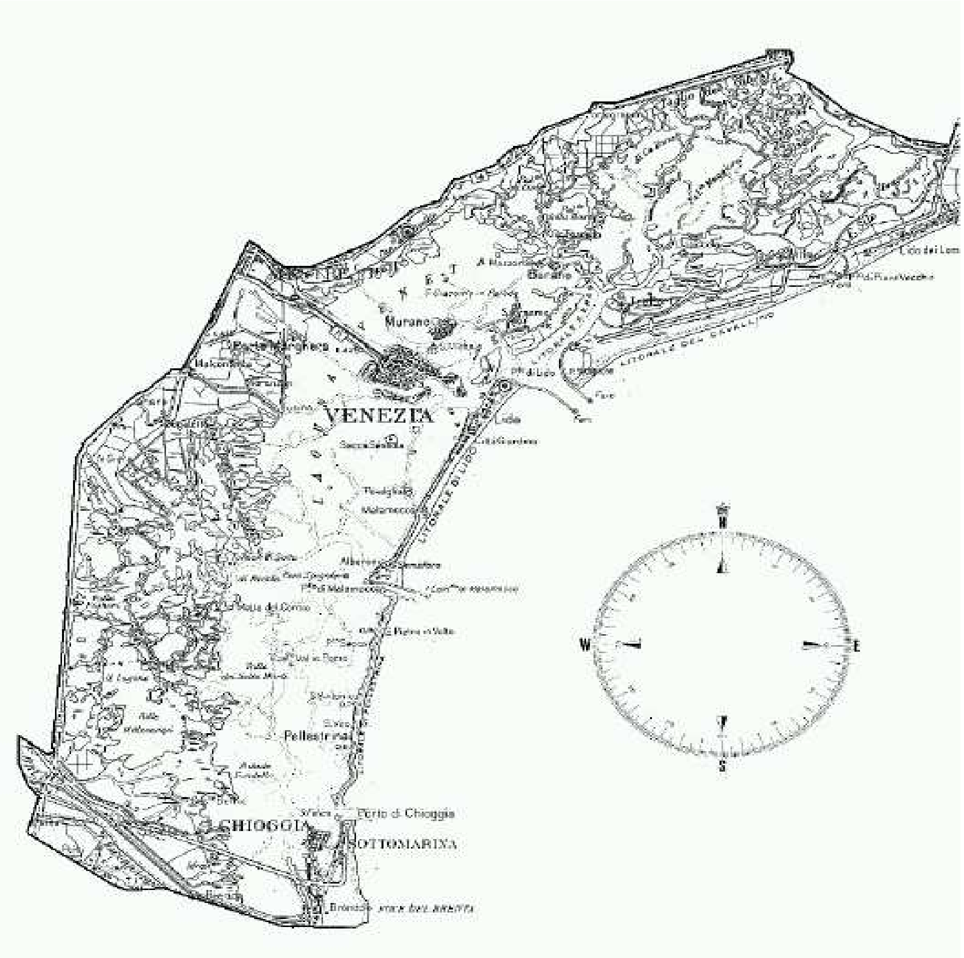

Figure 5. The Lagoon of Venice as reproduced by the Italian Military Geographical Institute.

Section 3, the boundary points constitute 38 constraints. One of them is the *outer* border, while the remaining 37 refer to the boundary of the inner islands. We used reduced Clough-Tocher elements and the corresponding gradients have been estimated by the minimization of the energy functional (7), associated to the reduced Clough-Tocher element, by our *local* approach (by the program `clt_der`). Due to the width of the area to be reproduced, i.e. about $1,500$ square kilometers, we have restricted the pictures to the most famous areas: the island of the city of Venice and the island of "La Giudecca". The Figures 6 and 7 are the triangulations of the lagoon bed around Venice's island. The Figure 6 is a zoom of the triangulation as obtained with the TRIPACK where is visible the island of "La Giudecca" and the "Canal Grande". The other Figure 7 is the triangulation as obtained with LABSUP around the island of the town of Venice, the island "La Giudecca" and some other smaller islands. This triangulation is obtained from the constrained one as generated with the TRIPACK package and post-processing it with the program `rk2tri`. It is clear from the figure that the 'deletion algorithm' impletemented in the program `rk2tri` has cancelled the spurious lines evident in Figure 6.

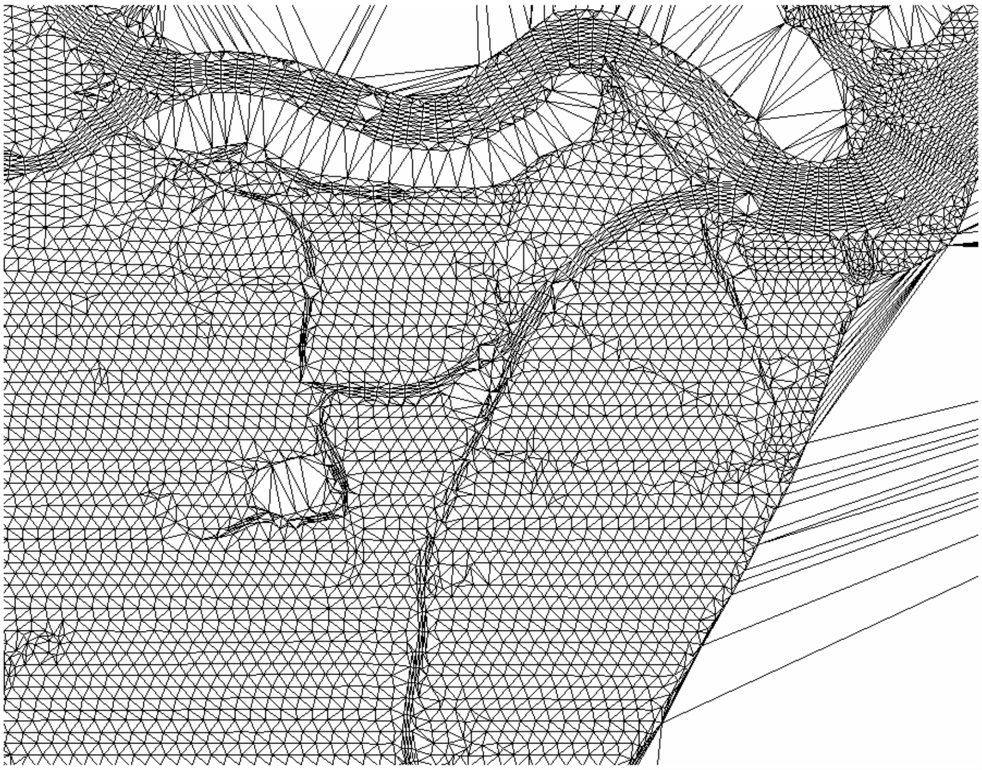

Figure 6. The Delaunay triangulation of the lagoon of Venice by the original TRIPACK: a zoom around the "Canale della Giudecca". The spurious lines left by TRIPACK are evident.

We conclude this Section observing that at the CNR-ISDGM of Venice, cf. [4],

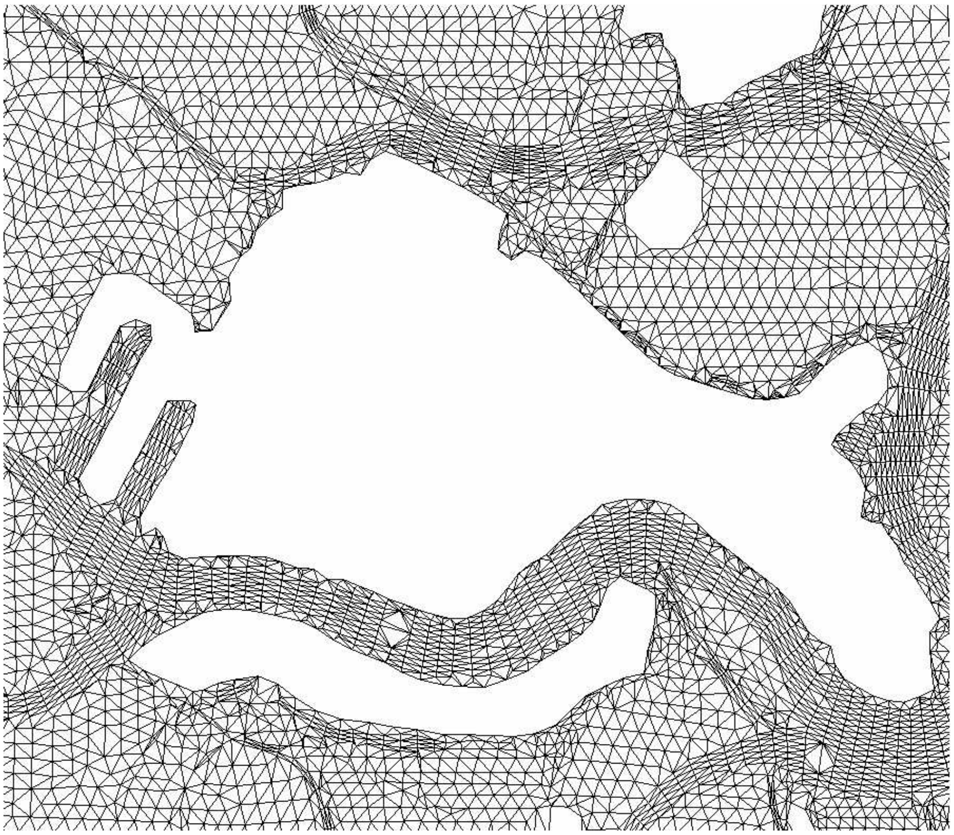

Figure 7. A zoom of the constrained Delaunay triangulation around the Island of the City of Venice obtained with our package.

they developed a package that displays the bed of the lagoon as a contour plot by different colors depending on the depth. Our method is more general since it allows us to visualize the bed of the lagoon and its contour plot. Moreover our method can be used to reproduce an interpolating surface scattered or sampled on a bunch of points.

Acknowledgements. This work has been carried out with the support of the M.U.R.S.T., Ministry of Research and Technology, project "*Advanced Numerical Methods for Scientific Computing*", `Scientific Research Programs of Relevant National Interest` scheme, year 1998. The third author did part of this work during his Master's thesis at the Dipartimento di Matematica Pura e Applicata, University of Padua, in 1995.

We also thank the Software Editor for her suggestions that improved both the manuscript and the software.

References

[1] H. Akima, On estimating partial derivatives for bivariate interpolation of scattered data.*Rocky Mountain J. Math.*, 14(1):41–52, 1984.

[2] P. Alfeld, Derivative generation from multivariate scattered data by functional minimization.*Comput. Aided Geom. Design*, 2:281–296, 1985.

[3] R. E. Barnhill and G. Farin, C^1 quintic interpolation over triangles: Two explicit representations.*Internat. J. Numer. Methods Engrg.*, 17(12):1763–1778, 1981.

[4] A. Bergamasco and A. Tomasin, Batimetria della Laguna di Venezia - dati informatizzati del CNR-ISDGM. Data available at `http://157.138.180.25/ivsla_slv/miscella.html`, 1996.

[5] P. G. Ciarlet, Sur l'element de Clough et Tocher. *RAIRO Analyse Numerique*, R2:19–27, 1974.

[6] P. G. Ciarlet, Interpolation error estimates for the reduced Hsieh-Clough-Tocher triangles. *Math. Comp.*, 32:335–344, 1978.

[7] P. G. Ciarlet, Basic error estimates for elliptic problems. In P.G. Ciarler and J.L. Lions editors, *Handbook of Numerical Analysis, Vol. II: Finite Element Methods (part 1)*, 19–351, 1991. North Holland, Amsterdam.

[8] A. K. Cline and R. L. Renka, A storage-efficient method for construction of a Thiessen triangulation. *Rocky Mountain J. Math.*, 14(1):119–139, 1984.

[9] A. K. Cline, R. J. Renka, A constrained two-dimensional triangulation and the solution of closest node problems in the presence of barriers. *SIAM J. Numer. Anal.*, 27:1305–1321, 1990.

[10] S. De Marchi, On computing derivatives for $\mathcal{C}^1$ interpolation schemes: an optimization. *Computing*, 60(1):29–53, 1998.

[11] G. Farin, Triangular Bernstein-Bézier patches. *Comput. Aided Geom. Design*, 3:83–127, 1986.

[12] D. Fasoli, Interpolazione C^1 di dati "scattered" e procedure per la stima delle derivate parziali. "Laurea"'s thesis, Dipartimento di Matematica Pura ed Applicata, Università degli Studi di Padova (In Italian), 1995.

[13] T. N. T. Goodman, H. B. Said, and L. H. T. Chang, Local derivative estimation for scattered data interpolation. *Appl. Math. Comput.*, 68:41–50, 1995.

[14] M. J. D. Powell and M. A. Sabin, Piecewise quadratic approximations on triangles. *ACM Trans. Math. Software*, 3(4):316–325, 1977.

[15] E. Quak and L. L. Schumaker, Cubic spline fitting using data dependent triangulations. *Comput. Aided Geom. Design*, 7:293–301, 1990.

[16] R. J. Renka, Algorithm 751; TRIPACK: a constrained two-dimensional Delaunay triangulation package. *ACM Tran. Math. Soft.*,22(1):1–8, 1996.

[17] R. J. Renka and A. K. Cline, A triangle-based C^1 interpolation method. *Rocky Mountain J. Math.*, 14(1):223–237, 1984.

[18] P. Sablonière, Error bounds for Hermite interpolation by quadratic splines. *IMA J. Numer. Anal.*, 7:495–508, 1987.

[19] L. L. Schumaker, Triangulation methods. In C. K. Chui, L. L. Schumaker, and L. Utreras, editors, *Topics in Multivariate Approximation*, 219–273, 1987. Academic Press, New York.

[20] E. Secco, Un modello ad elementi finiti per lo studio di problemi di acque basse: un'applicazione alla laguna di Venezia. *Ph.D. Thesis*, University of Padua, 1995 (In Italian).
[21] A. Ženišek, Interpolation polynomials on the triangle. *Numer. Math.*, 15:283–296, 1970.
[22] A. Ženišek, A general theorem on triangular finite C^m-elements. *RAIRO Inform. Théor. Appl.*, 8(2):119–127, 1974.