LABSUP*

A LAboratory for Bivariate $\mathcal{C}^1$ SUrfaces and Patches

User's Guide

Maria MORANDI CECCHI[†],

Stefano DE MARCHI[‡],

Damiano FASOLI[§]

July, 1999

*This work has been carried out with the support of the M.U.R.S.T., Ministry of University, Scientific Research and Technology, project "*Advanced Numerical Methods for Scientific Computing*", `Scientific Research Programs of Relevant National Interest` scheme, year 1998.

[†]Dipartimento di Matematica Pura ed Applicata, Università di Padova, Via Belzoni 7, 35131 Padova (Italy). **e-mail**:*mcecchi@math.unipd.it*

[‡]Dipartimento di Matematica e Informatica, Università di Udine, Via delle Scienze, 206 - 33100 Udine (Italy). **e-mail**:*demarchi@dimi.uniud.it*

[§]Verona Software, S.r.l., Via Trento 18 - 37124 Verona (Italy). **e-mail**:*vrsoft@sis.it*

Contents

1 Introduction

LABSUP is a package designed for scattered data interpolation with $\mathcal{C}^1$ triangular polynomial patches. We present here a short *User's Guide* that describes the main features of the package and especially how to use it for the representation of $\mathcal{C}^1$ interpolating surfaces to scattered data. In particular we shall stop a little more on the programs that make up the package and on the commands that the user can choose in order to produce and manage the resulting surface.

All programs are written in C language except for the examples whose triangulation has some constraints. In this particular case it is necessary to use a *driver* file in Fortran 77 which allows to link to the TRIPACK package (for more details see file README.1ST).

With the package we provide a `Makefile` that describes the relationships among files, compiles and links all the programs by the utility `make`. Remember that in order to use the Makefile on your own Operating System, you must change it as appropriate.

The package was originally intended to be part of a project aimed to study different aspects and phenomena acting on the Venice's lagoon, in particular the visualization of its bed. This latter is important for two main reasons: firstly it represents an interesting bidimensional *scattered data* test problem (it consisted of about 132000 scattered points that could be reduced to 65636 after a careful analysis of their distribution) and secondly it allows to have a graphical representation of the bed of the lagoon in order to study some details related to the change in the depth due to the tide, the pollution and so on. This problem has then been solved by using typical techniques from scattered data approximation by smooth functions. The requirement of the $\mathcal{C}^1$ continuity is set in order to obtain a pleasant and smooth aspect of the lagoon bed.

2 Description of the programs

The programs of the package may be classified into five main classes, depending on the particular task performed. In this section we present the first four classes that can be also used separately. In fact, the construction of the Delaunay triangulation, the generation of the partial derivatives and the computation of the energy associated to a surface are steps that the users can perform without displaying the resulting surface. For this reason we have chosen to present in the next Session the program `labsup3d` designed for visualizing and managing surfaces.

2.1 Triangulation programs

Let $(x_i, y_i, z_i),\ i = 0, ..., N-1$ be N points in the space $I\!R^3$. Let us consider their projections (x_i, y_i) onto the plane $z = 0$. τ is a triangulation of these points, if it is a set of triangles having all these points, and only them, as vertices so that two distinct triangles intersect at most in a common edge or in a common vertex. The domain of the triangulation, that is the area of the plane covered by its triangles, is usually the convex hull of the points, also called nodes of the triangulation. Among all possible triangulations of these points, the Delaunay triangulation τ is characterized by the following property: all circumcircles of the triangles of τ do not contain any node inside. This triangulation, that is not uniquely defined if there are at least two triangles with the same circumcircle, maximizes the smallest angle among all the angles of all triangles of the triangulations of the same points (over the same domain).

The program `do_trian` builds the Delaunay triangulation over the convex hull of the projections of the interpolation points onto the plane $z = 0$. On the command window one should type

```
do_trian file1 input_flag [file2] [-change]
```

where `file1` is the data file made up of 2-dimensional points or 3-dimensional points. The coordinates of the points may be separated by any combination of spaces, tabs and carriage returns, even though it is recommended to use a row for each point. The program works properly only if there are no points repeated twice or more. We suggest to use the suffix `.dat` for this file.

`input_flag` may be '-i2' or '-i3': use '-i2' for (x, y) coordinates in file1 or '-i3' for a (x, y, z) coordinates in file1 (in this case the z-coordinate is not considered by the algorithm for the reason explained above).

`file2` is the Delaunay triangulation data file generated by this program over the convex hull of the input points, also known as nodes (omit it to use standard output). The results should be stored in a file with suffix `.tri`. This file has three sections.

- The first, starting with a single capital letter C in the first row, lists in subsequent blocks separated by -1 the index of a node and the indices of its neighbors in a counterclockwise order. If the node does not belong to the boundary of the convex hull, the first node in the list of its neighbors is repeated again before the separating value -1.

- The second section, starting with a single capital letter B alone in a row, lists the edges belonging to the boundary of the convex hull of all nodes in a counterclockwise direction using the following 5-columns matrix format:

  column 1: index of starting node of the edge,
  column 2: index of ending node of the edge,
  column 3: index of the first vertex of the triangle ownering the edge,
  column 4: index of the second vertex of the triangle ownering the edge,
  column 5: index of the third vertex of the triangle ownering the edge.

- The third section, starting with a single capital letter T in a row, list all the triangles of the triangulation built using the following $N \times 12$ matrix, where N is the number of points and the 12 columns have the following meaning.

  column 1: index of the first vertex of the triangle, V1;
  column 2: index of the second vertex of the triangle, V2;
  column 3: index of the third vertex of the triangle, V3;
  column 4: index of the first vertex of the neighboring triangle along the edge V1V2, or -1 if this triangle does not exist;
  column 5: index of the second vertex of the neighboring triangle along the edge V1V2, or -1 if this triangle does not exist;
  column 6: index of the third vertex of the neighboring triangle along the edge V1V2, or -1 if this triangle does not exist;
  column 7: index of the first vertex of the neighboring triangle along

the edge V2V3, or -1 if this triangle does not exist;

column 8: index of the second vertex of the neighboring triangle along the edge V2V3, or -1 if this triangle does not exist;

column 9: index of the third vertex of the neighboring triangle along the edge V2V3, or -1 if this triangle does not exist;

column 10: index of the first vertex of the neighboring triangle along the edge V3V1, or -1 if this triangle does not exist;

column 11: index of the second vertex of the neighboring triangle along the edge V3V1, or -1 if this triangle does not exist;

column 12: index of the third vertex of the neighboring triangle along the edge V3V1, or -1 if this triangle does not exist.

If the fourth parameter `-change` is given the program reads the triangulation stored in `file2` and provides the user with an interactive shell supporting the following commands:

`add` : Add a new node
`ptlist`: View the current nodes and their indices.
`exit` : Save in the specified files the new data and the new triangulation and exits from the interactive shell.

Since, constrained Delaunay triangulations are often more useful, the user should use the TRIPACK package (cf. [12]) for building the triangulation for such domains. In this case the user must take care of the file format produced by the TRIPACK. Indeed, its format is different from the format of our triangulation file. Therefore, it is necessary to preprocess the triangulation file produced by the TRPRNT subroutine of TRIPACK, which has suffix `.rk`, in our format. In the file README.1ST at part 3, there is the description of the Example 3 which has a domain with two constraints. In that description we pointed out how to produce the file in TRIPACK format and how to translate it to our format. For this purpose the program `rk2tri` should be used. It takes in input a file in TRIPACK format and produces a file in the format suitable for our package. The syntax is:

```
rk2tri file1 file2
```

where `file1` is the triangulation file containing the output generated by the subroutine TRPRNT of the TRIPACK package (see file `src.f` for more details) while `file2` is the conversion of `file1` into the triangulation file in the format generated by the program `do_trian`. This file has three sections, as explained above for the program `do_trian`. In this case `file2` uses only this first section and the heading of the second section, that is a single capital letter B in the row. During the conversion the program removes the triangles belonging to constrained regions in order to generate the triangulation only on the required domain.

2.2 Derivative estimators

For scattered data $\mathcal{C}^1$ interpolation problems, derivatives at the data points of the triangulation are not ever known. If the domain D of minimization is the whole triangulation, we say that they perform a *global* estimation otherwise we refer to a *local* estimation.

2.2.1 Global estimators

Let $(x_i, y_i, q_i),\ i = 0, ..., N-1$ be N points in the space $I\!R^3$. Let us consider their projections (x_i, y_i) onto the plane $z = 0$. τ is a triangulation of these points, that is a set of triangles having all these points, and only them, as vertices so that two distinct triangles intersect at most in a common edge or in a common vertex. The aim is to interpolate the points (x_i, y_i, q_i) with a $\mathcal{C}^1$ function of two variables.

- The Q_{18} interpolation scheme builds such a function using a Q_{18} polynomial patch on each triangle, assuming to know the first and second derivatives at each vertex of the triangulation τ. The program computes the parameters which define the Q_{18} patch (in the following we shall refer to them as $qix, qiy, qixx, qixy, qiyy,\ i = 0, ..., N-1$, or more often as to the $5N$-dimensional vector v) in the "smoothest" possible way for a Q_{18} interpolating function over the whole triangulation τ. To evaluate the "smoothness" of the Q_{18} interpolating function, we minimized the functional

$$\phi(v) = \sum_{t \in \tau} \int_t (Q_{xx}^2 + 2\,Q_{xy}^2 + Q_{yy}^2)dxdy\ , \qquad (1)$$

  with respect to v. This is equivalent to solve

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

  which is minimum when v solves the system

$$Av + g = 0\ .$$

  The program computes

$$c = \phi(0),\ A,\ g_i = \phi(e_i) - \frac{1}{2}\,a_{ii} - c\ ,$$

  where e_i is the i-th vector of the canonical basis of $I\!R^{5N}$, and solves the resulting linear system with the Cholesky factorization.

- The Clough-Tocher's interpolation method builds up such a function using a polynomial patch on *each* triangle, provided that one knows the first derivatives at each vertex of the triangulation τ. The Clough-Tocher's polynomial element splits a triangle in three sub-triangles (using the barycenter) over which it builds three cubic polynomial patches. The program computes these partial derivatives (in the following we shall refer to them as to $qix, qiy\ ,\ i = 0, ..., N-1$, or more often as to the $2N$-dimensional vector v) in the "smoothest" possible way for a Clough-Tocher interpolating function over the whole triangulation τ. To evaluate the "smoothness" of a Clough-Tocher interpolating function we have again minimized the functional (1).

- The Powell-Sabin (PS) interpolation method builds up such a function using a Powell-Sabin polynomial patch on each triangle, assuming that the first and second partial derivatives are known at each vertex of the triangulation τ. The Powell-Sabin polynomial element is based on two different splittings of the triangle (the macro-triangle) so that we can have two types of PS-elements: A-type and B-type. This latter if the macro-triangle has an angle greater than 75 degrees.

The Powell-Sabin polynomial element of A-type, splits a triangle into six sub-triangles (using the circumcenter, the vertices of the macro-triangles and the middle edge points) over which it builds six quadratic polynomial patches.

The Powell-Sabin polynomial element of B-type, splits a triangle into twelve sub-triangles (using the barycenter, the vertices of the macro-triangles, the middle edge points and the intersections between lines connecting them and lines connecting the barycenter and the vertices of the macro-triangle) over which it builds twelve quadratic polynomial patches.

The aim of this program is to compute such parameters (in the following we shall refer to them as to qix, qiy. $i = 0, ..., N-1$, or more often as to the $2N$-dimensional vector v) in the "smoothest" possible way for a Powell-Sabin interpolating function over the whole triangulation τ.

To evaluate the "smoothness" of a Powell-Sabin interpolating function we have again minimized the functional (1).

The programs `cltglder` (global Clough-Tocher), `do_glder` (global Q_{18}) and `pwsglder` (global Powell-Sabin) minimize the `energy functional` (1) on the whole triangulation producing the required estimates of the derivatives. The synopses are:

```
do_glder file1 file2 [file3] (Q18)
cltglder file1 file2 [file3] (Clough-Tocher)
pwsglder file1 file2 [file3] (Powell-Sabin)
```

where `file1` is the file of the data points, `file2` is the triangulation file in the format generated by `do_trian` or `rk2tri` and `file3` is the file of the estimated derivatives, where for each the N points of the triangulation there are 2 or 5 columns depending on the derivatives required. For compatibility, whenever second derivatives are not required, we have filled with zeros the positions corresponding to the second derivatives, i.e. the columns 3, 4 and 5. If `file3` is omitted the results will be displayed on the commend window. We suggest to use the following suffixes for `file3`: `.ggd` for the element Q_{18}, `.gct` for the Clough-Tocher's element and `.gpw` for the Powell-Sabin's element.

2.2.2 Local estimators

We start again with a set of N points in $\mathbb{R}^3$. These programs compute the partial derivatives up to the order two at each point (x_i, y_i, q_i), $i = 0, ..., N-1$.

- For the element Q_{18} we refer to them as to $qix, qiy, qixx, qixy, qiyy$, $i = 0, ..., N-1$ or simply as to the $5N$-dimensional vector v. For every fixed i, $i = 0, ..., N-1$ the program computes the vector v in the "smoothest" possible way over the triangulation $\tau loc(i)$ made up by all triangles sharing the i-th interpolation point, V_i (that is the cell centered in V_i). Then it considers only the values $qix, qiy, qixx, qixy, qiy$ where i is now fixed. The criteria considered to evaluate the "smoothness" of a Q_{18} interpolating function is the minimization of the functional

$$\phi(v_{loc(i)}) = \sum_{t \in \tau_{loc(i)}} \int_t (Q_{xx}^2 + 2\, Q_{xy}^2 + Q_{yy}^2) dx\, dy \tag{2}$$

with respect to $v_{loc(i)}$, the same of v when τ becomes $\tau loc(i)$.

Then, for every fixed $i, i = 0, ..., N-1$, this is equivalent to solve

$$\phi(v_{loc(i)}) = \frac{1}{2}\, v_{loc(i)})^T\ A\ v_{loc(i)}) + g^T\ v_{loc(i)}) + c$$

is minimum when $v_{loc(i)})$ solves the system

$$A\ v_{loc(i)} + g = 0.$$

Let M be the number of nodes in the cell $\tau_{loc(i)}$. The program computes

$$c = \phi(0),\ A,\ g_j = \phi(e_j) - \frac{1}{2}\, a_{jj} - c,$$

where e_j is the j-th vector of the canonical basis of $I\!R^{5M}$, and solves the resulting linear system with the Cholesky factorization. Hence, it considers only the parameters relative to V_i, the center of $\tau_{loc(i)}$.

- For the Clough-Tocher (CT) polynomial patch the program simply computes the parameters assuming that the first and second partial derivatives are known at each vertex of the triangulation τ. As above, we shall refer to them as to $qix, qiy,\ i = 0, ..., N-1$ or simply to the 2N-dimensional vector v. For every fixed i, $i = 0, ..., N-1$ we compute these parameters in the "smoothest" possible way for a Clough-Tocher interpolating function over the triangulation $\tau_{loc(i)}$, where $\tau_{loc(i)}$ is the set of all triangles having the i-th point V_i as a common vertex (that is the cell centered in V_i). The program minimizes the functional (2) with respect to $v_{loc(i)}$, the same of v when τ becomes $\tau loc(i)$. Let M be the number of nodes in the cell $\tau_{loc(i)}$. The program computes

  $$c = \phi(0),\ A,\ g_j = \phi(e_j) - \frac{1}{2}\, a_{jj} - c,$$

  where e_j is the j-th vector of the canonical basis of $I\!R^{2M}$.

- For the Powell-Sabin polynomial patch, we proceed as for the CT-element but taking into account the possibility of elements of A-type or B-type (see above). The functional to be minimized is the again (2) with respect to $v_{loc(i)}$, the same of v when τ becomes $\tau loc(i)$. Let M be the number of nodes in the cell $\tau_{loc(i)}$. The program computes

  $$c = \phi(0),\ A,\ g_j = \phi(e_j) - \frac{1}{2}\, a_{jj} - c,$$

  where e_j is the j-th vector of the canonical basis of $I\!R^{2M}$.

The programs `do_der` (for the element Q_{18}), `clt_der` (for Clough-Tocher), and `pws_der` (for Powell-Sabin) estimate the derivatives by minimizing the energy functional (2) over the star of triangles which share a node. The syntax is:

```
do_der file1 file2 [file3]  (Q18)
clt_der file1 file2 [file3] (Clough-Tocher)
pws_der file1 file2 [file3] (Powell-Sabin)
```

where `file1` is the file of the data points, `file2` is the triangulation file in the format generated by `do_trian` or `rk2tri` and `file3` is the file of the estimated derivatives. If `file3` is omitted the results will be displayed on the command window. We suggest to use the following suffixes for `file3`: for the element Q_{18} `.g_d`, for the Clough-Tocher's element `.ct` and for the Powell-Sabin's element `.pw`.

2.2.3 Optimized Clough-Tocher

Only for the Clough-Tocher patch the estimate can be improved by using the formula (cf. [4]):

$$g_i = (1-q)\; g_{i,i} + q\; R(d_{i,j}, g_{i,j}), \quad 0 \le q \le 1 \;,$$

here

g_i : is the actual estimation of the gradient at the i-th point P_i;
$g_{i,i}, g_{i,j}$: are the actual estimation of the gradient at the i-th and j-th point P_j, $j = 1, ..., k$ in the neighbor of the i-th one;
k : is the number of neighboring nodes;
$d_{i,j}$: is the Euclidean distance between P_i and P_j;

and

$$R(d_{i,j}, g_{i,j}) = \frac{g_{i,1}\; \alpha\; E_1/d_{i,1}^{\gamma} + \cdots + g_{i,k}\; \alpha\; E_k/d_{i,k}^{\gamma}}{\alpha\; E_1/d_{i,1}^{\gamma} + \cdots + \alpha\; E_k/d_{i,k}^{\gamma}}$$

with $E_j = \exp(-\beta\; d_{i,j}^2)$, α, β, γ are real numbers to be set. For $\alpha = 1, \beta = 0$ and $\gamma = 2$ formula [8, (8)] is then obtained. Then the program requires in input the values q, α, β and γ. We suggest to use $q = 0.5$.

The syntax is:

```
clt_der2 file1 file2 [file3] (Optimized Clough-Tocher)
```

where `file1`, `file2` and `file3` are as above. The suffix proposed for `file3` is `.ct2`.

2.3 Energy functional computation

Once we have generated the partial derivatives we may compute the value of the functional (1). The syntax is as follows:

```
funz_q18 file1 file2 file3 (Q18 element)
funz_clt file1 file2 file3 (Clough-Tocher)
funz_pws file1 file2 file3 (Powell-Sabin)
```

where `file1` is the data file made up of 3-dimensional points. `file2` is the triangulation data file in the format generated by `do_trian` or `rk2tri` and `file3` is the derivatives data in which each row refer to the same vertex of the triangulation.

The output is a *real number* representing the energy of the interpolating surface. The lower is this number the better is the surface so computed.

2.4 Miscellaneous

We provide three programs that give more information about the triangulation and its nodes.

`gen_rand` : generates a file of random 3D-coordinates points or allows the user to input a bunch of points by means of a simple script shell accepting three different strings;

`tri_qlty` : provides information on the angles and the areas of triangles;

`info_vic` : provides the maximum and the average number of nodes per cell.

The synopses are as follows.

```
gen_rand file1 [-o2]
```

This program uses a simple script that accepts three input strings:

```
 1) random - to compute N random points between [xmin,xmax],
             [ymin,ymax] and [zmin,zmax].
             Input values: N, xmin, xmax, ymin, ymax, zmin, zmax.

 2) input  - to allow the user to introduce N points.
             Input value: N.

 3) exit   - to exit from the program.
```

The optional argument `-o2` allows to use 2-dimensional (x, y) points instead of 3-dimensional (x, y, z) points.

```
tri_qlty file1 file2
```

```
file1: the data file made up of the 3-dimensional points.

file2: the triangulation data file in the format generated
       by the program 'do_trian' or 'rk2tri'.
```

Finally,

```
info_vic file1
```

```
file1: the triangulation data file in the format generated by the
       program 'do_trian' or 'rk2tri'.
```

3 Visualizing a surface with `labsup3d`

The **visualization program** of the package is `labsup3d`. It builds and manages the $\mathcal{C}^1$ surfaces interpolating scattered data, using methods based on triangulations. It draws the triangulation and the corresponding surface.

We considered two types of interpolation problems:

1. Parametric interpolation of class C^∞ of $(n+1)(n+2)/2$ tridimensional points by n-degree parametric Bézier patches.

2. Non parametric interpolation of class $\mathcal{C}^1$ of $N \geq 3$ tridimensional points, referred to $z = 0$, with interpolation schemes chosen among the following ones.

   (a) Clough-Tocher with linear normal derivatives;

   (b) Powell-Sabin with subdivisions of A-type or B-type;

   (c) Q_{18}.

   Besides the interpolation points, it needs a triangulation over which to build the interpolating function as a polynomial spline of class C^1 by using non parametric Bézier patches.

For the representation and the visualization of smooth interpolating surfaces by Bézier patches there is an interactive program that we called `labsup3d`, which manages three dimensional images. The program works with two graphics libraries: the *OpenGL* on UNIX operative systems supporting the GLX library and the *free OpenGL* work-alike library *Mesa* supporting GLUT library.

3.1 Compilation hints for `labsup3d`

In the file `README.1ST` as well as in the `Makefile` we described all the necessary details for its compilation. To be precise, before to compile one has to know the operating system (Unix native, Linux or other Unix systems) and if the OpenGL or the Mesa graphics libraries are installed.

For any problems, for more detailts on the exact location of the libraries and for the correct setting of the flags and the variables in the Makefile we suggest to contact your *System Manager*.

In the following example we set some flags and variables in order to compile `labsup3d` under Linux with pthread and Mesa Graphics Library. For completeness we have also written (after the comment symbol '#' of the Makefile) the corresponding lines for machines with AIX or Solaris and OpenGL Graphics Library.

```
# CC      = gcc # C-compiler
CC      = gcc   # Gnu C-compiler


# CFLAGS = -DLABSUP3D_AUX                            # OpenGL C-compiler option
CFLAGS  = -DLABSUP3D_GLUT -I/usr/local/mesa/include  # Mesa C-compiler option

XLIBS   = -L/usr/X11R6/lib -lX11 -lXext -lXmu -lXi      # Linux
```

```
# XLIBS = -lm /usr/lpp/X11/lib/libX11.a                 # AIX
# XLIBS   = -lm -lX11 -lXi -lXext -L/usr/openwin/lib    # Solaris

FLAGS = -I/usr/local/include                     # Linux
# FLAGS = -I/usr/openwin/include                 # Solaris

LIBPTH  = -lpthread             # Linux with pthread

# OpenGL Graphics Libraries
#
# LIBSGL  =  -lGL -lGLU -laux

# Mesa Graphics Libraries
#
LIBSGL  = -L/usr/local/mesa/lib -lglut -lMesaGLU -lMesaGL -lm
```

Then the corresponding TARGETS in the Makefile are:

```
# Linux with pthread
#
labsup3d:  labsup3d.o
           $(CC) $(CFLAGS) -o $@ labsup3d.o\
           -L/usr/local/lib $(XLIBS) $(LIBSGL) $(LIBPTH) $(FLAGS)

# Solaris
#
#labsup3d:  labsup3d.o
#          $(CC) $(CFLAGS) -o $@ labsup3d.o $(XLIBS) $(LIBSGL) $(FLAGS)


# AIX
#
#labsup3d:  labsup3d.o
#          $(CC) -o $@ labsup3d.o $(XLIBS) -L/usr/X11/lib $(LIBSGL) $(CFLAGS)
```

Finally the executable file `labsup3d` is created by typing the command:

```
make labsup3d
```

3.2 Running `labsup3d`

The program starts by typing `labsup3d` on the command window. It then displays a graphic window of default size 500×500 pixels that can be resized in the range from 100 up to 800 pixels. Of course, this option does not interfere with standard resizing, maximization and minimization actions that the user will be normally able to apply to the window by any window manager. Then, the following menu will be displayed:

```
*** LABSUP3D MENU ***

-     a     -  Global Bezier method (Only one triangle)
-     b     -  C1 surface: Clough-Tocher with linear normal derivatives
-     c     -  C1 surface: Powell-Sabin
-     d     -  C1 surface: Q18
-     1     -  Input points for global Bezier method
-     2     -  Modify interpolation points for global Bezier method
-     3     -  View interpolation points for global Bezier method
-     4     -  View control points for global Bezier method
-     5     -  Input points, triangulation and derivatives for C1 surfaces
-     6     -  Multiply input z values and derivatives by a magnifying factor
-     7     -  View points/derivatives
-     8     -  View control points for C1 surfaces
- p q r s   -  Output to files for menu items 3 4 7 8 respectively
-     t     -  Output of the points computed for C1 surfaces
-     g     -  Set graphic parameters and view representation
-     v     -  View triangulation
-   z l 0   -  On/Off switches for depth-buffer,light and box respectively
-     o     -  Output options format and choice of colors
-    ESC    -  Quit program

 MAKE YOUR CHOICES FROM GRAPHIC WINDOW.
 Mouse left button performs rotation in surface mode and prints
 the coordinates of a selected point in triangulation mode.
 Up and down arrows perform central zoom in surface mode.
 Mouse right button (twice pressed) performs selected zoom in any mode.
 Mouse middle button cancels selected zooms in surface mode and allows
 to work with a subtriangulation in triangulation mode.
```

It is worthwhile noticing that all choices have to be done on the **graphic window**.

Let us examine the input choices (numbers `1` and `5`).

- **Bézier global method.** After typing `1` on the graphic window, the user can choose to load the data by hand or from a file previously created. In both cases, since the interpolation is parametric, the program will require the values of the barycentric coordinates [6, 5] u and v ($w = 1 - u - v$) for all points introduced.

  The user who chooses to load the data from a file must take care to the (fixed) format of this file: the data must be grouped on $\frac{(n+1)(n+2)}{2}$ rows and six columns. The first three columns are the barycentric coordinates u, v, w (actually the program reads only the first two and skips the third one, because of the identity $w = 1 - u - v$), while the latter three are the x, y, z cartesian coordinates of the points.

- **Methods based upon triangulations.** After typing `5` on the graphic window, the user has to write the names of the data files where he/she has previously stored the interpolation points, the triangulation and the derivatives values. All these files must be in their own format, as explained above.

To display the interpolating function or to visualize the input and output data points one has to choose the numbers `3` or `7`, respectively.

The underlying triangulation can be displayed by typing `v` on the graphic window.

The zoom area is selected by pressing twice the right mouse button (or the equivalent to the right mouse button for mouses having only two buttons), first at the lower left corner then at the upper right corner. By clicking the left mouse button on the graphic window we get the x, y cartesian coordinates of the point where the pointer is located. These couple of numbers is printed on the window where we run `labsup3d`.

The interpolating function is managed by means of `a, b, c` or `d` keys. The resulting control points can be viewed by pressing keys `4` or `8`.

To store on a file the points and all information about the surface or the triangulation, keys `p, q, r` or `s` correspond to keys `3, 4, 7` or `8`, respectively.

The central button of the mouse allows to define a "clipping box" so that the triangulation can be restricted only to the set of triangles which intersect the interior region of the clipping box. The effect of this selection is to limit the operations only to the selected triangles. If the clipping is performed again, the program gives the possibility to restore the original triangulation or to go ahead with the definition of a new clipping box.

3.3 Displaying the surface on the graphic window

Once the mathematical construction of the interpolating function has been completed, we may display the resulting surface by pressing the `g` key on the graphic window in order to set some necessary parameters for the visualization of the surface in orthographic projections. The parameters to set are as follows.

- *The number n_l of subdivisions.* The display of the interpolating function is done, for every micro-triangle, by computing $\dfrac{(n_l+2)(n_l+3)}{2}$ points on the surface, placed along the intersections with the isobarycentric lines:
$$u = \tfrac{0}{n_l}, u = \tfrac{1}{n_l}, \ldots, u = \tfrac{n_l}{n_l} \; ;$$
$$v = \tfrac{0}{n_l}, v = \tfrac{1}{n_l}, \ldots, v = \tfrac{n_l}{n_l} \; ;$$
$$w = \tfrac{0}{n_l}, w = \tfrac{1}{n_l}, \ldots, w = \tfrac{n_l}{n_l} \; ,$$
where u, v are the barycentric coordinates with respect to the current micro-triangle. In this way, the user has the possibility to check the number of interpolation points drawn on the surface, simply by varying the parameter n_l. In particular, for $n_l = 0$ the program draws only the micro-triangles on the surface, that is the piecewise linear interpolant. If $n_l \geq 1$ we can clearly see the curvature of the surface.

- *The viewing angles λ and φ.* All control points of the polynomial patches that form the interpolating surface can be surrounded by a parallelepiped, parallel to the axes, which depends on the smallest and the biggest values of their components. It is the so called *minmax box* of the surface. If we consider the sphere with center in the minmax box, called *view reference point* or `VRP` (cf. [3]), and radius equal to half the minmax box diagonal, the whole surface lies in the interior of the sphere. Then, we define the *viewing line* as the line connecting the point of view (outside the sphere) with the center of the sphere itself. Hence, λ and φ are the latitude and the longitude of the intersection of the sphere with the viewing line. The actual distance of the view point from the center of the sphere has no relevance since the program uses only orthographic projections. Therefore we can assume that this distance is infinite i.e. a parameter outside the control of the user. The image we can see is thus the orthographic projection of the surface onto the viewing plane, that is the plane normal to the viewing line and passing through the view reference point.

- *The cartesian coordinates x, y, z of the so called* view-up-vector *or* `VUV`*.* These are useful in order to avoid the ambiguity in the definition of a cartesian reference system in the viewing plane. Therefore, we are able to specify in a unique way the viewing plane. On it we have to define a cartesian reference system u, v where the axes correspond to the horizontal and vertical directions on the screen. The *view-up-vector* is just a vector, whose orthographic projection lies on the viewing plane and whose origin is the view reference point, which defines a direction that will look as a vertical on the screen. Of course, in order to be projected this vector will not have to be parallel to the viewing line. On this matter, the program asks to introduce again the view-up-vector if its direction is too close to the viewing line. In most cases, except when $\lambda = \pm 90°$ or it takes values very close to the right angle, we advise to use $(0, 0, 1)$ as view-up-vector, so that the horizontal direction on the screen is parallel to the reference plane $z = 0$.

Further possibilities are left to the user and they are listed for the sake of completeness.

- Left mouse button allows to define again the viewing angles in an interactive way by means of the affine correspondence between the graphical area of the window with the rectangle $-\pi \leq \varphi \leq \pi$, $-\pi/2 \leq \lambda \leq \pi/2$, where the point $(-\pi, -\pi/2)$ is identified with the lower left corner of the window. Otherwise, pressing again g on the graphic window and introducing again the same value for n_l, it is possible to change the viewing angles, and, eventually, also the VUV, without computing again the points on the surface.
- Pressing twice the right mouse button, we can define a zoom area: once for the lower left corner and once for the upper right corner.
- Central mouse button allows to restore the zoom area.
- "Up arrow" and "down arrow" keys allow to execute a permanent zoom with a scale factor set by default to 1.5, and easily changeable by the program options.
- The `z` key allows to activate or to switch it off the hidden surfaces removal.
- The `l` key allows to define a point in the Euclidean space where to put a light source in order to lighten the surface or to switch it off.
- The `O` key allows to display the bounding box around the surface or to hidden it.
- In the case of C^1 surfaces on triangulations, the keys `b`, `c`, and `d`, allow to choose the three interpolation methods so far implemented. The `v` key, allows the visualization of the current triangulation.

3.4 Changing standard program options

Pressing `o` on the graphic window, one redefines some default parameters.

1. The *width* and *precision* fields by which real numbers are printed (both to screen and to file) and which refer respectively to the full number of characters visualized (by default 20) and to the number of decimal digits (by default 10).
2. The scale factor (by default 1.5) for central permanent zooms.
3. The colors used to draw the triangles edges, their interior regions and the window background. The new color data, to be introduced as real numbers in the range 0...1, are the RGB components. The filling for the interior of triangles is active only for the triangles on the surfaces, while the window background color is used for the triangulation. The default values are $(0.4, 0.5, 0.5)$ for the window background color, $(0, 0, 1)$ for the interior regions filling and $(0, 1, 1)$ for the color of the triangles edges.

References

[1] P. Alfeld. Derivative generation from multivariate scattered data by functional minimization. *CAGD*, 2:281–296, 1985.

[2] R. E. Barnhill and G. Farin. C^1 quintic interpolation over triangles: Two explicit representations. *Int. J. Numer. Meth. Eng.*, 17(12):1763–1778, dec 1981.

[3] K. Singleton. An implementation of the GKS-3D PHIGS viewing pipeline. *Eurographics 86*, North-Holland, 1986.

[4] S. De Marchi, On Computing Derivatives for $\mathcal{C}^1$ Interpolating Schemes: an Optimization *Computing*, 60:29-53, 1998.

[5] G. Farin. Triangular Bernstein-Bézier patches. *CAGD*, 3:83–127, 1986.

[6] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, 2 edition, 1988.

[7] 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, University of Padua, 1995. (In Italian)

[8] M. Morandi Cecchi, S. De Marchi and D. Fasoli, A Package for Representing $\mathcal{C}^1$ Interpolating Surfaces: Application to the Lagoon of Venice's Bed. *Numer. Algorithms, 20* 1999.

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

[10] E. Quak and L. L. Schumaker. Cubic spline fitting using data dependent triangulations. *CAGD*, 7:293–301, 1990.

[11] R. J. Renka and A. K. Cline. A triangle-based C^1 interpolation method. *Rocky Mountain J. Math.*, 14(1), winter 1984.

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