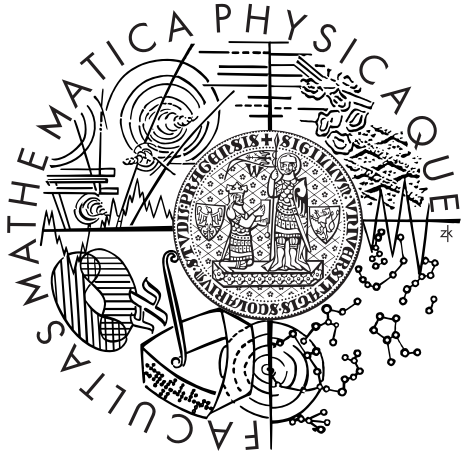
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Kateřina Sládková

Integrace po částech polynomiálních funkcí na sítích typu „non-matching“

Katedra numerické matematiky

Vedoucí bakalářské práce: prof. RNDr. Vít Dolejší, Ph.D., DSc.

Studijní program: Matematika

Studijní obor: Obecná matematika

Praha 2012

Ráda bych poděkovala své rodině za podporu nejen při tvorbě této práce, ale i při celém studiu. Zvláštní poděkování patří Robinu Pokornému za důkladnou recenzi a v neposlední řadě Elvisu Presleymu za hudbu, při jejímž poslechu se i nefunkční algoritmus zdá být skvělým. Poslední díky věnuji člověku, který zavedl bakalářské studium a tím mi umožnil sepsat tuto bakalářskou práci.

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 2.8.2012 Kateřina Sládková

Název práce: Integrace po částech polynomiálních funkcí na sítích typu „non-matching“

Autor: Kateřina Sládková, kackasladkova@gmail.com

Katedra: Katedra numerické matematiky

Vedoucí bakalářské práce: prof. RNDr. Vít Dolejší, Ph.D., DSc., Katedra numerické matematiky, dolejsi@karlin.mff.cuni.cz

Abstrakt: Náplní této práce je numerické řešení časově závislých parcialních diferenciálních rovnic pomocí numerických metod. Pozornost věnujeme zejména případu, kdy se využívají sítě typu „non-matching“ definované v různých časových krocích. V tomto případě je potřeba přepočítat přibližné po částech polynomiální řešení z jedné sítě na druhou. Představujeme algoritmus vyvinutý k tomuto účelu a také několik numerických testů.

Klíčová slova: Integrace, po částech polynomiání funkce, překrývající se sítě

Title: Integration of piecewise polynomial functions on non-matching grids

Author: Kateřina Sládková, kackasladkova@gmail.com

Department: Department of Numerical Mathematics

Supervisor: prof. RNDr. Vít Dolejší, Ph.D., DSc., Department of Numerical Mathematics, dolejsi@karlin.mff.cuni.cz

Abstract: In this thesis we deal with a numerical solution of time-dependent partial differential equations with the aid of numerical methods. Particularly, we focus on case, when different non-matching grids are employed on different time steps. Then piecewise polynomial approximate solution has to be recomputed from one mesh to the second one. We present the developed algorithm as well as several numerical tests.

Keywords: Integration, piecewise polynomial functions, overlapping grids

Contents

Preface

In this thesis we deal with the numerical solution of a scalar nonstationary nonlinear convection-diffusion equation with the aid of the discontinuos Galerkin finite element method (DGFEM). Solution of industrial problems leads to necessity of employing different unstructured non-nested grids for each time level. An example is the so-called full space-time DGFEM. This method requires an evaluation of volume integrals of a product of two functions which are piecewise polynomial on different grids.

In the first chapter we present the convection-diffusion problem as well as DGFEM. By this chapter we prepare theoretical bases for introducing the programmed solution of the volume integration.

Second chapter presents the algorithm of integration we obtained in the previous chapter. This section of thesis is divided into four parts. The first one is devoted to the separation of the task into several independent problems. In the second part we present structures, variables and their use. In the last section we explain the algorithm and code themselves.

Finally third chapter is dedicated to presentation of numerical and graphical verification of our work. Our task is not to connect to whole DGFEM method but to calculate (some kind of) mass conservation by volume integrals over different grids. By this mass conservation we can also prove the funcionality of our method.

1. Theoretical introduction

In this chapter we recall some results from [1] and [2] to briefly remind selected facts from the the theory of convection-diffusion problems and a weak solution. We also present triangulation and semi-discretization (as it was presented in [1] and [2]), which leads us to volume integration of two functions which are piecewise polynomial on different grids.

We would also like to note that we use the standard notation for function spaces (see [3]). Therefore, $L^2(\Omega)$ denote the Lebesgue space and $H^1(\Omega) = W^{k,2}(\Omega)$ are the Sobolev spaces. After all, by H_0^1 we denote the subspace of all functions from $H^1(\Omega)$ with zero traces on $\partial\Omega$.

1.1 Convection-diffusion problem

We present the nonlinear *convection-diffusion problem*. Let $\Omega \in \mathbb{R}^d$ $(d = 2, 3)$ be polygonal bounded domain and let us denote its nonempty boundary by $\partial\Omega$. This boundary consists of two disjoint parts and we assume $\emptyset \neq \partial\Omega = \partial\Omega_D \cup \partial\Omega_N$. Moreover, $T > 0$ and we set $Q_T = \Omega \times (0, T)$.

Furthermore, for the function $\mathbf{F}$, which represents the convective term, we assume that $\mathbf{F}(u) = (F_1(u), \ldots, F_d(u)) : \mathbb{R} \to \mathbb{R}^d, d = 2, 3$, where $\mathbf{F}_s \in C^1(\mathbb{R}), F_s(0) = 0, s = 1, \ldots, d$. For nonlinear function $\mathbf{K}(u) = \{K_{ij}(u)\}_{i,j=1}^d : \mathbb{R} \to \mathbb{R}^{d\times d}$, which represents diffusion term, we assume that $\mathbf{K}$ is bounded and positively definite. In addition $\mathbf{n}$ is outer normal to $\partial\Omega$, $f \in L^2(\Omega \times (0, T))$, u_D is trace of some $u^* \in H^1(\Omega) \cap L^\infty(\Omega)$ on $\partial\Omega_D, u^0 \in L^2(\Omega)$, $f_N \in L^2(\partial\Omega)$. Finally, we assume that the Dirichlet boundary condition is constant with respect to time.

Let us consider following problem: Find $u : Q_T \to \mathbb{R}$ such that

$$\begin{aligned} \frac{\partial u}{\partial t} + \nabla \cdot \mathbf{F}(u) - \nabla \cdot (\mathbf{K}(u)\nabla u) &= f(x,t), \\ u(x,0) &= u^0(x), x \in \Omega \\ u\,\big|_{\partial\Omega_D\times(0,T)} &= u_D, \\ \mathbf{K}(u)\frac{\partial u}{\partial \mathbf{n}}\,\big|_{\partial\Omega_N\times(0,T)} &= f_N, \end{aligned} \tag{1.1}$$

where symbol ∇ means gradient operator and symbol $\nabla\cdot$ represents divergence operator.

1.2 Weak solution

Definition 1.1. We say that function u is the *weak solution* of 1.1, if the following conditions are satisfied:

$$
\begin{aligned}
&\text{a) } u - u^* \in L^2(0,T;H_0^1(\Omega)), u \in L^\infty(Q_T),\\
&\text{b) } \frac{d}{dt}\Big(\int_\Omega u(t)\,w\,dx\Big) + \int_\Omega \nabla\cdot\mathbf{F}(u)\,w\,dx + \int_\Omega [\mathbf{K}(u(t))\,\nabla u(t)]\nabla w\,dx = \int_\Omega f(t)\,w\,dx,\\
&\quad \forall w \in H_0^1(\Omega) \quad \text{in the sense of distributions on } (0,T)\\
&\text{c) } u(0) = u^0 \qquad \text{in } \Omega.
\end{aligned}
\tag{1.2}
$$

By u(t) we denote the function on Ω such that $u(t)(x) = u(x,t), x \in \Omega$.

Let us note that the assumption $u \in L^\infty(Q_T)$ in 1.2a) quarantees the boundednessw of functions $\mathbf{F}$(u) and $\mathbf{K}$(u); thus it assures the existence of integrals in 1.2b). This assumption can be weakened if functions $\mathbf{F}$(u) and $\mathbf{K}$(u) satisfy some growth conditions.

1.3 Triangulations

Let us consider $Q_T = \Omega \times (0,T)$ as it was defined above. Let $0 = t_0 < t_1 < \ldots < t_r = T$ be a partition of $(0,T)$, which generates time intervals $I_m = (t_{m-1}, t_m],\ m = 1,\ldots,r$ of the length $|I_m| = \tau_m$ and let us define τ subsequently $\tau = \max_{m=1,\ldots,r}\tau_m$. At every time level $t_m,\ m = 0,\ldots,r$, we consider generally different space partition $\mathcal{T}_{h,m}$. Each of this space partition consists of a finite number of closed d-dimensional simplices K with mutually disjoint interiors. These simplices covers whole closure $\overline{\Omega}$, i.e., $\overline{\Omega} = \bigcup_{K\in\mathcal{T}_{h,m}} K,\ m = 0,\ldots,r$.

Furthermore, we consider two elements K and K' as neighbours if $(d-1)$-dimensional measure of $K \cap K'$ is positive. By ∂K we denote the boundary of element $K \in \mathcal{T}_{h,m}$ and we set $h_K = diam(K)$, $h_m = \max_{K\in\mathcal{T}_{h,m}} h_K$, $h = \max_{m=0,\ldots,r} h_m$. By $\Gamma_{h,m}$ we denote a union of all edges of partition $\mathcal{T}_{h,m}$, i.e., $\Gamma_{h,m} = \bigcup_{K\in\mathcal{T}_{h,m}} \partial K$. At almost every point of $\Gamma_{h,m}$ it is possible to define a unit normal vector $\mathbf{n} = (n_1,\ldots,n_d)$. For the points in the interior of Ω the orientation of normals is arbitrary. But for the points on the boundary $\partial\Omega$ we assume that the corresponding normals are outer to Ω.

Finally, we denote by

$$
\mathcal{T}_{h,\tau} := \{\mathcal{T}_{h,m},\ m = 0,\ldots,r\} \tag{1.3}
$$

the set of triangulations on all time levels.

1.4 Function spaces

We define the so-called *broken Sobolev spaces* over the triangulations $\mathcal{T}_{h,m},\ m = 0,\ldots,r$ by

$$
H^s(\Omega,\mathcal{T}_{h,m}) = \{w; w|_K \in H^s(K)\ \forall\, K \in \mathcal{T}_{h,m}\},\ m = 0,\ldots,r \tag{1.4}
$$

and let there be the norm $\|w\|^2_{H^s(\Omega,\mathcal{T}_{h,m})} := \sum_{K\in\mathcal{T}_{h,m}} \|w\|^2_{H^s(K)}$ and the seminorm $|w|^2_{H^s(\Omega,\mathcal{T}_{h,m})} := \sum_{K\in\mathcal{T}_{h,m}} |w|^2_{H^s(K)}$.

For $w \in H^1(\Omega, \mathcal{T}_{h,m}),\ m = 0, \ldots, r$, we introduce the following notation on $\Gamma_{h,m} \setminus \partial\Omega$:

$$w_R(x) = \lim_{\delta \to 0+} w(x + \delta \mathbf{n}),\ w_L(x) = \lim_{\delta \to 0-} w(x + \delta \mathbf{n}),$$
$$\langle w \rangle = \frac{1}{2}\left(w_R + w_L\right),\ [w] = w_L - w_R.$$

On $\partial\Omega$ we simply put $\langle w \rangle = [w] = w_L(x) = \lim_{\delta \to 0-} w(x + \delta \mathbf{n})$. The previous limits are considered in the sense of traces.

Finally, we define the spaces of discontinuous piecewise polynomial functions

$$S_{h,p,m} = S^{p,-1}(\Omega, \mathcal{T}_{h,m}) = \{w; w|_K \in P_p(K)\ \forall\, K \in \mathcal{T}_{h,m}\}, \tag{1.5}$$

$m = 0, \ldots, r$, where $P_p(K)$ denotes the space of all polynomials on K of degree $\leq p$, where $p \geq 1$ is a given degree of approximation. Obviously, $S_{h,p,m} \subset H^1(\Omega, \mathcal{T}_{h,m}),\ m = 1, \ldots, r$.

1.5 Space semidiscretization

We present the space semi-discretization of equation (1.1) with the aid of the non-symmetric interior penalty Galerkin variant of DGFEM method. For $m = 1, \ldots, r$ and $u, w \in H^2(\Omega, \mathcal{T}_{h,m})$ we define the forms

$$\begin{aligned}
A_h(u, w) :=& \sum_{K \in \mathcal{T}_{h,m}} \int_K \mathbf{K}(u) \nabla u \,\cdot\, \nabla w \, dx \\
&- \int_{\Gamma_{h,m}} \mathbf{K}(u) \Big(\langle \nabla u \rangle \cdot \mathbf{n}[w] - \langle \nabla w \rangle \cdot \mathbf{n}[u] \Big) dS + \int_{\Gamma_{h,m}} \sigma [u][w] dS, \\
b_h(u, w) :=& \int_{\Gamma_{h,m}} H(u_L, u_R, \mathbf{n})[w] dS - \sum_{K \in \mathcal{T}_{h,m}} \int_K \sum_{s=1}^{d} F_s(u) \frac{\partial w}{\partial x_s} dx \\
l_h(w)\,(t) :=& \,(f(t), w) + \int_{\partial\Omega} \left(\mathbf{K}(u) \nabla w \cdot \mathbf{n}\, u_D + \sigma\, u_D\, w\right) dS
\end{aligned} \tag{1.6}$$

The forms A_h and b_h depend on m, but for simplicity we omit the index. The function H in definition of b_h is the so-called *numerical flux*, which approximates the convective flux by $\mathbf{F}(u) \cdot \mathbf{n} \approx H(u_L, u_R, \mathbf{n})$ on an element face. Here u_L, u_R formally denotes the traces of u on ∂K from the left-hand and right-hand sides of the face from $\Gamma_{h,m}$. On $\partial\Omega$ the values u_R is taken from the boundary condition in the equation 1.1. We shall assume that the numerical flux has the following properties:

1. $H(u, w, \mathbf{n})$ is *Lipschitz-continuous* with respect to u, w,
2. $H(u, w, \mathbf{n})$ is *consistent*, i.e., $H(u, u, \mathbf{n}) = \mathbf{F}(u) \cdot \mathbf{n}$,
3. $H(u, w, \mathbf{n})$ is *conservative*, i.e., $H(u, w, \mathbf{n}) = -H(w, u, \mathbf{n})$

The weight function $\sigma : \Gamma_{h,m} \to \mathbb{R}$ in 1.6 is given by

$$\sigma(x) = (h_K + h_{K'})^{-1}, \tag{1.7}$$

where $K, K' \in \mathcal{T}_{h,m}$ are neighbouring elements sharing x, i.e., $x \in K \cap K'$.

Using the consistency of H, we find that the exact solution u satisfies the identity

$$\Big(\frac{\partial u}{\partial t}, w_h\Big) + A_h(u(t), w_h) + b_h(u(t), w_h) = l_h(w_h)(t) \tag{1.8}$$

for all $w_h \in S_{h,p,m}$ and all $t \in I_m, m = 1, \ldots, r$.

1.6 Volume integration

In order to carry out the full discontinuous Galerkin (DG) space-time discretization, we define the space of space-time piecewise polynomial functions by

$$S_{h,p}^{\tau,q} = \left\{ w \in L^2(Q_T) : w|_{I_m} = \sum_{s=0}^{q} t^s z_s,\ t \in I_m,\ z_s \in S_{h,p,m},\ m = 1, \ldots, r \right\}, \tag{1.9}$$

where $q \geq 0$ is an integer. Moreover, we put

$$\{w\}_m = w_+^m - w_-^m, \qquad w_\pm^m = \lim_{\delta \to 0\pm} w(t_m + \delta), \tag{1.10}$$

denoting a jump of $w \in S_{h,p}^{\tau,q}$.

To derive the definition of approximate solution $U \in S_{h,p}^{\tau,q}$ of 1.2, we integrate equation 1.8 over time interval I_m. We focus on term $\Big(\frac{\partial u}{\partial t}, w_h\Big)$.

$$\int_{I_m} (u', w_h) dt = -\int_{I_m} (u, w_h') + (u, w_h)|_{t_m}^- - (u, w_h)|_{t_{m-1}}^+ \tag{1.11}$$

As you can see, we have integrated the term (u', w_h) over the interval I_m and afterwards we have used the integration by parts. Consequently, we use the thought of upwinding and we approximate the value of u in t_{m-1} from the right by the value in the same point from the left, i.e., $u|_{t_{m-1}}^+ := u|_{t_{m-1}}^-$. Thus we have

$$\begin{aligned}
&\int_{I_m} (u', w_h) dt = -\int_{I_m} (u, w_h')\, dt + (u, w_h)|_{t_m}^- - (u, w_h)|_{t_{m-1}}^- \\
&= -\int_{I_m} (u, w_h')\, dt + (u, w_h)|_{t_m}^- - (u, w_h)|_{t_{m-1}}^+ + (u, w_h)|_{t_{m-1}}^+ - (u, w_h)|_{t_{m-1}}^- \\
&= \int_{I_m} (u', w_h)\, dt + (u, w_h)|_{t_{m-1}}^+ - (u, w_h)|_{t_{m-1}}^- \\
&= \int_{I_m} (u', w_h) + (u|_{t_{m-1}}^+ - u|_{t_{m-1}}^-, w_h')
\end{aligned} \tag{1.12}$$

In the second equality we have just added and taken away same term $((u, w_h)|_{t_{m-1}}^+)$, in the third equality we have used the integration by parts once more but in in-

verse direction. Thus by integration of the equation 1.8 we obtain

$$\begin{aligned}\int_{I_m} \big((u', w) - A_h(u, w) + b_h(u, w)\big)dt + (\{u\}_{m-1}, w_+^{m-1}) \\ = \int_{I_m} l_h(w)dt\end{aligned} \tag{1.13}$$

Definition 1.2. We define *approximate solution* $U \in S_{h,p}^{\tau,q}$ of 1.2 by

$$\begin{aligned}\int_{I_m} \big((U', w) - A_h(U, w) + b_h(U, w)\big)dt + (\{U\}_{m-1}, w_+^{m-1}) \\ = \int_{I_m} l_h(w)dt \qquad \forall w \in S_{h,p}^{\tau,q},\ m = 1, \dots, r.\end{aligned} \tag{1.14}$$

In the equation 1.14, more precisely in the term $(\{U\}_{m-1}, w_+^{m-1})$, we finally get to volume integration of product of two functions which are piecewise polynomial on different grids. By this definition we have prepared the theoretical bases for explaining the algorithm.

2. The algorithm

In this chapter we will explain the programming solution of the volume integration. Our task is to calculate volume integral of a product of two functions which are piecewise polynomial on different grids. To evaluate this integral, we have to do severeal independent steps; thus further in this chapter we present the algorithm step by step.

Let us briefly remind the problem. We have polygonal domain Ω, $T > 0$ and we define $Q_T = \Omega \times (0, T)$. Let $0 = t_0 < t_1 < \ldots < t_r = T$ be a partition of $(0, T)$ which generates time intervals $I_m = (t_{m-1}, t_m],\ m = 1, \ldots, r$. At every time level t_m we consider generally different triangulation $\mathcal{T}_{h,m}$. Each of these triangulations covers whole closure $\overline{\Omega}$. We define two triangulations which are adjacent in time, i.e., we can denote these two triangulations $\mathcal{T}_{h,m-1}, \mathcal{T}_{h,m}$. Without loss of generality we put $m = 2$; therefore we have two grids $\mathcal{T}_{h,1}, \mathcal{T}_{h,2}$. Hereafter the names of variables are the same as in the code.

2.1 The separation to independent problems

First, we have to find the intersection of two grids and describe them as a set of polygons which will be afterwards separated into triangles. Consecutively, we evaluate the product of the polynomial functions and we integrate this product over the triangles.

We have two grids which consist of particular elements, in our case the elements are triangles. Grids $\mathcal{T}_{h,1}$ and $\mathcal{T}_{h,2}$ are composed of finite number of triangles. Let us denote this number by *nelem1* in $\mathcal{T}_{h,1}$ and by *nelem2* in $\mathcal{T}_{h,2}$. In the code itself every triangle in each grid has its own integer identificator which describes it uniquely and sufficiently. As a result of this fact we can obtain any information about the triangle by only knowing the grid and the identificator.

Now, we divide our task into several parts which will be explained further:

1. Find and describe the intersection of one concrete triangle from $\mathcal{T}_{h,1}$, let it have number K (hereafter triangle K), and whole $\mathcal{T}_{h,2}$. More precisely:
   (a) Find one triangle (let it have indentificator equal to I) from $\mathcal{T}_{h,2}$ which have a non-empty intersection withtriangle K.
   (b) Create linked list *link*. Add all neighbours of triangle with identificator I (hereafter triangle I) to *link*. We remind that two elements I and I' are neighbours if $(d-1)$-dimensional measure of $I \cap I'$ is positive.
   (c) Take a triangle from *link*. Delete it from *link*. Count intersection of this triangle and triangle K, if it hasn't been done yet. If the intersection is non-empty add his neighbours to *link*.
   (d) Repeate (c) until the list is empty.
2. The counted intersections are polygons. Separate these polygons into triangles.
3. Count product of the polynomial functions in quadrature nodes.

4. Integrate the product over the triangles.

All this steps are repeated for every triangle in $\mathcal{T}_{h,1}$, that means *nelem1* times.

2.2 Specification of variables

In order to help reader understand our algorithm, we introduce some important structures and variables in this module. It is not the task of our work to connect our module to the code which is applying the DGFEM (ADGFEM code). But we try to make the module in such a way that it will be possible to connect it later and we are also using some parts of this program. So that we present two types of structures. First, these which are used both in the ADGFEM code and our module. As they were prepared for and used in the part of program, which precedes ours, we have no need to use all parts of these structures. Thus we shall present only the parts of these structures which we consider necessary for understanding. Second, we introduce the structures used only in our module and made only for our purpose. So in that case we use the entire structure and we shall explain whole structure.

2.2.1 General structures

This subsection concerns of important common structures such as structure to represent a grid (derived data type *mesh*) and structure to represent a triangle (derived data type *element*).

At first, let us pay attention to type *element*. This type contains several useful data objects, such as *i, face, xc, diam, dof, Qnum*. We can observe all used data objects of element and their use in the following table.

Data object	**Sense**
i	index of element
face - neigh	indices of neighbours
face - idx	indices of nodes
xc	coordinates of barycentre
diam	diameter
dof	number of quadr. functions
Qnum	type of quadrature

Table 2.1: Element's data objects

Next we should observe type *mesh* and it's data objects *nelem, element, x*.

Data object	**Sense**
nelem	number of elements
element	array of elements
x	coordinates of nodes

Table 2.2: Mesh's data objects

2.2.2 Local structures

The most important and probably the only interesting local structures are these which we use to record counted intersections. Different types of linked lists are also more or less important because we use them for storing numbers of triangles which should be counted. But these linked lists are well-known; thus there is no need to introduce them.

However, let us introduce one derived type in which we store intersections of triangle K from $\mathcal{T}_{h,1}$ and whole $\mathcal{T}_{h,2}$ as it was described in section 2.1. We call this derived type *intersect* (hereafter only *intersect*). From this type we can read all necessary information about the intersection. But for better use of this structure there are defined two more structures to help us with the using of *intersect*.

First, it is integer *NumTri* which stores the count of triangles from $\mathcal{T}_{h,2}$ which have nonempty intersection with triangle K. Second, it is an array of integers called *triangles* which stores identificator of every triangle which has a non-empty intersection with triangle K.

Therefore, we can easily move in *intersect* from one triangle with a non-empty intersection to another. From the *intersect* itself we can easily read several information for every triangle of $\mathcal{T}_{h,2}$: whether the intersection with triangle $\underline{\mathrm{K}}$ has been counted; whether it is a non-empty intersection; how many points define this intersection and the points which they are. To study derived type *intersect* more precisely, please see table 2.3.

Data object	**Sense**
arrays: (1–*nelem*)of logical variables	
done(j)	Has the intersection with the j-th triangle been counted?
had(j)	Has the j-th triangle a non-empty intersection with triangle K?
array: (1–*nelem*) of $\{1, \ldots, 6\}$	
NumPnt(j)	number of points defining the intersection with the j-th triangle
array: (1–*nelem*)x(1–*NumPnt*)x(1–2)	
IntPnt	Intersection points 1st position is the identificator of the triangle 2nd is the number of intersection point, 3rd are the physical coordinates of intersection point

Table 2.3: Intersect's data object

2.3 The explanation of the alghoritm

In this section we explain more deeply all steps of our solution. Individual parts will correspond with the separation in section 2.1.

2.3.1 Step 1

Let us focus on the problem of finding the intersection of one triangle K from $\mathcal{T}_{h,1}$ and whole $\mathcal{T}_{h,2}$. This part is calculated in the module by subroutine called *IntersectGridTri*. Once we will have this problem solved then it will be sufficient to repeat this part for all triangles from $\mathcal{T}_{h,1}$. That is to substitute all integers from *1* to *nelem1* for K. To figure out this problem, we have to, at first, know for which triangles we should count the intersection, because counting intersection for every triangle of $\mathcal{T}_{h,2}$ would be inefficient. That means we need to find triangles which have a non-empty intersection, or at least these which have high probability to have a non-empty intersection.

Our first ideas led to make some kind of square grid where every square would remember which triangles have nonempty intersection with it. Then we would find in which squares triangle K is and we would count the intersections of triangle K and all triangles laying in these squares. This solution would probably work, but it will be quite inefficient because we would have to go through whole $\mathcal{T}_{h,2}$ which is necessary anyway. But we would also have to go through whole square grid for every triangle K which is needless in our solution of this particular problem.

The main thought of our solution is to somehow mark out the area where triangle K lies so that we can easily and quickly find triangles from $\mathcal{T}_{h,2}$ which lie in this marked area. Suprisingly, it is sufficient to find only one triangle which satisfies the condition of nonempty intersection with triangle K and afterwards to focus our attention only on his neighbours and on neighbours of his neighbours.

More precisely, we find maximum diameter of triangles lying in $\mathcal{T}_{h,2}$; let us denote this maximum diameter by *MaxDiam*. Then we make something like imaginary circle with radius equal to *MaxDiam* and centre lying in the barycentre of triangle K and we focus on every triangle whose barycentre lies in this circle. Therefore, we go through whole $\mathcal{T}_{h,2}$ element by element (Let us denote current element by triangle I.) and we compare the distance from barycentre of triangle I to barycentre of triangle K with the radius of the circle. If the distance is less than or equal to radius then we count intersection of triangle I and triangle K (subroutine which counts this intersection will be described further in this chapter). If the distance is greater than radius then we move on to next triangle from $\mathcal{T}_{h,2}$.

Moreover, if the counted intersection is nonempty[1] then we increase *NumTri* by one and we also add triangle I to array *triangles*. Finally, we add all neighbours of triangle I to *link* and we stop "the going through" of whole $\mathcal{T}_{h,2}$ there.

[1] In this part we consider everything except empty set as a nonempty intersection. Even one point which has measure equal to zero; thus, the volume integral over this intersection would be equal to zero, too. It's important to add these types of intersections too, because despite the fact that triangle with intersection containing only one point isn't important at all, his neighbours could be important and it can possibly happen that we wouldn't add them to *link* by different mean.

Therefore there ends the part of finding the triangle with nonempty intersection with triangle K. But if the intersection of triangle I and triangle K was empty then we move on to next triangle and continue with "the going through" the grid $\mathcal{T}_{h,2}$ until we find triangle with nonempty intersection with triangle K or until we run out of triangles from $\mathcal{T}_{h,2}$. If it happens, it is probable that some mistake has occured because the range of MaxDiam from the barycentre should be sufficient. But we have implemented something like "the backup parachute" just in case of some peripheral problem. In that case we multiply actual radius of the circle by two and we start up whole part of finding one triangle with nonempty intersection with triangle K once more. If it won't help then we dare to say that triangle K is not in domain which is covered by $\mathcal{T}_{h,2}$.

Now we are situated in the stage when we found 1st triangle from $\mathcal{T}_{h,2}$ which has nonempty intersection with triangle K and we added its neighbours to linked list *link*. Further follows several steps which are repeated until *link* is empty. Here they are:

1. We take 1st triangle from *link*.
2. We count the intersection of this triangle and triangle K and fill appropriate variables.
3. If the intersection is non-empty we add neighbours of this triangle.
4. If the *link* isn't empty go to 1 otherwise end.

By emptying out the *link* we have finished Step 1. There's just one more thing to be done in this section and that's to explain the counting of intersection of two triangles. We will do so in next section.

The intersection of 2 triangles

Let us explain the most difficult part of our task: Finding and describing of intersection of two triangles (triangle I and triangle K). In the code subroutine *IntersectTris* was done to do this job. To count the intersection, we pass following arguments to the subroutine: integers I and K, 2 variables of derived data type mesh *gridK, grid* representing both $\mathcal{T}_{h,2}$ and $\mathcal{T}_{h,1}$, variable of type *intersect* to be filled by evaluated intersection and logical variable *fin*. The variable *fin* can end whole subroutine. If its value is "truth" then we have covered whole area of triangle K and we can end the subroutine).

The main idea of this subroutine is to figure out where the vertices of triangle I lie with respect to triangle K, i.e., how many vertices of triangle I is outside, inside, on the edge or are similar with a vertice of triangle K. According to this information we have a rough idea of what the intersection looks like and which edges of the triangle should have intersection with triangle K. Thus we know which edges of triangle I and triangle K would cross and therefore, should be counted their intersection.

Probably the most thorny problem of all was to add the points which define intersection in right order so that, they make polygons. It was also tough not to forget all peripheral problems, e.g. these, when vertex of triangle I (point B in figure2.1) is really closed to edge of triangle K but it can be still judge as inside

vertex. Despite the fact that intersection with line-made from this vertex to some vertex outside triangle K-and some edge of triangle K should exist, we cannot find any because this intersection is pretty much similar to the vertex inside. Hence as intersection we consider only points 1, A, III from figure 2.1 and there appears a small loss of volume in the results. But as the numerical verification shows the loss is small enough not to influence the evaluation too much.

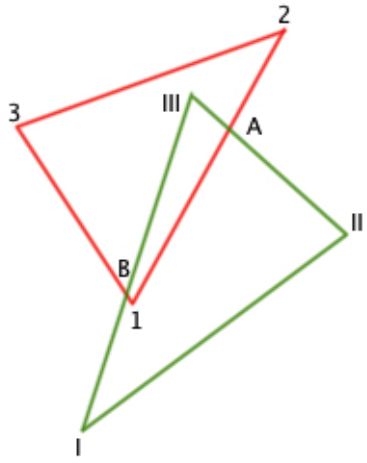


Figure 2.1: Error in case B

Now let us present subroutine *PosNodes* which finds out where vertices of triangle I lie in view of triangle K. Let us denote vertices of triangle I by *nodesi* and vertices of triangle K by *nodesK* (both of these are in the program arrays of real numbers with size $(1:3, 1:2)$). Subroutine *PosNodes* also fills appropriate variables, i.e., array *post* which records position of every vertex of triangle I and other four variables (derived data type *pos*, variables: *edg, ver, ous, ins*) which remember how many of *nodesi* is of the corresponding type (edge, vertex, outside, inside) and which number does this particular vertex has.

This subroutine finds out where all three vertices of triangle I lie but the important part is to find the position of one vertex then it is just repeating of the same thing three times. So let us consider one vertex of the triangle I and let us call it D. First, we compare D with 3rd vertex of nodesK. If they're similar we fill appropriate variables and we're done. Otherwise we count the point where two lines intersect. It is the line from 3rd vertex of nodesK to D and line from 1st to 2nd vertex of nodesK. But for now the important thing isn't the intersection point, not even if the abscissae has the intersection or just the line has it; the important things are the parameters (the unknowns in the system of linear equations) which we use to count the intersection. According to values of these parameters we can decide where D lies. To know which value of the parameters means which position, please observe table 2.4, where the parameters are denoted by $t(1), t(2)$ and *nodesK* by A, B, C, respectively.

Moreover, we present three other important subroutines: *SwapNodes, Inside, FillInOrder*. We pass *nodesi* and integer i as arguments to subroutine *SwapNodes* and this subroutine swaps the order of nodesi so that the node which was on the j-th position is 1st and the node which was on the 1st position is j-th. To the subroutine *Inside* we pass physical coordinates of point x, physical coordinates

t(1)	**t(2)**	**position**
$(0,1)$	$(1,\infty)$	inside
$(-\infty,0);(1,\infty)$	$\mathbb{R}$	outside
$\mathbb{R}$	$(-\infty,0);(0,1)$	
$\mathbb{R}$	1	line AB
1	$\mathbb{R}$	line BC
0	$\mathbb{R}$	line CA
0	1	vertex A
1	1	vertex B
eliminated	before	vertex C

Table 2.4: Position of D in dependence on values of t

of triangle (array (1:3,1:2)) *triang* and logical variable *ins*. With the aid of barycentric coordinates subroutine returns TRUE if the point x lies in *triang*. The subroutine evaluates the barycentric coordinates of x in view of *triang* and if they belong to interval $(0,1)$ then x lies inside the triangle. We have also implemented subroutine *Inside2* which is similar to *Inside* but it returns TRUE also when x lies on the edge of the triangle. The last important is subroutine *FillInOrder* which fills variables *inter%IntPnt* and *inter%NumPnt* with points $x1, x2$ in "right order." Besides mentioned variables we pass to *FillInOrder* edges of triangle K on which the intersections points $x1, x2$ lie. This subroutine places the intersections points so that they are ordered anticlockwise, e.g. first one is the intersection lying on edge which conects vertices 1 and 2 and second one is the intersection lying on edge from vertex 2 to 3. This is important because we hold this direction through whole code so that the intersections create polygons.

We explain the algorithm of intersection of two triangles with aid of flowchart which you can observe on figure 2.3. We carry some special marking in the flowchart. Firstly, "nodesi(j:k)" means j-th and k-th vertex of triangle I. Then we denote the edge of triangle I , i.e., abscissae from nodesi(1) to nodesi(2), by 1&2. In some cases instead of numbers 1 and 2 there can be placed variables restoring integers from 1 to 3. For all figures in flowchart red is reserved for nodesi, green for nodesK and yellow marks the intersection of two triangles.

2.3.2 Step 2

In this subsection we should explain how to make triangles from polygons. Generally, the intersection can contain from one to six points. The interesting intersections are of course these which are composed from three and more points. Because two points make line and one point or line have both measure equal to zero, thus volume integration over them is equal to zero, too. If the intersection is made from three points then it is a triangle and there is no need to separate it into more triangles. The case of four and more points is described on figures 2.2. The vertices of polygon are inscribed by arabic numerals, i.e. $1, 2, \ldots, 6$ and the triangles are inscribed by roman numerals, i.e. I,...,IV. As you can see count of triangles is equal to count of vertices of polygon minus two.

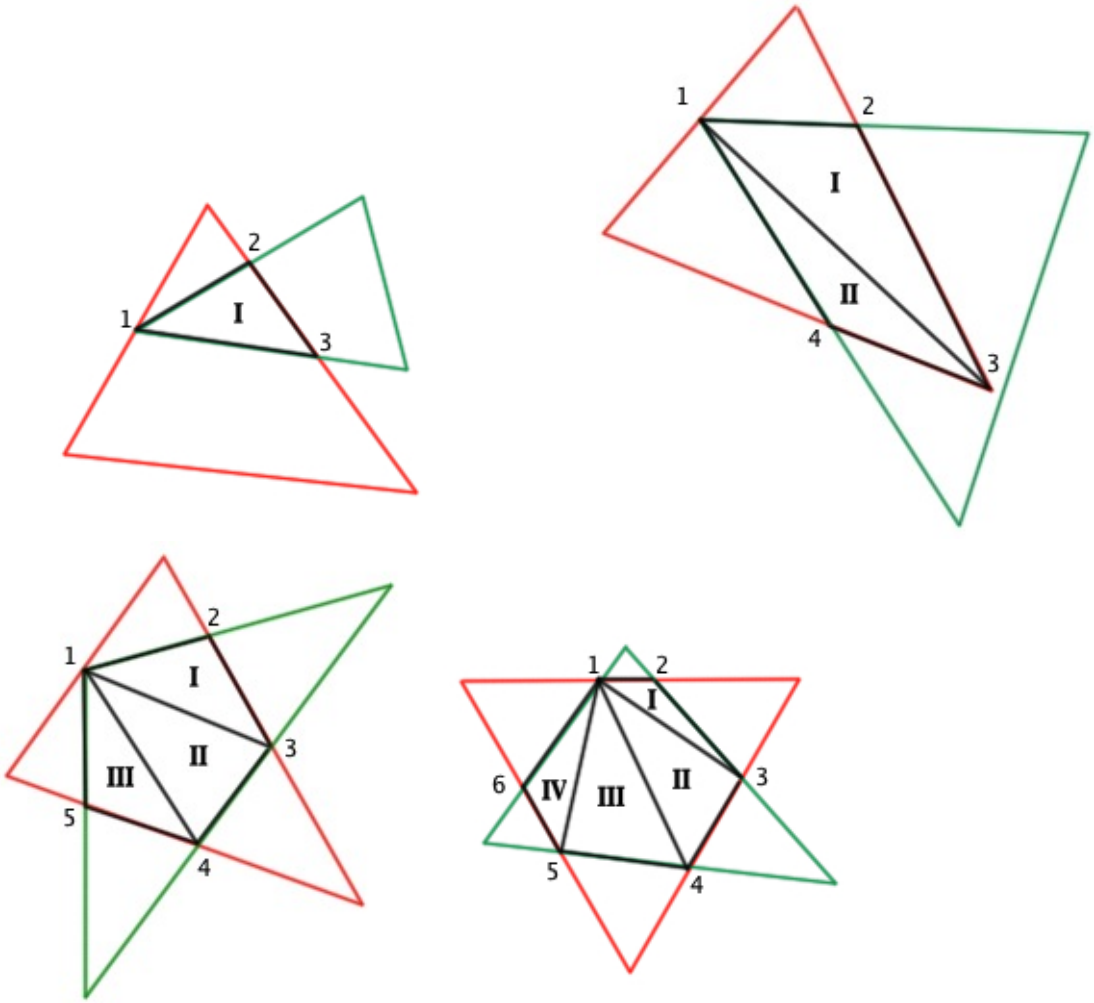
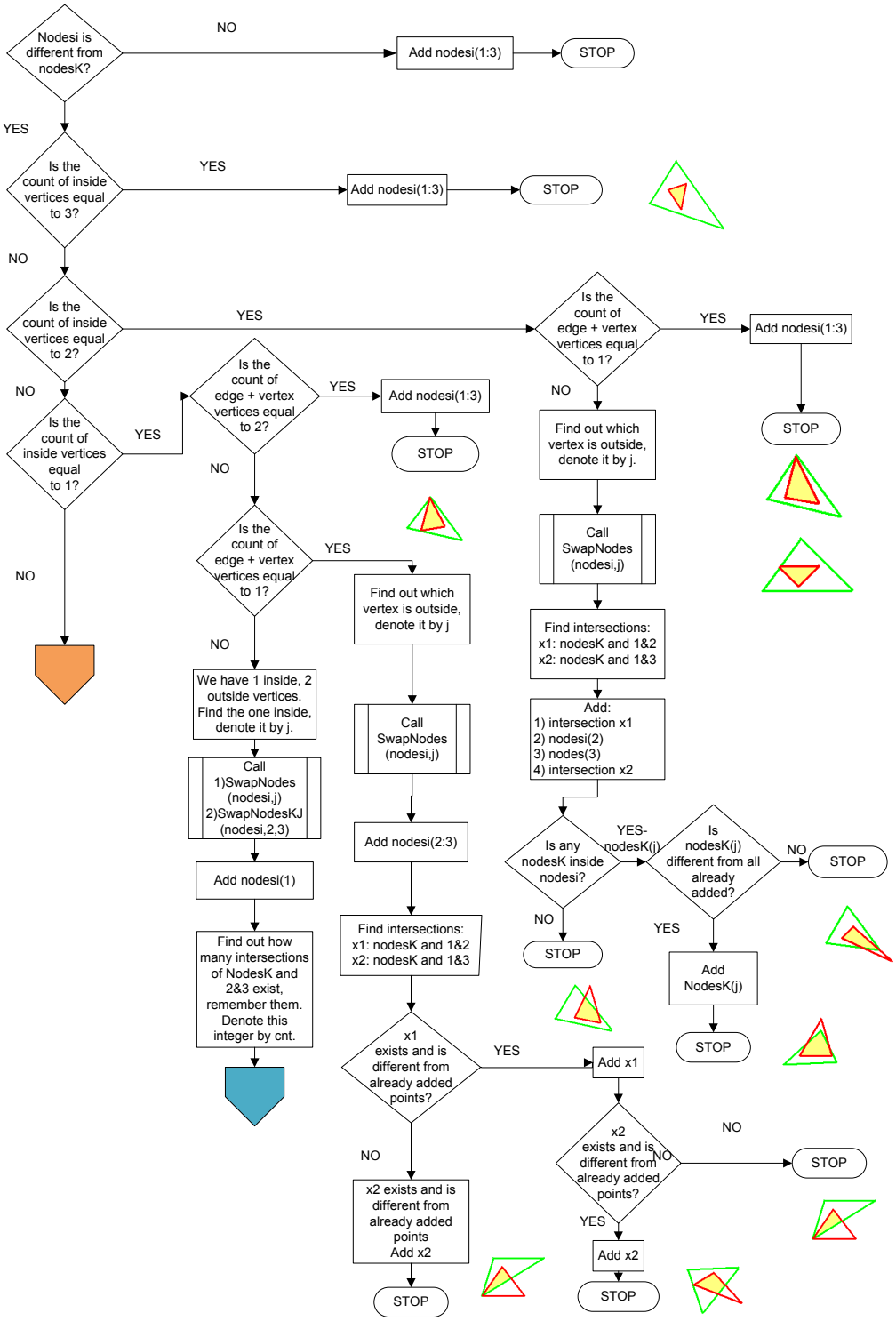


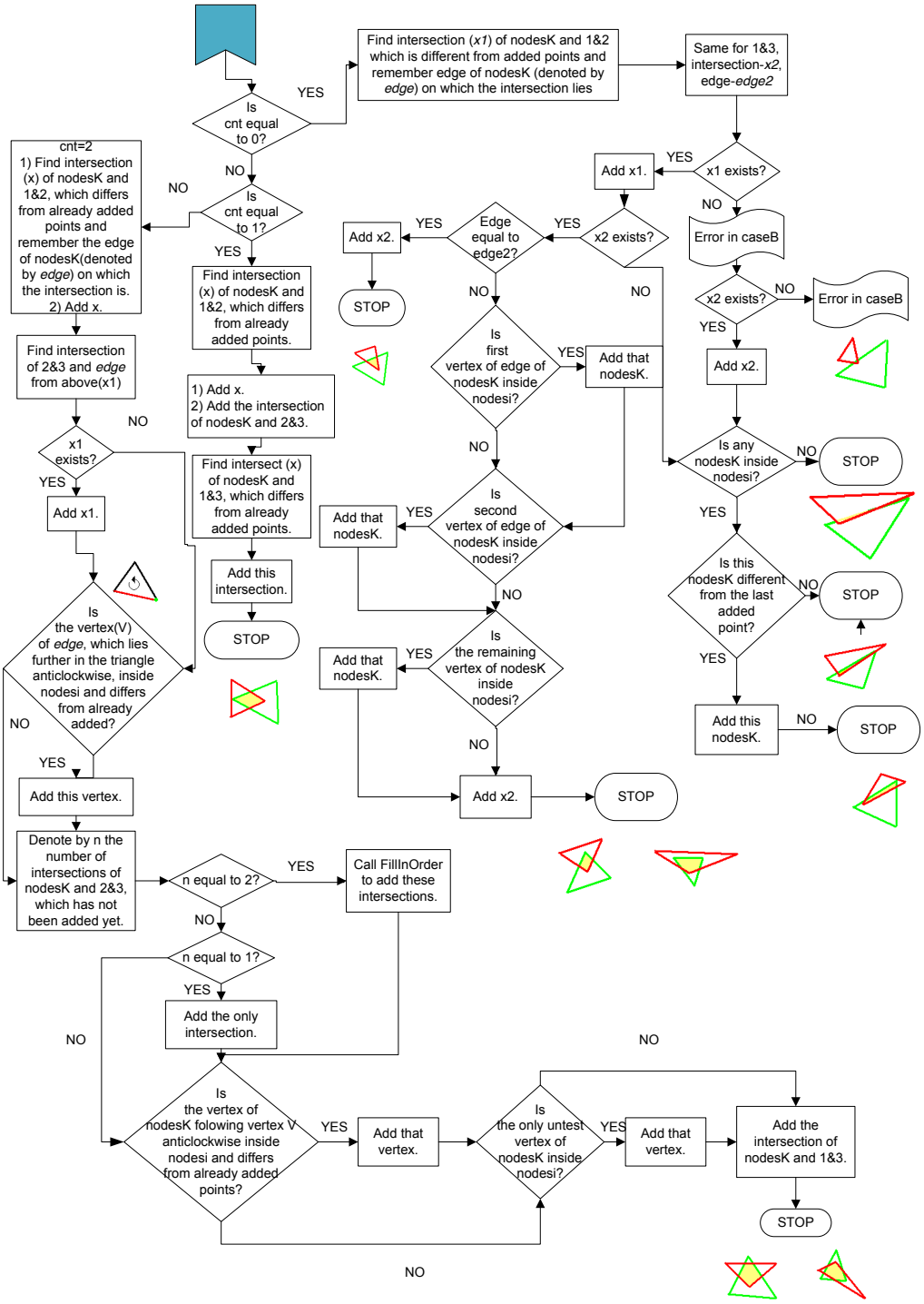
Figure 2.2: The division of polygons into triangles

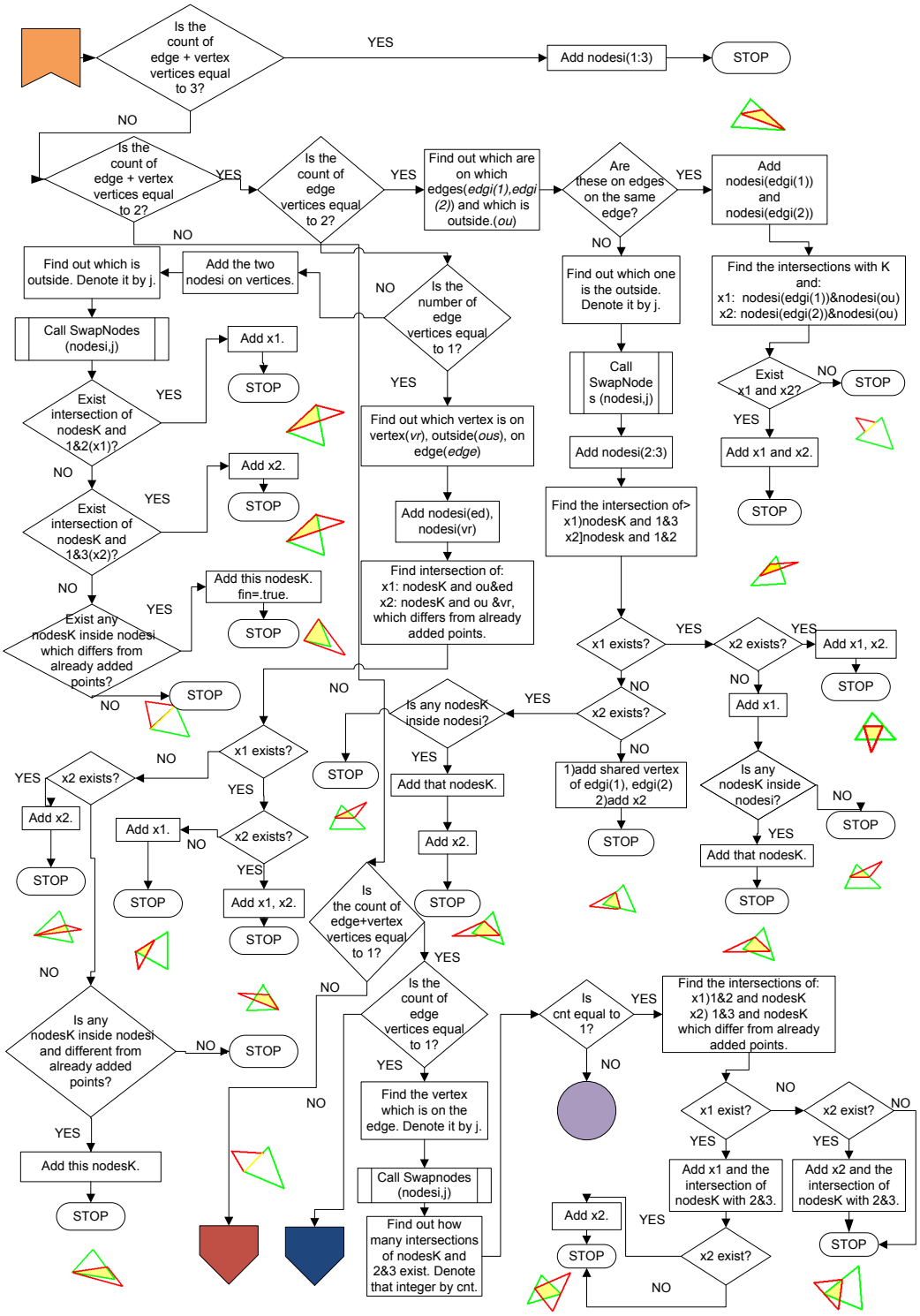
2.3.3 Steps 3 & 4

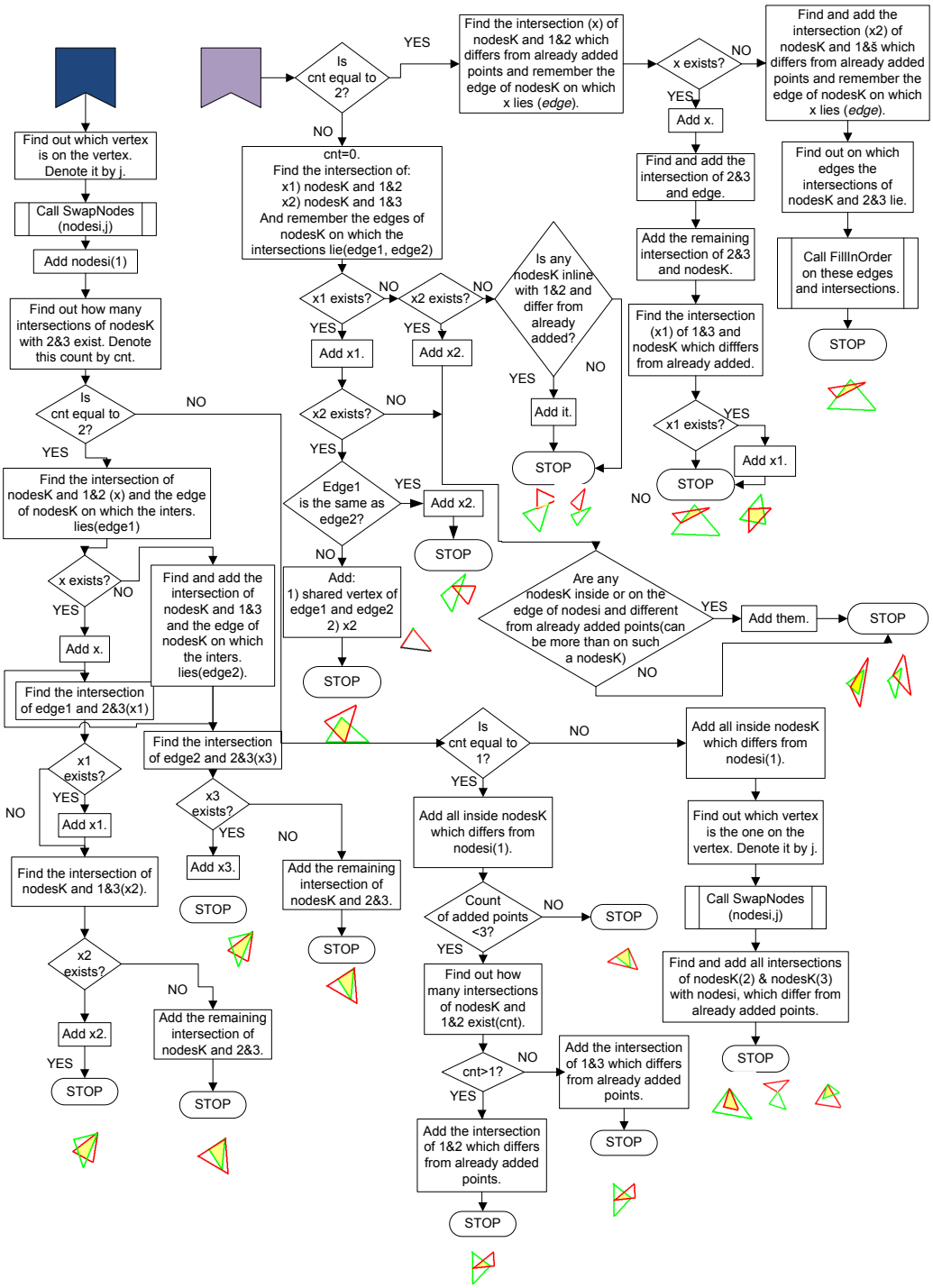
Steps 3 and 4 aren't too complicated. In Step 3 we just have to convert one barycentric coordinates to another. In every triangle of $\mathcal{T}_{h,1}$ and $\mathcal{T}_{h,2}$ we have points in which quadrature functions are known. Let us denote these points by x_i. But after counting the intersections and separating these intersections into triangles we have new triangles. Let us denote them *tris* and the triangles from $\mathcal{T}_{h,1}$ and $\mathcal{T}_{h,2}$ which contains them I and K, respectively. In *tris* we don't know what the values of these quadrature functions are. But *tris* are subsets of triangles from $\mathcal{T}_{h,1}$ and $\mathcal{T}_{h,2}$. So that we take the barycentric coordinates of x_i in *tris* and we calculate the barycentric coordinates of x_i in triangle I and triangle K. Let us denote new barycentric coordinates in triangle I x_{ii} and in triangle K x_{iK}. Afterwards we count the product of these two quadrature functions (defined in x_{ii} and x_{iK}) and we multiply it by the weight function and in the end we count integral of the result over *tris*.

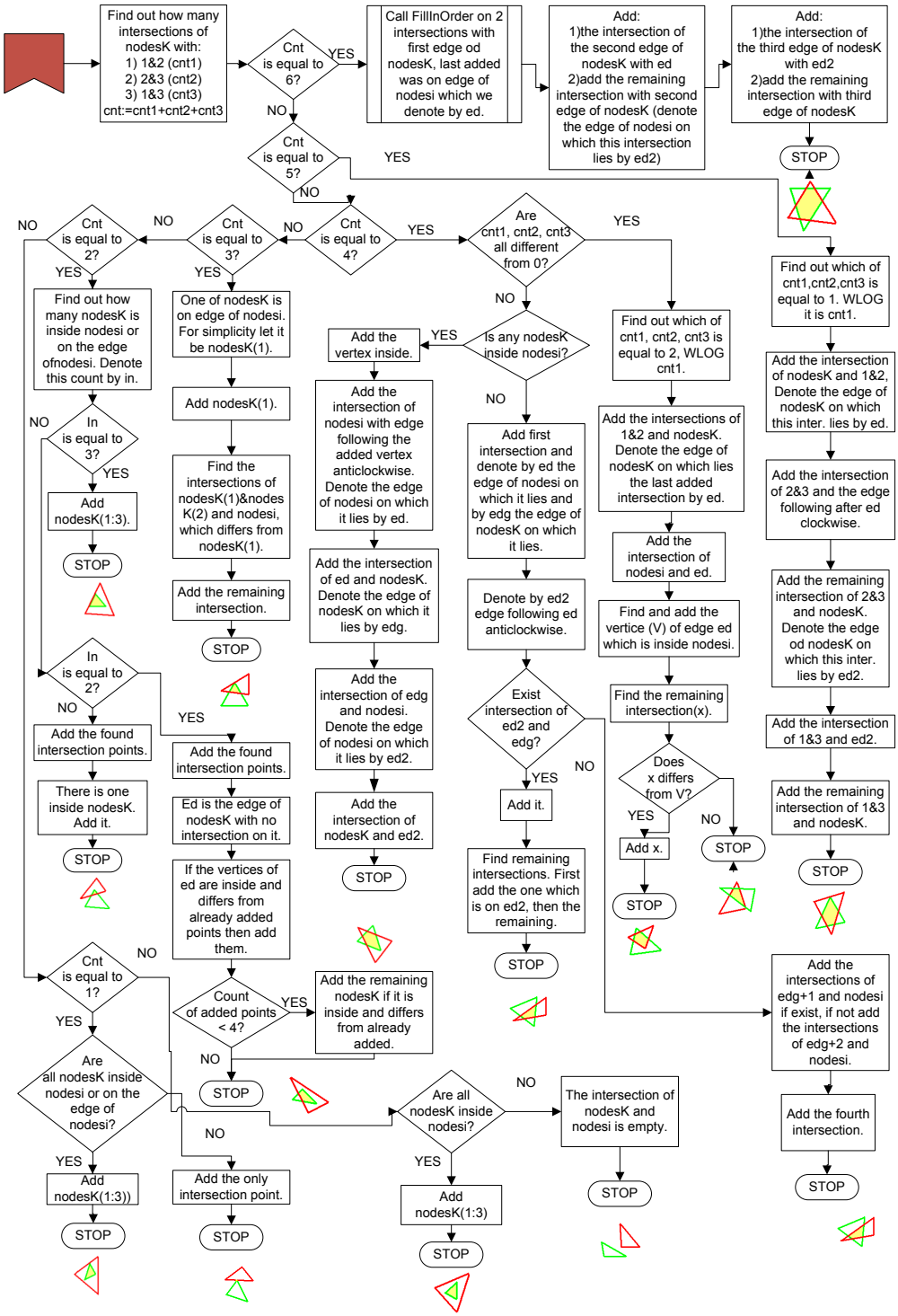
Figure 2.3: The flowchart of the algorithm




Is cnt equal to 0?
YES
Find intersection (x1) of nodesK and 1&2 which is different from added points and remember edge of nodesK (denoted by edge) on which the intersection lies
Same for 1&3, intersection-x2, edge-edge2
NO
Is cnt equal to 1?
cnt=2
1) Find intersection (x) of nodesK and 1&2, which differs from already added points and remember the edge of nodesK(denoted by edge) on which the intersection is.
2) Add x.
Find intersection (x) of nodesK and 1&2, which differs from already added points.
Find intersection of 2&3 and edge from above(x1)
1) Add x.
2) Add the intersection of nodesK and 2&3.
x1 exists?
Add x1.
Find intersect (x) of nodesK and 1&3, which differs from already added points.
Add this intersection.
STOP
Is the vertex(V) of edge, which lies further in the triangle anticlockwise, inside nodesi and differs from already added?
Add this vertex.
Denote by n the number of intersections of nodesK and 2&3, which has not been added yet.
n equal to 2?
Call FillInOrder to add these intersections.
n equal to 1?
Add the only intersection.
Is the vertex of nodesK folowing vertex V anticlockwise inside nodesi and differs from already added points?
Add that vertex.
Is the only untest vertex of nodesK inside nodesi?
Add that vertex.
Add the intersection of nodesK and 1&3.
STOP
x1 exists?
Add x1.
Error in caseB
x2 exists?
Add x2.
Error in caseB
Is any nodesK inside nodesi?
STOP
Is this nodesK different from the last added point?
STOP
Add this nodesK.
STOP
x2 exists?
Add x2.
Edge equal to edge2?
Add x2.
STOP
Is first vertex of edge of nodesK inside nodesi?
Add that nodesK.
Is second vertex of edge of nodesK inside nodesi?
Add that nodesK.
Is the remaining vertex of nodesK inside nodesi?
Add that nodesK.
Add x2.
STOP


Is the count of edge + vertex vertices equal to 3?
YES
Add nodesi(1:3)
STOP
NO
Is the count of edge + vertex vertices equal to 2?
YES
Is the count of edge vertices equal to 2?
YES
Find out which are on which edges(edgi(1),edgi (2)) and which is outside.(ou)
Are these on edges on the same edge?
YES
Add nodesi(edgi(1)) and nodesi(edgi(2))
NO
Find out which one is the outside. Denote it by j.
Call SwapNode s (nodesi,j)
Add nodesi(2:3)
Find the intersection of> x1)nodesK and 1&3 x2]nodesk and 1&2
Find the intersections with K and: x1: nodesi(edgi(1))&nodesi(ou) x2: nodesi(edgi(2))&nodesi(ou)
Exist x1 and x2?
NO
STOP
YES
Add x1 and x2.
STOP
NO
Find out which is outside. Denote it by j.
Add the two nodesi on vertices.
Call SwapNodes (nodesi,j)
Add x1.
STOP
Exist intersection of nodesK and 1&2(x1)?
YES
NO
Add x2.
STOP
Exist intersection of nodesK and 1&3(x2)?
YES
NO
Add this nodesK. fin=.true.
STOP
Exist any nodesK inside nodesi which differs from already added points?
YES
NO
STOP
NO
Is the number of edge vertices equal to 1?
YES
Find out which vertex is on vertex(vr), outside(ous), on edge(edge)
Add nodesi(ed), nodesi(vr)
Find intersection of: x1: nodesK and ou&ed x2: nodesK and ou &vr, which differs from already added points.
x1 exists?
YES
x2 exists?
YES
Add x1, x2.
STOP
NO
Add x1.
Is any nodesK inside nodesi?
YES
NO
x2 exists?
NO
1)add shared vertex of edgi(1), edgi(2) 2)add x2
STOP
Is any nodesK inside nodesi?
NO
STOP
YES
Add that nodesK.
STOP
NO
x1 exists?
YES
NO
x2 exists?
YES
NO
Add x2.
STOP
Add x1.
STOP
x2 exists?
YES
NO
Add x1, x2.
STOP
Add that nodesK.
Add x2.
STOP
Is the count of edge+vertex vertices equal to 1?
YES
NO
Is any nodesK inside nodesi and different from already added points?
NO
STOP
YES
Add this nodesK.
STOP
Is the count of edge vertices equal to 1?
YES
NO
Find the vertex which is on the edge. Denote it by j.
Call Swapnodes (nodesi,j)
Find out how many intersections of nodesK and 2&3 exist. Denote that integer by cnt.
Is cnt equal to 1?
YES
NO
Find the intersections of: x1)1&2 and nodesK x2) 1&3 and nodesK which differ from already added points.
x1 exist?
NO
YES
Add x1 and the intersection of nodesK with 2&3.
x2 exist?
NO
YES
Add x2 and the intersection of nodesK with 2&3.
STOP
Add x2.
STOP
YES
x2 exist?
NO


Find out which vertex is on the vertex. Denote it by j.
Call SwapNodes (nodesi,j)
Add nodesi(1)
Find out how many intersections of nodesK with 2&3 exist. Denote this count by cnt.
Is cnt equal to 2?
NO
YES
Find the intersection of nodesK and 1&2 (x) and the edge of nodesK on which the inters. lies(edge1)
x exists?
NO
YES
Add x.
Find and add the intersection of nodesK and 1&3 and the edge of nodesK on which the inters. lies(edge2).
Find the intersection of edge1 and 2&3(x1)
x1 exists?
YES
Add x1.
NO
Find the intersection of nodesK and 1&3(x2).
x2 exists?
NO
Add x2.
YES
STOP
Add the remaining intersection of nodesK and 2&3.
STOP
Find the intersection of edge2 and 2&3(x3)
x3 exists?
YES
Add x3.
STOP
NO
Add the remaining intersection of nodesK and 2&3.
STOP
Is cnt equal to 2?
YES
NO
cnt=0.
Find the intersection of:
x1) nodesK and 1&2
x2) nodesK and 1&3
And remember the edges of nodesK on which the intersections lie(edge1, edge2)
x1 exists?
NO
YES
Add x1.
x2 exists?
NO
YES
Edge1 is the same as edge2?
YES
Add x2.
STOP
NO
Add:
1) shared vertex of edge1 and edge2
2) x2
STOP
x2 exists?
NO
YES
Add x2.
STOP
Find the intersection (x) of nodesK and 1&2 which differs from already added points and remember the edge of nodesK on which x lies (edge).
Is any nodesK inline with 1&2 and differ from already added?
YES
NO
Add it.
STOP
Are any nodesK inside or on the edge of nodesi and different from already added points(can be more than on such a nodesK)
YES
Add them.
STOP
NO
Is cnt equal to 1?
NO
YES
Add all inside nodesK which differs from nodesi(1).
Count of added points <3?
NO
STOP
YES
Find out how many intersections of nodesK and 1&2 exist(cnt).
cnt>1?
NO
YES
Add the intersection of 1&2 which differs from already added points.
STOP
Add the intersection of 1&3 which differs from already added points.
STOP
Add all inside nodesK which differs from nodesi(1).
Find out which vertex is the one on the vertex. Denote it by j.
Call SwapNodes (nodesi,j)
Find and add all intersections of nodesK(2) & nodesK(3) with nodesi, which differ from already added points.
STOP
x exists?
NO
YES
Add x.
Find and add the intersection of 2&3 and edge.
Add the remaining intersection of 2&3 and nodesK.
Find the intersection (x1) of 1&3 and nodesK which difffers from already added.
x1 exists?
YES
Add x1.
NO
STOP
Find and add the intersection (x2) of nodesK and 1&š which differs from already added points and remember the edge of nodesK on which x lies (edge).
Find out on which edges the intersections of nodesK and 2&3 lie.
Call FillInOrder on these edges and intersections.
STOP


Find out how many intersections of nodesK with: 1) 1&2 (cnt1) 2) 2&3 (cnt2) 3) 1&3 (cnt3) cnt:=cnt1+cnt2+cnt3
Cnt is equal to 6?
YES
NO
Call FillInOrder on 2 intersections with first edge od nodesK, last added was on edge of nodesi which we denote by ed.
Add: 1)the intersection of the second edge of nodesK with ed 2)add the remaining intersection with second edge of nodesK (denote the edge of nodesi on which this intersection lies by ed2)
Add: 1)the intersection of the third edge of nodesK with ed2 2)add the remaining intersection with third edge of nodesK
STOP
Cnt is equal to 5?
YES
NO
Cnt is equal to 4?
YES
NO
Cnt is equal to 3?
YES
NO
Cnt is equal to 2?
YES
NO
Are cnt1, cnt2, cnt3 all different from 0?
YES
NO
Find out which of cnt1,cnt2,cnt3 is equal to 1. WLOG it is cnt1.
Add the intersection of nodesK and 1&2, Denote the edge of nodesK on which this inter. lies by ed.
Add the intersection of 2&3 and the edge following after ed clockwise.
Add the remaining intersection of 2&3 and nodesK. Denote the edge od nodesK on which this inter. lies by ed2.
Add the intersection of 1&3 and ed2.
Add the remaining intersection of 1&3 and nodesK.
STOP
Add the intersections of edg+1 and nodesi if exist, if not add the intersections of edg+2 and nodesi.
Add the fourth intersection.
STOP
Find out which of cnt1, cnt2, cnt3 is equal to 2, WLOG cnt1.
Add the intersections of 1&2 and nodesK. Denote the edge of nodesK on which lies the last added intersection by ed.
Add the intersection of nodesi and ed.
Find and add the vertice (V) of edge ed which is inside nodesi.
Find the remaining intersection(x).
Does x differs from V?
YES
NO
Add x.
STOP
STOP
Is any nodesK inside nodesi?
YES
NO
Add first intersection and denote by ed the edge of nodesi on which it lies and by edg the edge of nodesK on which it lies.
Denote by ed2 edge following ed anticlockwise.
Exist intersection of ed2 and edg?
YES
NO
Add it.
Find remaining intersections. First add the one which is on ed2, then the remaining.
STOP
Add the vertex inside.
Add the intersection of nodesi with edge following the added vertex anticlockwise. Denote the edge of nodesi on which it lies by ed.
Add the intersection of ed and nodesK. Denote the edge of nodesK on which it lies by edg.
Add the intersection of edg and nodesi. Denote the edge of nodesi on which it lies by ed2.
Add the intersection of nodesK and ed2.
STOP
One of nodesK is on edge of nodesi. For simplicity let it be nodesK(1).
Add nodesK(1).
Find the intersections of nodesK(1)&nodesK(2) and nodesi, which differs from nodesK(1).
Add the remaining intersection.
STOP
Add the found intersection points.
Ed is the edge of nodesK with no intersection on it.
If the vertices of ed are inside and differs from already added points then add them.
Count of added points < 4?
YES
NO
Add the remaining nodesK if it is inside and differs from already added.
STOP
Add the only intersection point.
STOP
Find out how many nodesK is inside nodesi or on the edge ofnodesi. Denote this count by in.
In is equal to 3?
YES
NO
Add nodesK(1:3).
STOP
In is equal to 2?
YES
NO
Add the found intersection points.
There is one inside nodesK. Add it.
STOP
Cnt is equal to 1?
YES
NO
Are all nodesK inside nodesi or on the edge of nodesi?
YES
NO
Add nodesK(1:3))
STOP
Are all nodesK inside nodesi?
YES
NO
Add nodesK(1:3)
STOP
The intersection of nodesK and nodesi is empty.
STOP

3. Numerical verification

In this chapter we demonstrate the funcionality of the program and whole thought of the volume integration of a product of two functions which are piecewise polynomial on different grids. As it was presented in previous chapters, while applying DGFEM it turns out to be neccesary to evaluate volume integrals product of two functions which are piecewise polynomial on different grids. As we do not apply DGFEM as entirity we found a different mean how to present the results of our work. As main thought of proving the funcionality we take the law of conservation mass.

For simplicity we consider little bit different equation from the one presented in chapter 1. In this case we know the exact solution of this equation which is useful for verification of our results. We present the scalar nonlinear convection-diffusion equation

$$-\nabla \cdot (\mathbf{K}(u)\nabla u) - \frac{\partial u^2}{\partial x_1} - \frac{\partial u^2}{\partial x_2} = f \qquad \text{in } \Omega := (0,1)^2, \tag{3.1}$$

where $\mathbf{K}(u)$ is nonsymmetric matrix given by

$$\mathbf{K}(u) = \epsilon \begin{pmatrix} 2 + \arctan(\mathrm{u}) & (2\text{-}\arctan(\mathrm{u}))/4 \\ 0 & (4 + \arctan(\mathrm{u}))/2 \end{pmatrix},$$

the parameter $\epsilon > 0$ plays a role of an amount of diffusitivity and we put $\epsilon = 10^{-3}$. We prescribe a Dirichlet boundary condition on $\partial\Omega$ and set the source term f such that the exact solution is

$$u(x_1, x_2) = (x_1^2 + x_2^2)^{\alpha/2} x_1 x_2 (1 - x_1)(1 - x_2), \qquad \alpha \in \mathbb{R}. \tag{3.2}$$

Please observe pictures 3.1 and 3.2 to see how the solution looks like with different parameter α.

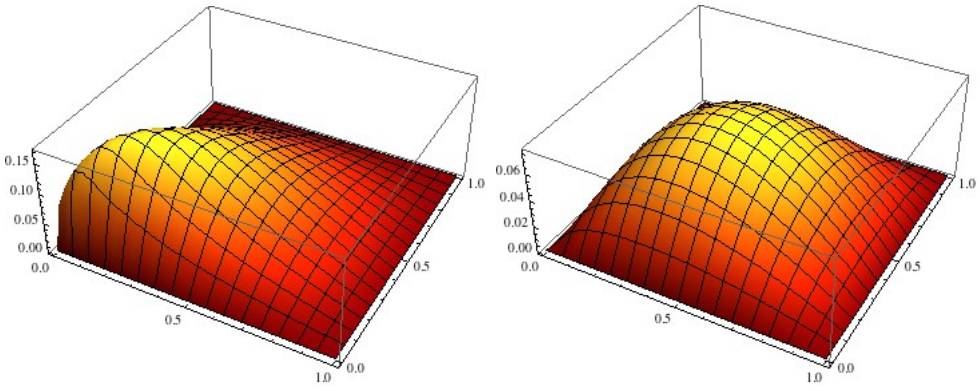


Figure 3.1: Exact solution with parameter $\alpha = -1.5$ and $\alpha = -0.5$

As it was presented before the main thought of verification of our results is based on the law of conservation mass. This law says that the mass m of the volume $V(t)$ does not depend on time. The mass of the volume is defined as an integral from density over volume, i.e., $m(V(t), t) = \int_{V(t)} \rho(x, t)dx$. We start the computation with the aid of ADGFEM code on a mesh $\mathcal{T}_{h,1}$ and obtain a

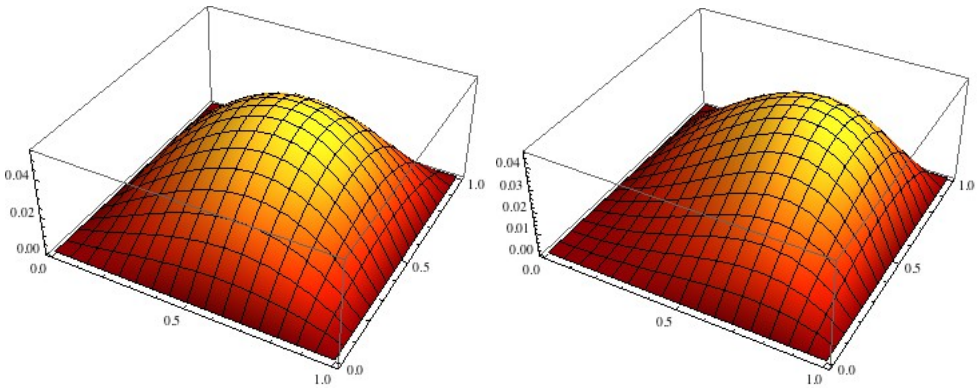


Figure 3.2: Exact solution with parameter $\alpha = 0.5$ and $\alpha = 1.5$

piecewise polynomial approximate solution w_h. Then by the ADGFEM code, we generate a new grid $\mathcal{T}_{h,2}$ and interpolate function w_h on the new grid $\mathcal{T}_{h,2}$ by the L^2-projection. In this projection we need to integrate two piecewise polynomial functions given on different grids, hence we employ the algorithm developed in previous chapters. We investigate the difference between piecewise polynomial function w_h given on $\mathcal{T}_{h,1}$ and its projection $\tilde{w}_h$ on $\mathcal{T}_{h,2}$ which in fact represents a conservation of mass in the interpolation process (if the quantity w represents a density).

We briefly remind that $\mathcal{T}_{h,1}$ consists of finite number of triangles, i.e. $\mathcal{T}_{h,1} = \bigcup_{j=1}^{nelem1} K_j$, where $nelem1$ is number of triangles in $\mathcal{T}_{h,1}$. Let us define function $w_h \in S_{h,p,1}$, for the definition of this space see equation 1.5 in section 1.4. We interpolate function w_h on $\mathcal{T}_{h,2}$; interpolated function $\tilde{w}_h \in S_{h,p,2}$ and therefore it has form $\tilde{w}_h = \sum_j \tilde{w}_j \tilde{\varphi}_j$ where $\tilde{\varphi}_j$ are functions which are piecewise polynomial of degree p on elements of $\mathcal{T}_{h,2}$. The interpolation is defined by L^2 projections subsequently:

$$(\tilde{w}_h, \tilde{\varphi}_i) = (w_h, \tilde{\varphi}_i) \qquad \forall \tilde{\varphi}_i \in S_{h,p,2}. \tag{3.3}$$

Moreover, we have

$$\sum_j \tilde{w}_j(\tilde{\varphi}_j, \tilde{\varphi}_i) = (w_h, \tilde{\varphi}_i) = \sum_j \tilde{w}_j(\varphi_j, \tilde{\varphi}_i). \tag{3.4}$$

The equation 3.4 can be represented in the matrix form as M $\tilde{W} = R$, where $R = \sum_j \tilde{w}_j(\varphi_j, \tilde{\varphi}_i)$, M is the so-called mass matrix with elements $(\tilde{\varphi}_j, \tilde{\varphi}_i)$ and $\tilde{W}$ is the vector consisting of coefficient $\tilde{w}_i$ of function $\tilde{w}_h$. We have to solve this equation for every testing function which means to count the product of two piecewise polynomial functions $\tilde{\varphi}_j, \tilde{\varphi}_i$ which are polynoms of degree p (functions $\tilde{\varphi}_j$ are piecewise polynomial on $\mathcal{T}_{h,2}$ and functions φ_i on $\mathcal{T}_{h,1}$). Consequently, we find out the inverse matrix and we evaluate the product of two matrices. Now we have found the function $\tilde{w}$. Finally, we evaluate the integral of $\tilde{w}$ over $\mathcal{T}_{h,1}$ and over $\mathcal{T}_{h,2}$ and denote these two results as w_1 and w_2:

$$w_1 = \sum_{K \in \mathcal{T}_{h,1}} \frac{1}{|K|} \int_K w \, dx; \qquad w_2 = \sum_{I \in \mathcal{T}_{h,2}} \frac{1}{|I|} \int_I w \, dx, \tag{3.5}$$

where I, K are the elements of appropriate triangulations and $|K|, |I|$ means measure of element K and I.

As the result of the numerical verification we evaluate the difference between these two functions and in the following tables we present the results of this computation. Just for the tables let us denote the maximum diameter of all elements by H and the minimum diameter by h. Furthermore $nelem1$ stays 288 constantly but $nelem2$ changes and you can observe these changes in the tables, too.

We present results where the second grid $\mathcal{T}_{h,2}$ was obtained by two different settings in the ANGENER code, namely setting $numel = 1000$ in the first table and $numel = 4000$ in the second table.

α	nelem2	H	h	p	w_1	w_2	$w_1 - w_2$
-1.5	1405	7.578E-2	2.370E-2	1	1.140591E-1	1.140591E-1	-4.584444E-12
-1.5	1397	7.599E-2	2.525E-2	2	1.127597E-1	1.127597E-1	-4.163326E-17
-1.5	1388	7.623E-2	2.106E-2	3	1.123454E-1	1.123454E-1	-1.804112E-17
-0.5	978	8.463E-2	4.610E-2	1	6.717706E-2	6.717706E-2	9.714451E-17
-0.5	982	8.459E-2	4.457E-2	2	6.725752E-2	6.725752E-2	-4.163336E-2
-0.5	974	8.460E-2	4.475E-2	3	6.731427E-2	6.731427E-2	5.551115E-17
0.5	744	9.941E-2	4.83E-2	1	4.715243E-2	4.715243E-2	4.762960E-10
0.5	725	0.105	4.789E-2	2	4.724920E-2	4.724920E-2	6.245005E-17
0.5	750	0.105	4.702E-2	3	4.727498E-2	4.727498E-2	-6.245005E-17
1.5	505	0.136	4.783E-2	1	3.676470E-2	3.676470E-2	4.163336E-17
1.5	531	0.137	4.294E-2	2	3.676337E-2	3.676337E-2	6.245005E-17
1.5	517	0.137	4.282E-2	3	3.675132E-2	3.675132E-2	-1.387779E-16

Table 3.1: Results for grids with seting $numel = 1000$.

α	nelem2	H	h	p	w_1	w_2	$w_1 - w_2$
-1.5	3072	6.304E-2	1.234E-2	1	1.140591E-2	1.140591E-1	4.579670E-16
-1.5	3072	6.732E-2	1.368E-2	2	1.127597E-1	1.127597E-1	1.387779E-17
-1.5	3077	6.498E-2	1.603E-2	3	1.123454E-1	1.123454E-1	-4.163336E-16
-0.5	3071	5.151E-2	2.095E-2	1	6.717706E-2	6.717706E-1	-2.160494E-13
-0.5	3072	5.128E-2	2.057E-2	2	6.725752E-2	6.725752E-2	-1.110223E-16
-0.5	3073	5.308E-2	2.082E-2	3	6.731427E-2	6.731427E-2	-1.029593E-13
0.5	2828	5.289E-2	2.349E-2	1	4.715243E-2	4.715243E-2	-7.632783E-17
0.5	2826	5.057E-2	1.318E-2	2	4.724920E-2	4.724920E-2	1.318390E-16
0.5	2814	5.379E-2	2.392E-2	3	4.727498E-2	4.727498E-2	7.632783E-17
1.5	2114	7.510E-2	2.324E-2	2	3.676470E-2	3.676470E-2	2.081668E-17
1.5	2119	7.591E-2	2.102E-2	1	3.676337E-2	3.676337E-2	8.326673E-17
1.5	2136	7.410E-2	2.215E-2	3	3.675132E-2	3.675132E-2	-6.938894E-17

Table 3.2: Results for grids with setting $numel = 4000$.

Moreover we present some graphs of the solution. For all figures red is reserved for $\mathcal{T}_{h,2}$ and green for $\mathcal{T}_{h,1}$.

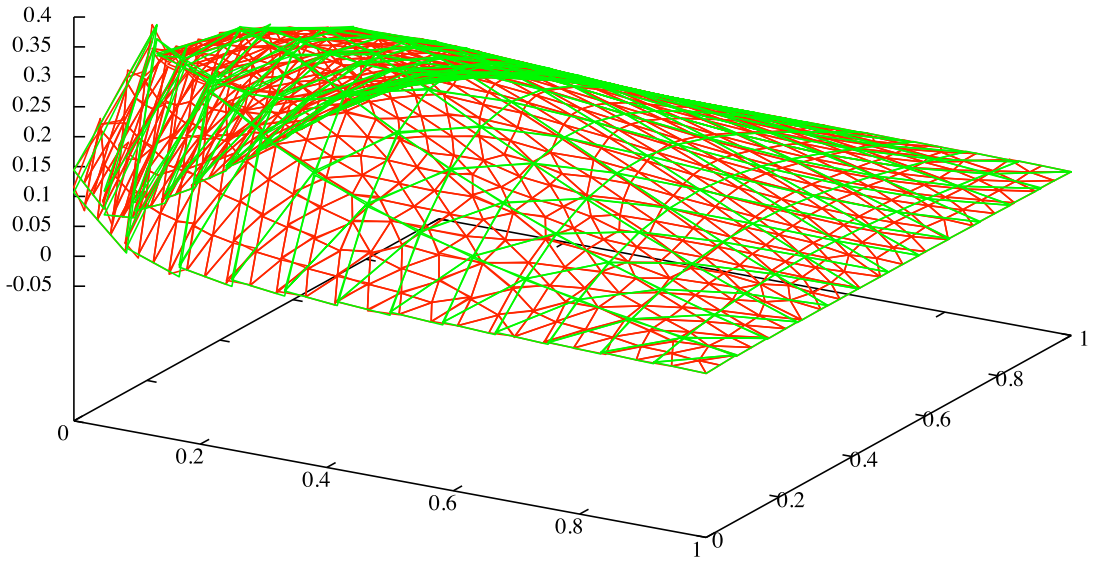


Figure 3.3: 3D graph of the solution, $\alpha = -1.5$; $p = 1$, approximately $nelem2 = 1000$

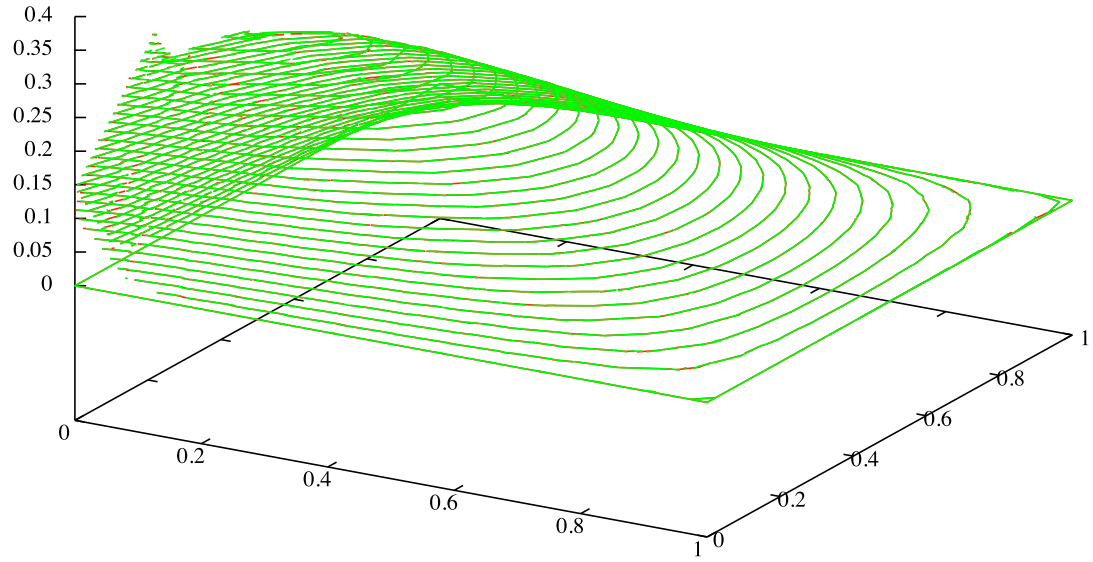


Figure 3.4: Iso-curve of the solution, $\alpha = -1.5$; $p = 1$

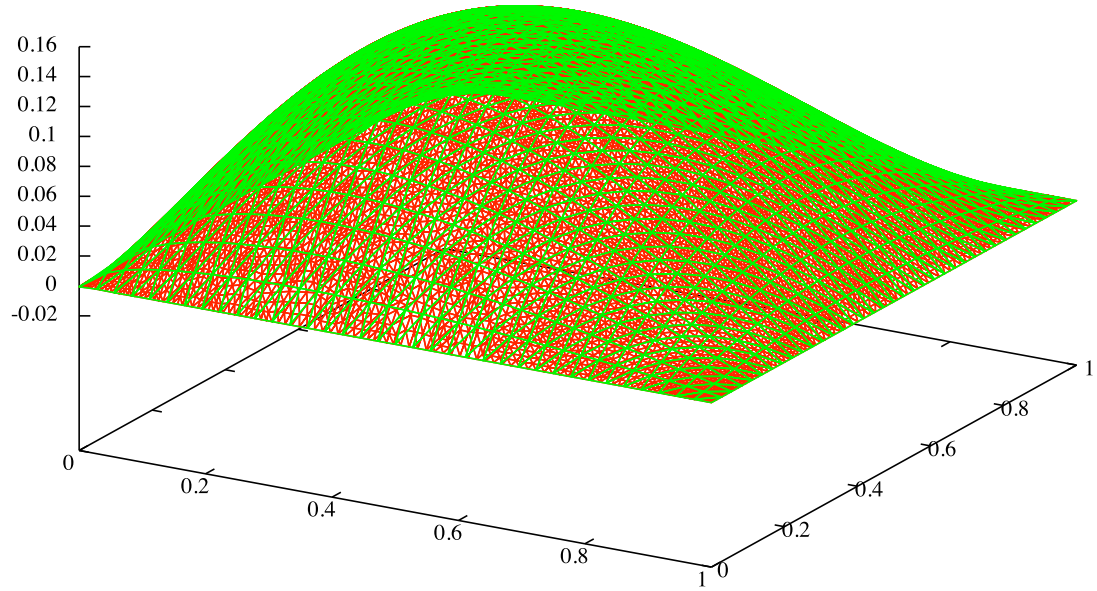


Figure 3.5: 3D graph of the solution, $\alpha = -0.5$; $p = 3$, approximately $nelem2 = 4000$

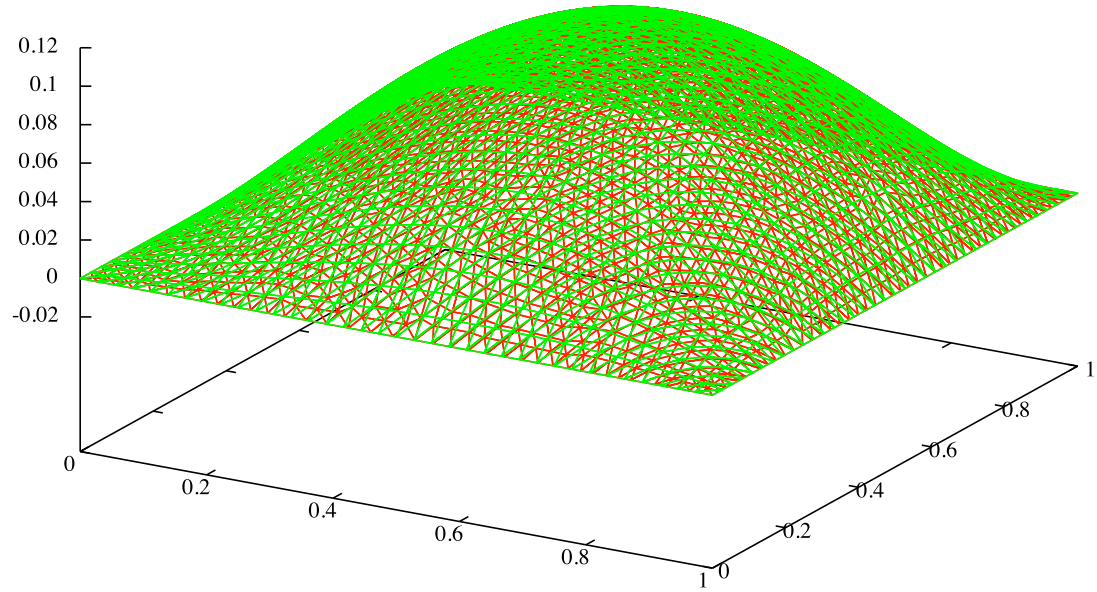


Figure 3.6: 3D graph of the solution, $\alpha = 0.5$; $p = 3$, approximately $nelem2 = 1000$

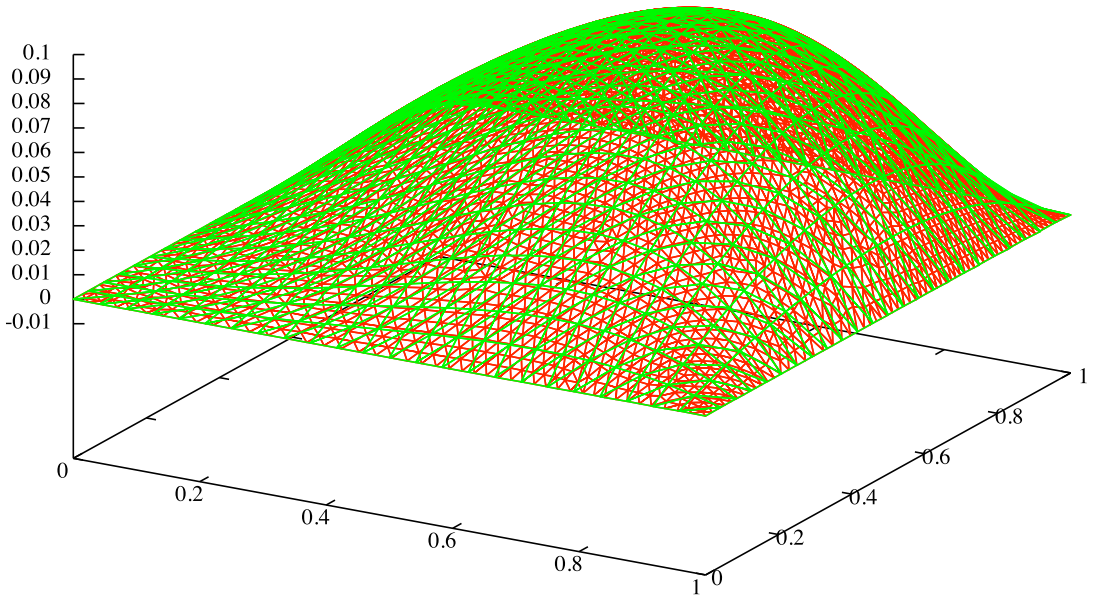


Figure 3.7: 3D graph of the solution, $\alpha = 1.5$; $p = 2$, approximately $nelem2 = 4000$

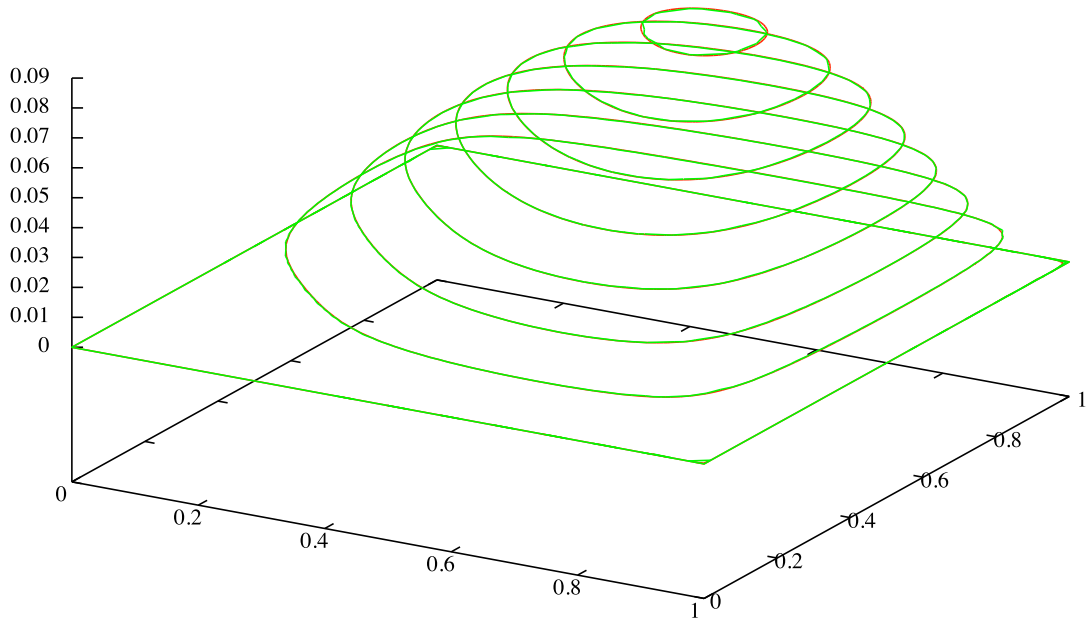


Figure 3.8: Iso-curve of the solution, $\alpha = 1.5$; $p = 2$

Bibliography

[1] Vlasák, M., Dolejší, V., Hájek, J. *A Priori Error Estimates of an Extrapolated Space-Time Discontinuous Galerkin Method for Nonlinear Convection-Diffusion Problems*. Published online in Wiley InterScience (www.interscience.wiley.com), 2010.

[2] Dolejší, Vít *hp-DGFEM for nonlinear convection diffusion problems*. Preprint submitted to Mathematics and Computers in Simulation, 2012.

[3] Kufner, Alois, John, Oldřich, Fučík, Svatopluk *Function spaces*. 1. vydání Academia, Prague, 1977. 454p. ISBN 90 286 0015 9