Object oriented finite element calculations using Maple

Dmytro Chibisov, Victor G. Ganzha, and Christoph Zenger

Institute of Informatics, Technical University of Munich, Garching 85748, Boltzmannstr. 3, Germany; e-mail: `chibisov@in.tum.de, ganzha@in.tum.de, zenger@in.tum.de`

Summary. Modern Computer Algebra Systems (CAS), such as *Maple* or *Mathematica*, with their symbolic facilities and visualization possibilities are powerful tools to design data structures and algorithms used in numerical simulation, with significantly lower costs compared to straightforward implementation in "real" programming languages, such as, for example, C or Java. The present paper shows how the CAS Maple can be used to design finite element software using linear and hierarchical bases. As a computational example the two-dimensional Poisson-equation with Dirichlet boundary conditions is presented.

1. Introduction

Modern Computer Algebra Systems (CAS), such as *Maple* or *Mathematica*, with their symbolic facilities and visualization possibilities are powerful tools to design data structures and algorithms used in numerical simulation, with significantly lower costs compared to straightforward implementation in "real" programming languages, such as, for example, C or Java (Fig. 1).

We will show how the Maple abilities to solve equations, symbolic integration and differentiation can be used in the context of finite element calculations. The possibility to perform computation with
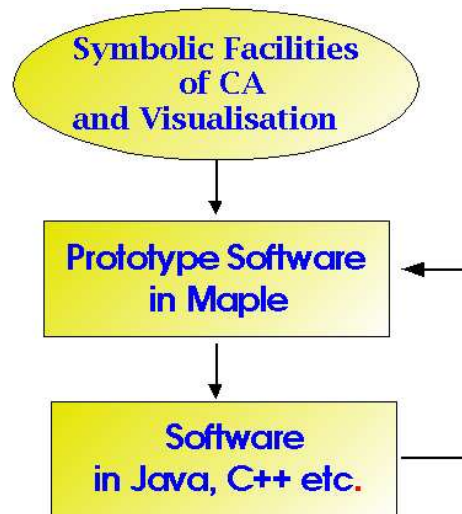
Fig. 1. Computer Algebra in Software Development Cycle

symbolic constants instead of numerical, for example in boundary conditions, is of industrial relevance.

The present work investigates some principles of symbolic programming languages, using Maple as example, with regard to its application in the context of Finite Element Method (FEM).

FEM is a powerful tool for the solution of many scientific and engineering modeling problems. Typical FEM programs consist of thousand lines of poorly structured Fortran or C code, in which the data are stored in lists and arrays and distributed over the complete system. The little change in such systems causes big bugs and, consequently, such software is not fit for many scientific and engineering purposes.

To cope with this complexity we need strategies, that would help us to understand principles of complex computational processes and structure the corresponding software based on this principles. Therefore, in section 2 the computational process of solving Partial Differential Equations (PDE's) using FEM is explained. Using space pavings and grids, we show how primitive data, for example numbers and symbols, can be combined to compound structures, which can be used to represent complex data and relationships among them. The notion of relationships between data representing pavings and grids will be formal described. All this leads to the notion of object oriented modeling. We will introduce our package, that allows to de-

velop object oriented programs in Maple. We shall show, how this package can be applied in the context of finite element simulation.

As computational examples, the solutions of two-dimensional Poisson equation using hierarchical and linear approximations with Dirichlet boundary conditions are presented.

2. Finite element computation

Consider the boundary value problem on the region Ω with the boundary Γ:

$$(1) \qquad u(x,y)_{xx} + u(x,y)_{yy} = f(x,y), \quad u(\Gamma) = 0.$$

In order to solve this equation using the finite element method, one have to minimize the following energy functional:

$$E(v) = \int_{\Omega} \int \left[\frac{1}{2}(v(x,y)_x^2 + v(x,y)_y^2) - v(x,y)f(x,y) \right] d\Omega$$

over a certain functional space X.

The finite approximation $\tilde{u}$ is assumed to be of the following form:

$$(2) \qquad \tilde{u}(x,y) = \sum_{i=1}^{N} c_i \phi_i(x,y),$$

where $\phi_i(x,y)$ are the so-called Ansatz functions and c_i the unknown coefficients to be determined.

According to the Ritz-Galerkin approach, c_i's can be computed by solving

$$A\overrightarrow{c} = \overrightarrow{b},$$

where A is the stiffness matrix and b is the load vector given by

$$(3) \qquad A(i,j) = \int_{\Omega} \int \left[\frac{\partial}{\partial x}\phi_i(x,y)\frac{\partial}{\partial x}\phi_j(x,y) \right.$$

$$\left. + \frac{\partial}{\partial y}\phi_i(x,y)\frac{\partial}{\partial y}\phi_j(x,y) \right] d\Omega,$$

$$(4) \qquad b(i) = \int_{\Omega} \int \phi_i(x,y)f(x,y)d\Omega.$$

Hat functions of the form

$$\phi_i(x) := \begin{cases} 1 - \dfrac{|x - x_i|}{h_i} & \text{if } |x - x_i| \leq h_i \\ 0 & \text{otherwise,} \end{cases}$$
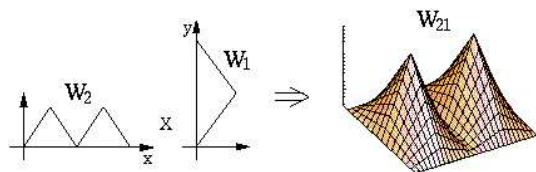
Fig. 2. Function spaces spanned by hat functions

where x_i is the middle point coordinate and h_i the half width of the hat, are typically used to approximate the unknown one-dimensional function (Fig. 2).

The set

$$W_n = \{\phi_1(x), \ldots, \phi_n(x)\}$$

of such hat functions spans a particular space. As it is shown in the figure, one can use the product of two one-dimensional sets to obtain the two-dimensional spanning space $W_{n,m}$:

$$W_{n,m} = W_n \circ W_m = \{\phi_1(x,y) \circ \phi_1(x,y), \phi_1(x,y) \circ \phi_2(x,y), \ldots,$$

$$\phi_2(x,y) \circ \phi_2(x,y), \phi_2(x,y) \circ \phi_3(x,y), \ldots\}.$$

Once $W_{m,n}$ is constructed, the stiffness matrix and load vector can be computed according to (3), (4). Let us consider this computational process more in detail.

3. Grid generation

In mathematical modeling and numerical simulation we have to deal not only with functions, which return a real value dependent on real parameters, but also with the domains over which such functions are defined and especially with parts of these domains - the regions. In this section we will consider the way in which such regions can be described, stored and manipulated in Maple.

As mentioned above, during finite element modeling we have to partition the spatial region of interest in disjunct elements. In one-dimensional case the elements are intervals, in two or three dimensions respectively squares or cubes can be used. This idea is very old and is, for example, used in the classical integration calculus (Riemann integration).

There is quite a choice as to how to define the partition; we will first use the most restrictive definition: dyadic pavings of $\mathbb{R}^n$ [2].

In one-dimensional case we decompose the domain into little intervals. Obviously, each two neighboring intervals have a common node.

We assign to each node a hat function which takes the value 1 in the given node and vanishes in all other nodes.

To use dyadic pavings in $\mathbb{R}^n$, we do essentially the same thing. We cut up $\mathbb{R}^n$ into cubes with sides 1 long. (By a "cube" we mean an interval in $\mathbb{R}$, a square in $\mathbb{R}^2$, a cube in $\mathbb{R}^3$, and analogs of cubes in higher dimensions.) Next we cut each side of a cube in half, cutting an interval in half, a square into four equal squares, a cube into eight equal cubes, etc. At the next level, we cut each side of those in half, and so on.

To define dyadic pavings in $\mathbb{R}^n$ precisely, we must first say what we mean by n-dimensional "cube". According to [2], for every

$$
(5) \qquad \mathbf{k} = \begin{bmatrix} k_1 \\ \vdots \\ k_n \end{bmatrix} \in \mathbb{Z}, \quad \text{where } \mathbb{Z} \text{ represents the integers,}
$$

we define the cube

$$
(6) \qquad C_{\mathbf{k},N} = \left\{ \mathbf{x} \in \mathbb{R}^n \mid \quad \frac{k_i}{2^N} \le x_i < \frac{k_i + 1}{2^N} \right\}.
$$

Each cube has two indices. The first index, $\mathbf{k}$, locates each cube: it gives the numerators of the coordinates of the cube's lower left-hand corner, when the denominator is 2^N. The second index, N, tells which level we are considering, starting with 0. The length of a side of cube is $1/2^N$ (Fig. 3). The coordinates of the cube's corners are:

$$
\left(\frac{k_l}{2^N}, \frac{k_m + 1}{2^N} \right), \quad \left(\frac{k_l}{2^N}, \frac{k_m}{2^N} \right), \quad \left(\frac{k_l + 1}{2^N}, \frac{k_m + 1}{2^N} \right),
$$

$$
\left(\frac{k_l + 1}{2^N}, \frac{k_m}{2^N} \right), \quad 0 \le l \le n, \quad 0 \le m \le n, \quad l \ne m.
$$

Definition 1. *The collection of cubes $C_{\mathbf{k},N}$ at a single level N, denoted $\mathcal{D}_N(\mathbb{R}^n)$, is the Nth dyadic paving of $\mathbb{R}^n$*

At this point we have to make a decision how $\mathcal{D}_N(\mathbb{R}^n)$ can be generated, stored and manipulated in our language.

The basic idea of data abstraction is to structure the programs that are to use data objects, so that they operate on "abstract data". That is, our programs should use data in such a way as to make no assumptions about the data that are not strictly necessary for performing the tasks at hand. At the same time, there are many different representations of the same data. As will be shown bellow, the dyadic paving can be stored in two ways:
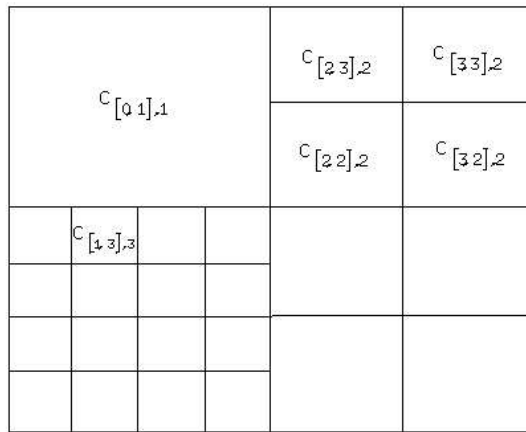
Fig. 3.   A dyadic decomposition in $\mathbb{R}^2$

- using list of middle point coordinates of each cube and its width (elements)
- using list of the corner points of cubes (grid)

As will be shown in the next section, in the finite element method both representations are relevant. Our purpose now is to design the dyadic pavings data representation transparent for the programs using it.

According to (5), (6) each two-dimensional cube

$$C_{\begin{bmatrix} i \\ j \end{bmatrix},N}$$

can be described by its middle point coordinates and its width. Thus, we introduce the triple

$$\left\langle \frac{2i+1}{2^{N+1}}, \frac{2j+1}{2^{N+1}}, 1/2^N \right\rangle.$$

Maple provides a list-construct to build compound data structures from several primitives. For example,

$$\mathcal{D}_0(\mathbb{R}^2) = C_{\begin{bmatrix} 0 \\ 0 \end{bmatrix},0} = \langle \frac{1}{2}, \frac{1}{2}, \frac{1}{2} \rangle$$

can be described by a triple:

```
[> [1/2,1/2,1/2];
                [1/2,1/2,1/2]
```

$\mathcal{D}_1(\mathbb{R}^2)$ consists of four triples

$$C_{\begin{bmatrix} 0 \\ 0 \end{bmatrix},1}, \quad C_{\begin{bmatrix} 0 \\ 1 \end{bmatrix},1}, \quad C_{\begin{bmatrix} 1 \\ 1 \end{bmatrix},1}, \quad C_{\begin{bmatrix} 1 \\ 0 \end{bmatrix},1},$$

which can be stored in the list of lists:

```
[> [[1/4, 1/4, 1/4], [1/4, 3/4, 1/4],
   [3/4, 1/4, 1/4], [3/4, 3/4, 1/4]];
```

The $\mathcal{D}_2(\mathbb{R}^2)$ consists of 16 triples:

```
[[1/8, 1/8, 1/8], [1/8, 3/8, 1/8], [1/8, 5/8, 1/8],
 [1/8, 7/8, 1/8], [3/8, 1/8, 1/8], [3/8, 3/8, 1/8],
 [3/8, 5/8, 1/8], [3/8, 7/8, 1/8], [5/8, 1/8, 1/8],
 [5/8, 3/8, 1/8], [5/8, 5/8, 1/8], [5/8, 7/8, 1/8],
 [7/8, 1/8, 1/8], [7/8, 3/8, 1/8], [7/8, 5/8, 1/8],
 [7/8, 7/8, 1/8]].
```

On the other hand, the coordinates of corners of cubes in $\mathcal{D}_N(\mathbb{R}^n)$ are:

$$\left(\frac{i}{2^N}, \frac{j+1}{2^N} \right), \quad \left(\frac{i}{2^N}, \frac{j}{2^N} \right), \quad \left(\frac{i+1}{2^N}, \frac{j+1}{2^N} \right),$$

$$\left(\frac{i+1}{2^N}, \frac{j}{2^N} \right), \quad 0 \le i, j \le N-1.$$

The grid corresponding to $\mathcal{D}_2(\mathbb{R}^2)$ can be stored in the similar way using list construct:

```
[[0, 0], [0, 1/4], [0, 1/2], [0, 3/4], [0, 1],
 [1/4, 0], [1/4, 1/4], [1/4, 1/2], [1/4, 3/4], [1/4, 1],
 [1/2, 0], [1/2, 1/4], [1/2, 1/2], [1/2, 3/4], [1/2, 1],
 [3/4, 0], [3/4, 1/4], [3/4, 1/2], [3/4, 3/4], [3/4, 1],
 [1, 0], [1, 1/4], [1, 1/2], [1, 3/4], [1, 1]].
```

The usage of lists becomes painful if we refine the grid by cutting some elements in half. Then we must remove some old elements and add new elements from the both lists `elements` and `grid`. We must perform following operations:

- find the position of the element in the list `elements`
- remove the element from the list `elements`
- compute and add new elements to the list `elements`
- compute and add new elements to the list `gird`

The situation will be much more tedious if we try to remove some elements. In this case the following operations have to be performed:

- find the position of the element in the list `elements`
- remove the element from the list `elements`

- for each node of element to remove, check the presence of other owners in the list `elements`
- find the position of each elements node in the list `grid`, which does not belong to any other elements
- remove this node

  These difficulties arise chiefly from two facts:

- the relationship between two different representations of a paving are not explicitly described
- the positions of elements and grid nodes in the list do not correspond to their spatial position

  Let us begin with the second problem and show the way to overcome this difficulty.

3.1. Hashing

The idea is simple : instead of using lists indexed by integers, we make the usage of the hashing principle and Maple ability to *quote* a data object, such as number or procedure. Using quotation ('...' or `convert(...,name)`) we can build symbols from numbers. For example:

```
[> '1/4':=1/4;
```

or

```
[> convert(1/4,name):=1/4;
```

Using quotation, the left hand side of an assignment is interpreted by Maple as a symbol, to which a particular value can be assigned. Using this idea, we store our elements in variables whose names are built from spatial coordinates. Let us define the following procedure, which generates a dyadic paving at level `N`:

```
Dyadic_Paving:=proc(N)
local i,j:

 for i from 0 to 2^N-1 do
  for j from 0 to 2^N-1 do
     x:=convert((2*i+1)/2^(N+1),name);
     y:=convert((2*j+1)/2^(N+1),name);

     element||x||y:=[(2*i+1)/2^(N+1),(2*j+1)/2^(N+1),1/2^(N+1)];
   od:
 od:

end proc:
```

We use concatenation operation (||) to build the complex names. For example, the element with middle point coordinates $x = 1/2, y = 1/2$ and width $h = 1/2$ will be stored in the variable "element1/21/2".

From here it must be clear that in this implementation operations like finding an element do not require computational costs, in contrast to searching in a list. For example, we can address an element with the spatial position $x = 1/4$, $y = 3/4$ by

```
[> element||'1/4'||'3/4';
                [1/4,3/4,1/4]
```

Before we address the second problem concerning the relationships between pavings and girds, it would be advisable to gather all the data, which describes an element, and a grid node, in one structured data unit.

3.2. Compound data objects

The data unit corresponding to a certain element consists of the following parts:

- variable `x` : x coordinate
- variable `y` : y coordinate
- variable `h` : element width

The following procedure generates variables, in which spatial coordinates of an element and its width are stored and returns the symbol built from spatial coordinates, by which this element can be referred to:

```
Element:=proc(x_i, y_i, h_i)
local this:

this:= ''||(convert(x_i,name))||(convert(y_i,name));

element||this||x:=x_i;
element||this||y:=y_i;
element||this||h:=h_i;

return this:
end proc:
```

the variables, in which the element's data is stored, are built from prefix `element` and the element's spatial coordinates. For example

```
[> elem1:=Element(1/2,1/2,1/2);
                    1/21/2
```

Now we can refer to data fields (for example `h`) of the just generated element by

```
[> element||elem1||h;
                       1/2
```

The data fields, which correspond to a grid node, are:

- variable x : x coordinate
- variable y : y coordinate

Similar to the constructor `Element`, the constructor `GridNode` can be implemented:

```
GridNode:=proc(x_i, y_i)
local this:

this:= ''||(convert(x_i,name))||(convert(y_i,name));

node||this||x:=x_i;
node||this||y:=y_i;

return this:
end proc:
```

The value returned by the procedures `Element` and `GridNode` can be considered as identifier of the compound data object. Through this identifier we can refer to data fields of this object.

The relationship between elements and grid nodes can be represented with the aid of such identifiers.

For example let us rewrite the procedure `GridNode` in the following way:

```
GridNode:=proc(x_i, y_i)
local this:

this:= ''||(convert(x_i,name))||(convert(y_i,name));

if node||this||exists = true then
   return this;
fi:

node||this||exists:=true;

node||this||x:=x_i;
node||this||y:=y_i;

node||this||elements:=[];

node||this||addElement:=proc(elem)
  node||this||elements:=[op(node||this||elements),elem];
end proc:

return this:
end proc:
```

First of all, the new data field, the list `elements` and the procedure `addElement` were added. We will see bellow how the generated elements can be stored in `elements` with the aid of this procedure.

Since each one node belongs to several elements, we need at the beginning of the procedure `GridNode` to check, whether this node has already been created or not. If this is not the case, we initialize all data fields, otherwise exit the procedure.

Then the procedure `Element` can be overwritten as follows:

```
Element:=proc(x_i, y_i, h_i)
local this:

this:= ''||(convert(x_i,name))||(convert(y_i,name));

element||this||x:=x_i;
element||this||y:=y_i;
element||this||h:=h_i;

element||this||node1:=Node(element||this||x-(element||this||h)/2,
                           element||this||y-(element||this||h)/2);
element||this||node2:=Node(element||this||x+(element||this||h)/2,
                           element||this||y-(element||this||h)/2);
element||this||node3:=Node(element||this||x-(element||this||h)/2,
                           element||this||y+(element||this||h)/2);
element||this||node4:=Node(element||this||x+(element||this||h)/2,
                           element||this||y+(element||this||h)/2);

node||(element||this||node1)||addElement(this);
node||(element||this||node2)||addElement(this);
node||(element||this||node3)||addElement(this);
node||(element||this||node4)||addElement(this);

return this:
end proc:
```

Additionally to data fields `x`, `y` and `h`, we have introduced the data fields `node1`, `node2`, `node3`, `node4`, which store the identifiers of its grid nodes.

To generate paving and grid we can use the following code:

```
for i from 1 to N do
  for j from 1 to N do
    Element(2*i+1)/2^(N+1),(2*j+1)/2^(N+1),1/2^(N+1));
  od:
od:
```

Note that in fact we have developed the way, in which object oriented programs can be implemented in Maple. The procedures `GridNode` and `Element` correspond to the classes in object oriented languages. The instances of classes or objects can be created by calls

of this procedures. With the aid of the concatenation operation "||",
we can refer to class variables and methods. In the next section we
shall present our package for object oriented programming in Maple
based on the ideas introduced in this chapter.

In case of N = 0 one element will be generated and we can refer
to data it is characterized with, for example to its width, by

```
[> element||'1/2'||'1/2'||width;
                      1/2
```

The element's nodes can be referred to by

```
[> element||'1/2'||'1/2'||node1;
   element||'1/2'||'1/2'||node2;
   element||'1/2'||'1/2'||node3;
   element||'1/2'||'1/2'||node4;


                       node00
                       node01
                       node10
                       node11
```

The data on nodes, the list of elements to which the node belongs,
can be obtained in the same way:

```
[> element||'1/2'||'1/2'||node1||elements;
               [element1/21/2]
```

In this way, we have represented relationships between grid nodes
and elements by "pointers" and can now discuss the second prob-
lem concerning representation of relations between elements end grid
nodes in more detail.

3.3. Relationships between data objects

We see that the procedure, which generates paving and grid, is equiv-
alent to a series of assignments.

For example, in case of $\mathcal{D}_0(\mathbf{R}^2)$ the data of a single element and
its 4 nodes will be stored in the following variables:

```
element1/2/1/2x:=1/2;
element1/2/1/2y:=1/2;
element1/2/1/2width:=1;
element1/2/1/2node1:=node00;
element1/2/1/2node2:=node01;
element1/2/1/2node3:=node10;
element1/2/1/2node4:=node11;

node00elems[1]:=element1/21/2;
node00x=0;
node00y=0;
```

```
node01elems[1]:=element1/21/2;
...
node10elems[1]:=element1/21/2;
...
node11elems[1]:=element1/21/2;
...
```

in case of $\mathcal{D}_1(\mathbf{R}^2)$:

```
element1/41/4width:=1/2;
element1/41/4node1:=node00;
element1/41/4node2:=node01/2;
element1/41/4node3:=node1/20;
element1/41/4node4:=node1/21/2;

element3/41/4width:=1/2;
element3/41/4node1:=node01/2;
element3/41/4node2:= ...
...
element1/43/4width:=1/2;
...
element3/43/4width:=1/2;
...

node00elems[1]      :=element1/41/4;

node01/2elems[1]    :=element1/41/4;
node01/2elems[2]    :=element1/43/4;

node1/20elems[1]    :=element1/41/4;
node1/20elems[2]    :=element3/41/4;

node1/21/2elems[1]  :=element1/41/4;
node1/21/2elems[2]  :=element1/43/4;
node1/21/2elems[3]  :=element3/41/4;
node1/21/2elems[4]  :=element3/43/4;

node01elems[1]      :=element1/43/4;
node10elems[1]      :=element3/41/4;
node11elems[1]      :=element3/41/4;
```

and so on.

Now we want to introduce the formal definition of the relationship between grid and paving in our computational model

Let $V = \{v_i\}$ be a linear ordered set of all assigned variables. We denote the function, which returns the name of the variable v_i, $v_i \in V$, by $\mathcal{N}(v_i)$. The evaluation function, which returns the value of the variable v_i, $v_i \in V$, will be denoted by $\mathcal{E}(v_i)$.

Additionally, we introduce the predicate $P(\mathcal{N}(v_i), prefix)$, which is true, iff the name of $v_i \in V$ has prefix $prefix$.

Furthermore let $\mathcal{G}$ be a linear ordered set of generated grid nodes and $\mathcal{D}$ the set of generated elements. Obviously, $\mathcal{G}$ consists of those variables, whose name begins with "node" and $\mathcal{D}$ of those, whose name begins with "element":

$$\mathcal{G} = \{\mathcal{N}(v_i) \mid P(\mathcal{N}(v_i), node)\},$$

$$\mathcal{D} = \{\mathcal{N}(v_i) \mid P(\mathcal{N}(v_i), element)\}.$$

For example, in case of $\mathcal{D}_0(\mathbf{R}^2)$ we obtain:

$$\mathcal{D} = \{\texttt{element1/2/1/2x}, \texttt{element1/2/1/2y},$$

$$\texttt{element1/2/1/2width}, \texttt{element1/2/1/2node1},$$

$$\texttt{element1/2/1/2node2}, \texttt{element1/2/1/2node3},$$

$$\texttt{element1/2/1/2node4}\},$$

$$\mathcal{G} = \{\texttt{node00elems}, \texttt{node00x}, \texttt{node00y}, \texttt{node01x}, \texttt{node01y},$$

$$\texttt{node01elems}, \texttt{node10x}, \texttt{node10y}, \texttt{node10elems}, \texttt{node11x},$$

$$\texttt{node11y}, \texttt{node11elems}\}.$$

The Cartesian product $\mathcal{G} \times \mathcal{D}$ can be described as follows :

$$\mathcal{D} \times \mathcal{G} = \{(\mathcal{N}(v_i), \mathcal{N}(v_j)) \mid$$

$$P(\mathcal{N}(v_i), element) \wedge P(\mathcal{N}(v_j), node) \vee$$

$$P(\mathcal{N}(v_i), node) \wedge P(\mathcal{N}(v_j), element)\}.$$

For example:
$\mathcal{D} \times \mathcal{G} = \{$

```
(element1/2/1/2x,node00elems),
(element1/2/1/2x,node00x),
(element1/2/1/2x,node00y), (element1/2/1/2x,node01elems), ...,
(element1/2/1/2x,node10elems), ..., (element1/2/1/2x,node11elems),
 ..., (element1/2/1/2y,node00elems), (element1/2/1/2y,node00x),
(element1/2/1/2y,node00y), (element1/2/1/2y,node01elems), ...,
(element1/2/1/2y,node10elems), ..., (element1/2/1/2y,node11elems),...
```

$\}$.

Let us define the correspondence relation $\mathcal{C} \subseteq \mathcal{G} \times \mathcal{D}$ between grid nodes and elements:

$$(7) \quad \mathcal{C} = \{(\mathcal{N}(v_i), \mathcal{N}(v_j)) \mid P(\mathcal{N}(v_i), element) = P(\mathcal{E}(v_i), node) \wedge$$

$$P(\mathcal{N}(v_j), node) = P(\mathcal{E}(v_j), element)\}.$$

For example, in case of $\mathcal{D}_1(\mathbf{R}^2)$, we obtain exactly the pairs of variables, which belong to corresponding elements and grid nodes:

$\mathcal{C} = \{$

```
(element1/41/4node1, node00elems[1]),
(element1/41/4node2,node01/2elems[1]),
(element1/41/4node3,node1/20elems[1]),
(element1/41/4node4,node1/21/2elems[1]),
(element3/41/4node1, node1/40elems[2]),
(element3/41/4node2,node1/21/2elems[3]), ...,
(element3/43/4node3, node1/21/2elems[4]), ...,
(element1/43/4node4, node1/21/2elems[2]), ...
```

$\}$

In this way the notion of relationships between compound data objects can be represented formally. In the next section we present the package based on this computational model, which allows to develop object oriented programs in Maple.

4. Object oriented programming in Maple: FEM in action

To make the data abstraction technique introduced in the previous chapter more reliable and user-friendly we have developed a package that makes it possible to implement the object-oriented programs in Maple ([3], [1]).

Additionally to the features shown in the previous chapter, our package:

- manages the instance identifiers
- allows the operations like
  - searching in object oriented data structures like trees
  - adding objects
  - removing objects

  to be implemented in a simple way.

In this chapter, we would like to show the way, in which this package operates using a finite element solver as example.

5. Finite element method using linear basis

In this section we show how the linear finite element method can be implemented using the ideas of object oriented programming.

The class Node introduced in the previous section consists of the following member variables

- x_i : x coordinate of the elements middle point
- y_i : y coordinate of the elements middle point
- list elems : stores the owners of this node

- list `ansatz` : stores the Ansatz functions assigned to this node

and of the following member procedures

- `addAnsatzFunction( f )`: stores the Ansatz function `f` in the list `ansatz`

```
GridNode:=proc(xi,yi)
local this:

this:=DECLARE_CLASS(node,xi,yi);

node||this||x_i:=xi;
node||this||y_i:=yi;
node||this||ansatz:=[];
node||this||elems:=[];

node||this||addAnsatzFunction:=proc(f, elem)
use oop in
 node||this||ansatz:=[op(node||this||ansatz),f]:
 node||this||elems:=[op(node||this||elems),elem]:
end use:
end proc:

return this:
end:
```

The main difference between this implementation and the implementation from the last chapter is the command DECLARE_CLASS provided by our package. This command builds the instance identifier and ensures that an object with given coordinates is generated only once (see previous chapter).

An instance of this class can be generated by using command `new` provided by our package:

```
[> new(GridNode,[1/2,1/2]);
              1/21/2
```

As arguments of `new` accept the class name (here `GridNode`) and the constructor parameters (here coordinates of the node).

The class `Element` introduced in the previous section consists of the following member variables

- x_i : x coordinate of the elements middle point
- y_i : y coordinate of the elements middle point
- h_i : the half width of the element
- node1, node2, node3, node4: identifiers of the element's nodes
- v1, v2, v3, v4: local Ansatz functions, which take the value 1 in each of four nodes and vanish in all other nodes

and can be implemented in Maple as follows:

```
Element:=proc(xi,yi,hi)

this:=DECLARE_CLASS(element,xi,yi);

element||this||x_i:=xi;
element||this||y_i:=yi;
element||this||h_i:=hi;

element||this||node1:= new(GridNode, [xi-hi,yi-hi]);
element||this||node1:= new(GridNode, [xi+hi,yi-hi]);
element||this||node1:= new(GridNode, [xi-hi,yi+hi]);
element||this||node1:= new(GridNode, [xi+hi,yi+hi]);

element||this||v1:= ...
element||this||v2:= ...
element||this||v3:= ...
element||this||v4:= ...

node||(element||this||node1)||addAnsatzFunction(f1,this);
node||(element||this||node1)||addAnsatzFunction(f2,this);
node||(element||this||node1)||addAnsatzFunction(f3,this);
node||(element||this||node1)||addAnsatzFunction(f4,this);

return this:
end:
```

Before we start our computation, the Ansatz functions v1, v2, v3, v4 must be determined.

5.1. Nodal basis

The Ansatz functions v1, v2, v3, v4 are assumed to be bilinear functions of the form $a_0 + a_1 x + a_2 y + a_3 xy$. The coefficients a_i depend on the spatial position of the element and can be determined from the requirement that each Ansatz function takes the value 1 in one of the element nodes and vanishes in all other nodes.

For example, for v1 the following system of equations may be produced:

$$
\begin{cases}
a_0 + a_1\left(x_i - h_i\right) + a_2\left(y_i - h_i\right) + a_3\left(x_i - h_i\right)\left(y_i - h_i\right) = 1, \\
a_0 + a_1\left(x_i + h_i\right) + a_2\left(y_i - h_i\right) + a_3\left(x_i + h_i\right)\left(y_i - h_i\right) = 0, \\
a_0 + a_1\left(x_i - h_i\right) + a_2\left(y_i + h_i\right) + a_3\left(x_i - h_i\right)\left(y_i + h_i\right) = 0, \\
a_0 + a_1\left(x_i + h_i\right) + a_2\left(y_i + h_i\right) + a_3\left(x_i + h_i\right)\left(y_i + h_i\right) = 0.
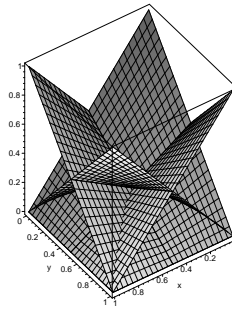\end{cases}
$$

Fig. 4. Local basis functions

We can find unknown coefficients a_0, a_1, a_2, a_3 using the Maple command `solve`.The following solution will be obtained:

$$\left\{ a_0 = 1/4 \frac{x_i y_i + x_i h_i + h_i y_i + h_i^2}{h_i^2}, \quad a_1 = -1/4 \frac{y_i + h_i}{h_i^2}, \right.$$

$$\left. a_2 = -1/4 \frac{x_i + h_i}{h_i^2}, \quad a_3 = 1/4 h_i^{-2} \right\}.$$

The coefficients of other Ansatz functions can be determined similarly. Finally, for each element four local Ansatz functions are constructed (Fig. 4).

5.2. Assembling of stiffness matrix

In this section we will compute the stiffness matrix and load vector according to (3), (4). Let us begin with the grid consisting of one element and four nodes:

```
[> e1 := new(Element,[1/2,1/2,1/2]);
```

The Ansatz functions can be referred to by

```
[> element||e1||v1;
   element||e1||v2;
   element||e1||v3;
   element||e1||v4;

          1-x-y+x*y
          y-x*y
          x - x y
          x y
```

According to (2) we are looking for the solution of (1) in the following form

(8) $\quad u(x, y) = c_1 v_1(x, y) + c_2 v_2(x, y) + c_3 v_3(x, y) + c_4 v_4(x, y).$

Maple provides constructs `Matrix` and `Vector`, which can be used to define stiffness matrix and load vector:

```
[> A:= Matrix(4);
   b:= Vector(4);
```

Then according to (3) the stiffness matrix can be computed using Maple procedures `diff` (symbolic differentiation) and `int`(symbolic integration) by:

```
[>for i from 1 to 4 do
   for j from 1 to 4 do
    A[i,j]:=int(int(diff(objthis1v||i,x)*diff(objthis1v||j,x),
                                          x=0..1),y=0..1)+
          int(int(diff(objthis1v||i,y)*diff(objthis1v||j,y),
                                          x=0..1),y=0..1);
   od:
  od:
```

We obtain the following matrix:

$$
(9) \quad \begin{bmatrix} 2/3 & -1/6 & -1/6 & -1/3 \\ -1/6 & 2/3 & -1/3 & -1/6 \\ -1/6 & -1/3 & 2/3 & -1/6 \\ -1/3 & -1/6 & -1/6 & 2/3 \end{bmatrix}.
$$

The rank of the matrix can be determined with the aid of procedure `Rank` from the Maple package `LinearAlgebra`:

```
[> Rank(A);
        3
```

Since the rank of `A` is 3, the system of equations $Ac = b$ does not have the unique solution and one of the coefficients in (8) must be provided.

The power and flexibility of Finite Element Method manifests itself in the possibility to model systems consisting of several elements. In this section we will show how the stiffness matrix for the complete system can be assembled from the individual element matrices (9).

At first, let us generate a grid consisting of 4 elements:

```
[> for i from 1 to 2 do
     for j from 1 to 2 do
       new(Element,[i/2,i/2,i/2]);
     od:
   od:
```

Our package provides the following functions to obtain the information about the generated objects:

- `getNumberOfInstances` - returns the number of instances of the given class
- `getInstances` - returns the list of instances of the given class

Let us illustrate how the function `getInstances` can be used:

```
[> D:=getInstances(element);
   G:=getInstances(node);

    D =
        [1/41/4, 3/41/4, 1/43/4, 3/43/4]

    G =
        [00, 1/20, 1/21/2, 01/2, 10, 11/2,
         01, 1/21, 11]
```

This loop thus also generates 4 elements and 9 grid nodes.

Recall from the previous section that we denote by $\mathcal{G}$ a linear ordered set of generated grid nodes and by $\mathcal{D}$ a linear ordered set of generated elements.

We denote the area of a finite element by Ω_i. Furthermore, we define the set of local Ansatz functions $\mathcal{V}_{\mathcal{D}}(i)$, which belong to the element $\mathcal{D}_i$, and the set of local Ansatz functions $\mathcal{V}_{\mathcal{G}}(i)$, which take the value 1 in the node $\mathcal{G}_i$.

Then the function ϕ_i can be represented as a sum of four local Ansatz functions:

$$\phi_i = \sum_{v_k \in \mathcal{V}_{\mathcal{G}(i)}} v_k.$$

Using the correspondence relation $\mathcal{C} \subseteq \mathcal{G} \times \mathcal{D}$ between elements and grid nodes (7), we introduce the set $\mathcal{O}$ of owners of a certain grid point $\mathcal{G}_i$:

$$\mathcal{O}(i) = \{j \mid (\mathcal{D}_j, \mathcal{G}_i) \in \mathcal{C}\}.$$

Then the equation (3) can be rewritten as follows:

$$A(i,j) = \sum_{k \in \mathcal{V}_G(i)} \sum_{l \in \mathcal{V}_G(j)} \left[\iint_{(\bigcup_{m \in \mathcal{O}(i)} \Omega_m) \bigcap (\bigcup_{n \in \mathcal{O}(j)} \Omega_n)} \left(\frac{\partial}{\partial x} v_k(x,y) \right. \right.$$

$$\left. \left. \times \frac{\partial}{\partial x} v_l(x,y) + \frac{\partial}{\partial y} v_k(x,y) \frac{\partial}{\partial y} v_l(x,y) \right) d\Omega \right].$$

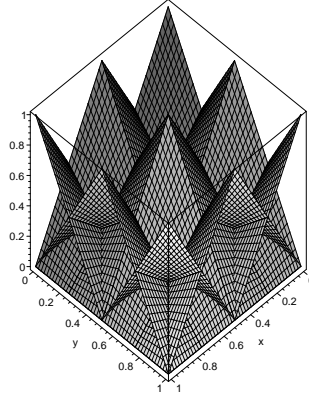**Fig. 5.** Linear basis

The following stiffness matrix can be obtained:

(10)
$$
\begin{bmatrix}
2/3 & -1/6 & -1/6 & -1/3 & 0 & 0 & 0 & 0 & 0 \\
-1/6 & 4/3 & -1/3 & -1/3 & -1/6 & -1/3 & 0 & 0 & 0 \\
-1/6 & -1/3 & 4/3 & -1/3 & 0 & 0 & -1/6 & -1/3 & 0 \\
-1/3 & -1/3 & -1/3 & 8/3 & -1/3 & -1/3 & -1/3 & -1/3 & -1/3 \\
0 & -1/6 & 0 & -1/3 & 2/3 & -1/6 & 0 & 0 & 0 \\
0 & -1/3 & 0 & -1/3 & -1/6 & 4/3 & 0 & -1/3 & -1/6 \\
0 & 0 & -1/6 & -1/3 & 0 & 0 & 2/3 & -1/6 & 0 \\
0 & 0 & -1/3 & -1/3 & 0 & -1/3 & -1/6 & 4/3 & -1/6 \\
0 & 0 & 0 & -1/3 & 0 & -1/6 & 0 & -1/6 & 2/3
\end{bmatrix}.
$$

The load vector b can be calculated according to (4), where $f(x,y)$ is the function on the right-hand side in (1), we use $f(x,y) = -2\pi^2 \sin(x\pi)\sin(y\pi)$:

$$
b(1) = -2\frac{4 - 4\pi + \pi^2}{\pi^2}, \quad b(2) = -8\frac{-2+\pi}{\pi^2}, \quad b(3) = -8\frac{-2+\pi}{\pi^2},
$$

(11) $\quad b(4) = -32\frac{1}{\pi^2}, \quad b(5) = -2\frac{4 - 4\pi + \pi^2}{\pi^2}, \quad b(6) = -8\frac{-2+\pi}{\pi^2},$

$$
b(7) = -2\frac{4 - 4\pi + \pi^2}{\pi^2}, \ b(8) = -8\frac{-2+\pi}{\pi^2}, \ b(9) = -2\frac{4 - 4\pi + \pi^2}{\pi^2}.
$$

Since the rank of A is lower as its dimension, the system of equations $Ac = b$ does not have the unique solution.

5.3. Boundary conditions

In order to guarantee the existence of the unique solution, the boundary conditions have to be specified.

For example, consider our model problem given by the following equation in the domain Ω with the boundary Γ:

$$u(x,y)_{xx} + u(x,y)_{yy} = -2\sin(x\pi)\pi^2\sin(y\pi).$$

The Dirichlet boundary conditions can be formulated

$$u(\Gamma) = 0.$$

Then this equation has the unique solution

$$u(x,y) = \sin(\pi x)\sin(\pi y)$$

shown in Fig. 6.

To introduce the boundary conditions in our system of equations we need the indices of coefficients c_i, which correspond to boundaries.

Let us call `getInstances(node)`

```
[> G:=getInstances(node);

  G =
      [00, 1/20, 1/21/2, 01/2, 10, 11/2,
       01, 1/21, 11]
```

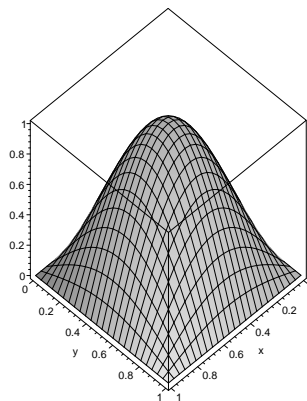We can write a simple procedure to determine the nodes corresponding to the boundaries:

Fig. 6. Exact solution

```
[> getNode:=proc(coordinate, value);
   local g, result:
   use oop in
     result:=[];
     g:=getInstances(node);
     for i from 1 to nops(g) do
        if node||(g[i])||(coordinate||_i)=value then
          result:=[op(result), c[i]]:
        fi:
     od:
   return result:
   end use:
   end:
```

Using

```
[> boundary:={op(getNode(x,0)), op(getNode(x,1)),
                         op(getNode(y,0)), op(getNode(y,1))}
```

we obtain coefficients $\{c_1, c_2, c_3, c_5, c_6, c_7, c_8, c_9\}$, which must be specified.

Now consider the system of equations $Ac = b$, where A and b are given by (10) and (11), respectively, in the following form:

$$-1/3c_4 - 1/6c_6 - 1/6c_8 + 2/3c_9 = -2\frac{4 - 4\pi + \pi^2}{\pi^2},$$

$$-1/3c_2 - 1/3c_4 - 1/6c_5 + 4/3c_6 - 1/3c_8 - 1/6c_9 = -8\frac{-2 + \pi}{\pi^2},$$

$$-1/6c_3 - 1/3c_4 + 2/3c_7 - 1/6c_8 = -2\frac{4 - 4\pi + \pi^2}{\pi^2},$$

$$-1/3c_3 - 1/3c_4 - 1/3c_6 - 1/6c_7 + 4/3c_8 - 1/6c_9 = -8\frac{-2 + \pi}{\pi^2},$$

$$-1/6c_1 + 4/3c_2 - 1/3c_3 - 1/3c_4 - 1/6c_5 - 1/3c_6 = -8\frac{-2 + \pi}{\pi^2},$$

$$-1/6c_1 - 1/3c_2 + 4/3c_3 - 1/3c_4 - 1/6c_7 - 1/3c_8 = -8\frac{-2 + \pi}{\pi^2},$$

$$2/3c_1 - 1/6c_2 - 1/6c_3 - 1/3c_4 = -2\frac{4 - 4\pi + \pi^2}{\pi^2},$$

$$-1/3c_1 - 1/3c_2 - 1/3c_3 + 8/3c_4 - 1/3c_5 - 1/3c_6,$$

$$-1/3c_7 - 1/3c_8 - 1/3c_9 = -32\frac{1}{\pi^2},$$

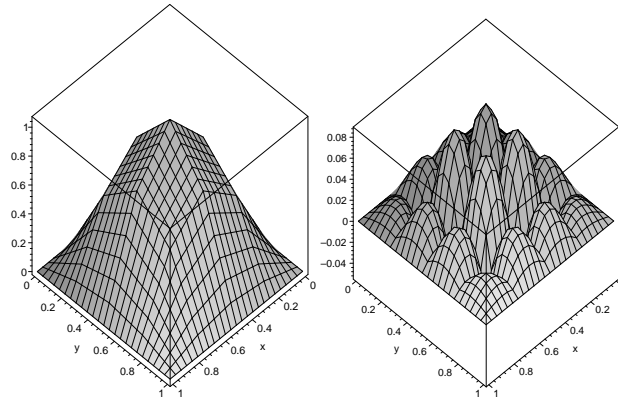$$-1/6c_2 - 1/3c_4 + 2/3c_5 - 1/6c_6 = -2\frac{4 - 4\pi + \pi^2}{\pi^2}.$$

Fig. 7. The numerical solution using $\mathcal{D}_2(\mathbb{R}^2)$ and corresponding error distribution

Since only one unknown coefficient, namely c_4, is independent we must obtain only one equation:

$$8/3c_4 = -32\pi^{-2} + 1/3c_1 + 1/3c_2 + 1/3c_3$$

$$+1/3c_5 + 1/3c_6 + 1/3c_7 + 1/3c_8 + 1/3c_9.$$

Using the Maple command `solve` as shown above, we obtain the solution:

$$c_4 = \frac{1}{8\pi^2}(c_3\pi^2 - 96 + c_1\pi^2 + c_2\pi^2$$

$$+c_8\pi^2 + c_5\pi^2 + c_6\pi^2 + c_7\pi^2 + c_9\pi^2).$$

In case of 16 elements and 25 grid nodes the elimination of dependent equations reduces the stiffness matrix to dimension 9:

$$\begin{pmatrix}
8/3 & -1/3 & 0 & -1/3 & -1/3 & 0 & 0 & 0 & 0 \\
-1/3 & 8/3 & -1/3 & -1/3 & -1/3 & -1/3 & 0 & 0 & 0 \\
0 & -1/3 & 8/3 & 0 & -1/3 & -1/3 & 0 & 0 & 0 \\
-1/3 & -1/3 & 0 & 8/3 & -1/3 & 0 & -1/3 & -1/3 & 0 \\
-1/3 & -1/3 & -1/3 & -1/3 & 8/3 & -1/3 & -1/3 & -1/3 & -1/3 \\
0 & -1/3 & -1/3 & 0 & -1/3 & 8/3 & 0 & -1/3 & -1/3 \\
0 & 0 & 0 & -1/3 & -1/3 & 0 & 8/3 & -1/3 & 0 \\
0 & 0 & 0 & -1/3 & -1/3 & -1/3 & -1/3 & 8/3 & -1/3 \\
0 & 0 & 0 & 0 & -1/3 & -1/3 & 0 & -1/3 & 8/3
\end{pmatrix}.$$

The load vector b is reduced to

$$b(1) = -\frac{16 - 8\pi + \pi^2}{\pi^2} + 2\frac{16\sqrt{2} - 4\sqrt{2}\pi + \pi^2 - 16}{\pi^2}$$

$$-\frac{48 - 8\sqrt{2}\pi - 32\sqrt{2} + \pi^2 + 8\pi}{\pi^2}$$

$$+1/3c_1 + 1/3c_2 + 1/3c_3 + 1/3c_5 + 1/3c_11,$$

$$b(2) = 8\frac{-4\sqrt{2} + \sqrt{2}\pi + 4 - \pi}{\pi^2} - 8\frac{-12 + \sqrt{2}\pi + 8\sqrt{2} - \pi}{\pi^2}$$

$$+1/3c_2 + 1/3c_5 + 1/3c_7,$$

$$b(3) = -\frac{16 - 8\pi + \pi^2}{\pi^2} + 2\frac{16\sqrt{2} - 4\sqrt{2}\pi + \pi^2 - 16}{\pi^2}$$

$$-\frac{48 - 8\sqrt{2}\pi - 32\sqrt{2} + \pi^2 + 8\pi}{\pi^2}$$

$$+1/3c_5 + 1/3c_7 + 1/3c_9 + 1/3c_{10} + 1/3c_{15},$$

$$b(4) = 8\frac{-4\sqrt{2} + \sqrt{2}\pi + 4 - \pi}{\pi^2} - 8\frac{-12 + \sqrt{2}\pi + 8\sqrt{2} - \pi}{\pi^2}$$

$$+1/3c_3 + 1/3c_{11} + 1/3c_{16},$$

$$b(5) = 64\frac{-3 + 2\sqrt{2}}{\pi^2},$$

$$b(6) = 8\frac{-4\sqrt{2} + \sqrt{2}\pi + 4 - \pi}{\pi^2} - 8\frac{-12 + \sqrt{2}\pi + 8\sqrt{2} - \pi}{\pi^2}$$

$$+1/3c_{10} + 1/3c_{15} + 1/3c_{20},$$

$$b(7) = \frac{16\sqrt{2} - 4\sqrt{2}\pi + \pi^2 - 16}{\pi^2} + \frac{-48 + 8\sqrt{2}\pi + 32\sqrt{2} - \pi^2 - 8\pi}{\pi^2}$$

$$-\frac{16 - 8\pi + \pi^2}{\pi^2} - \frac{-16\sqrt{2} + 16 + 4\sqrt{2}\pi - \pi^2}{\pi^2}$$

$$+1/3c_{11} + 1/3c_{16} + 1/3c_{21} + 1/3c_{22} + 1/3c_{23},$$

$$b(8) = \frac{-4\sqrt{2} + \sqrt{2}\pi + 4 - \pi}{\pi^2} - 8\frac{-12 + \sqrt{2}\pi + 8\sqrt{2} - \pi}{\pi^2}$$

$$+1/3c_{22} + 1/3c_{23} + 1/3c_{24},$$

$$b(9) = -\frac{16 - 8\pi + \pi^2}{\pi^2} + 2\frac{16\sqrt{2} - 4\sqrt{2}\pi + \pi^2 - 16}{\pi^2}$$

$$-\frac{48 - 8\sqrt{2}\pi - 32\sqrt{2} + \pi^2 + 8\pi}{\pi^2},$$

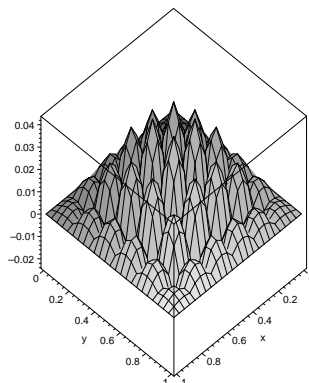$$+1/3c_{15} + 1/3c_{20} + 1/3c_{23} + 1/3c_{24} + 1/3c_{25}.$$

Fig. 8. The error distribution using $\mathcal{D}_4(\mathbb{R}^2)$

Using `LinearSolve` we obtain the solution.

The boundary coefficients are now: $\{c_1, c_2, c_3, c_5, c_7, c_9, c_{10}, c_{11}, c_{15},$
$c_{16}, c_{20}, c_{21}, c_{22}, c_{23}, c_{24}, c_{25}\}$.

The solution vanished on the boundary and its error distribution is shown in Fig. 7. Compare it with the error distribution using $\mathcal{D}_4(\mathbb{R}^2)$ shown in Fig. 8

6. Finite element method using hierarchical basis

The first known mathematician to use hierarchical ideas was Archimedes in "The quadrature of the parabola". By inductively exhausting the parabola with triangles, he was able to measure the area given by a parabola. In 1909 Faber [6] introduced the hierarchical basis and explicitly used it for the representation of functions. Yserentant [5] applied the hierarchical basis in 1986 for numerical methods. In 1990, Zenger [4] directly represented a smooth multivariate function u with a hierarchical tensor product basis instead of a standard nodal basis. The coefficients of this representation, the so-called hierarchical surpluses, decrease with the volume of the support of the corresponding basis functions. In this section we consider hierarchical finite element method on sparse grids and demonstrate how the hierarchical structures can be implemented in OO-Maple.

Let us consider our model problem (1) with $f(x, y) = 2(1-x)y(1-y) - 4xy(1-y) - 2x^2(1-x)$. Then the equation has the exact solution $u(x, y) = 2x^2(1 - x)y(1 - y)$ (Fig. 9).

Similarly to the case of linear basis we can define the class, whose objects would correspond to hierarchical finite elements. This class consists of the following members:
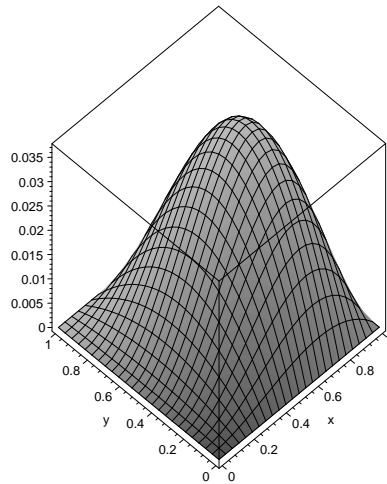
Fig. 9. The exact solution by $f(x,y) = 2(1-x)y(1-y) - 4xy(1-y) - 2x^2(1-x)$

- x_i : x coordinate of the elements middle point
- y_i : y coordinate of the elements middle point
- h_x : the element's half width in x-direction
- h_y : the element's half width in y-direction
- child1, child2, child3, child4: identify the children nodes of this element
- phi: Ansatz function, which takes the value 1 in the middle point of this element

Using hierarchical grid consisting of 4 nodes we obtain the following stiffness matrix A

$$\begin{bmatrix} 8/3 & 1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & 10/3 & 0 & 0 & 0 \\ 1/2 & 0 & 10/3 & 0 & 0 \\ 1/2 & 0 & 0 & 10/3 & 0 \\ 1/2 & 0 & 0 & 0 & 10/3 \end{bmatrix}$$

and load vector b:

$$\begin{bmatrix} -\frac{5}{48} \\ \frac{1}{1536} \\ -\frac{149}{1536} \\ -\frac{37}{768} \\ -\frac{37}{768} \end{bmatrix}.$$
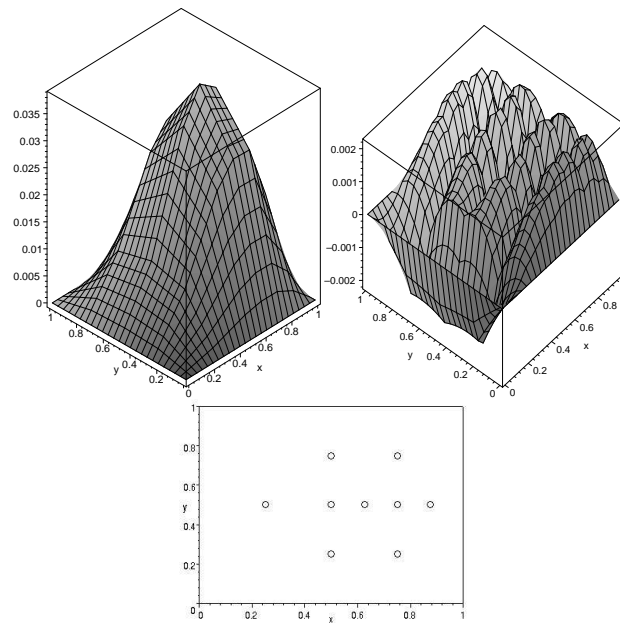
Fig. 10. The numerical solution by $f(x, y) = 2(1 - x)y(1 - y) - 4xy(1 - y) - 2x^2(1 - x)$ using coarse sparse grid and error distribution

The coefficients of this hierarchical representation can be obtained with the aid of `LinearSolve(A,b)`:

$$
\begin{bmatrix}
-\dfrac{289}{9088} \\[2mm]
\dfrac{361}{72704} \\[2mm]
-\dfrac{1769}{72704} \\[2mm]
-\dfrac{11}{1136} \\[2mm]
-\dfrac{11}{1136}
\end{bmatrix} .
$$

These coefficients decrease with the volume of the support of the corresponding basis functions. Consequently, the values of hierarchical surpluses is a very simple criterion for the decision of whether the contribution to the basis representation is important enough or not. This consideration leads to the concept of sparse grids [4] in which we order the basis functions in terms of their support volume. The above class generates exactly the sparse grid.

The main advantage of the hierarchical approach is the possibility to refine adaptively until the hierarchical surplus is lower than the given error tolerance limit.
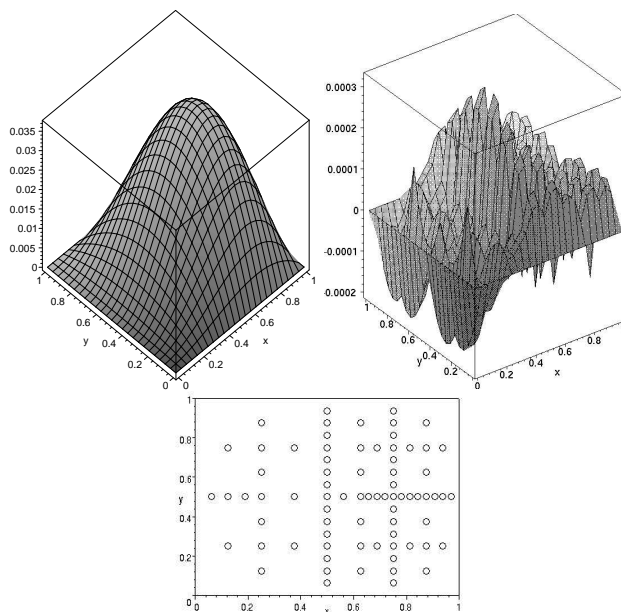
Fig. 11. The numerical solution by $f(x, y) = 2(1 - x)y(1 - y) - 4xy(1 - y) - 2x^2(1 - x)$ using refined sparse grid and error distribution

In this way for example the result shown in Fig. 10, 11 was obtained.

References

1. Chibisov, D. (2003): *Computational Models for Finite Element Simulation Using Maple*, Diploma Thesis, Technical University of Munich, Germany.
2. Hubbard, J. H. and Hubbard, B. B. (1999): *Vector Calculus, Linear Algebra and Differential Forms. A Unified Approach*, Prentice Hall.
3. Ganzha, V. G., Chibisov, D., and Voroztsov, E. V. (2001): GROOME-tool supported graphical object oriented modeling for computer algebra and scientific computing, in: Computer Algebra in Scientific Computing, Proc. 4nd Workshop on Computer Algebra in Scientific Computing (Ganzha, V. G., Mayr, E. W., and Voroztsov, E. V., Eds.), Springer-Verlag , Berlin, 213-232
4. Zenger, C. (1991): Sparse grids, in: Parallel Algorithms for Partial Differential Equations (Hackbush, W., Ed.), Vol. 31, 241-251, Vieweg, Braunschweig/Wiesbaden,
5. Yserentant, H. (1986): On the multi-level splitting of finite element spaces, Numer. Math., **49**, 379–412.
6. Faber, G. (1909): Über stetige funktionen [in German], Mathematische Annalen, **66**, 81–94.