# A Posteriori Error Estimates for *hp*-FEM: Comparison and Implementation

Bachelor Thesis

*written by*
Elke Spindler

*supervised by*
Prof. Dr. R. Hiptmair
Seminar for Applied Mathematics
ETH Zurich

Springsemester 2010

---

Abstract

This thesis first gives an introduction to finite element methods. Later on, different *hp*-adaptive stategies will be discussed and compared theoretically and by numerical experiments.

---

# Contents

# 1 Introduction

This paper first gives an introduction to the construction of the *finite element method* (FEM). We introduce a *general second order elliptic boundary value problem* (BVP), deduce the *weak formulation* of it and give a criterion for the existence and uniqueness of the solution. After that we discretize our problem and derive a sufficient criterion to get existence as well as uniqueness of the solution.
Furthermore, the concept of the *finite element space* on a *mesh* is introduced. We explain the different refinement strategies and when to choose which.

Later on, the *basic adaptive algorithm* is explained, which should give an idea of the structure of all the algorithms discussed. Sections 4 and 5 are going to give a more detailed explanation of the substeps of the basic algorithm:

Section 4 introduces the following two *a posteriori* error estimators:

- a *residual error estimator* taking advantage of certain properties of the Legendre polynomials.
- an *equilibrated error estimator* gained by a famous theorem called *hypercircle method*

Reliability and Efficiency is proven for both estimators. The second one also shows $p$-robustness – an important, but rarely found property. It guarantees the error estimator to remain efficient even at high polynomial degree.

In section 5 two marking strategies, called *maximum strategy* and *(weighted) fixed energy fraction* and two $hp$-deciders are introduced and explained. The $hp$-decider consists of the following ideas:

- a method which estimates the analyticity of the solution by looking at the decay of the coefficients of its Legendre series.
- an approach solving a minimalization problem to get the best refinement method on each element

After the theory is set, a complete MATLAB framework for 1D-$hp$-FEM is constructed. By numerical experiments, the algorithms are tested extensively and the obtained results are discussed and compared with the theory.

# 2 A basic overview of FEM

## 2.1 The model problem for elliptic differential equations

In this section we introduce a classical second order elliptic boundary value problem, without making any assumptions and restrictions about the regularity of the involved domains but boundedness. The statements, definitions and theorems made here are found in most of the literature written about FEM. This introduction is based on the detailed explanations in [8], [1] and [6].

The classical second order elliptic BVP consists of a functional equation and boundary conditions.
Let $\Omega \subset \mathbb{R}^d$, $\mathbf{C} \in C^0\left(\overline{\Omega}, \mathbb{R}^{d,d}\right) \cap C^1\left(\Omega, \mathbb{R}^{d,d}\right)$, symmetric and uniformly positive definite, $\mathbf{b} \in C^0\left(\overline{\Omega}, \mathbb{R}^d\right)$, $c \in C^0\left(\overline{\Omega}, \mathbb{R}\right)$, uniformly positive, $d \in \{1, 2, 3\}$ we define the linear Operator $\mathcal{L}$ as follows:

$$\mathcal{L}u = -\operatorname{div}\left(\mathbf{C}\,\mathbf{grad}\,u\right) + \mathbf{b} \cdot \mathbf{grad}\,u + cu \quad \text{for } u \in C^0\left(\overline{\Omega}, \mathbb{R}\right) \cap C^2\left(\Omega, \mathbb{R}\right) \tag{1}$$

Now, let $f \in C^0\left(\overline{\Omega}, \mathbb{R}\right)$, $\overline{\Gamma_D} \cup \overline{\Gamma_N} = \partial\Omega$, $\Gamma_D \cap \Gamma_N = \emptyset$, $g \in C^0\left(\overline{\Gamma_D}, \mathbb{R}\right)$ and $h \in C^0\left(\overline{\Gamma_N}, \mathbb{R}\right)$. Then *the classical elliptic boundary value problem* is defined in the following way:

Find $u \in C^0\left(\overline{\Omega}, \mathbb{R}\right) \cap C^2\left(\Omega, \mathbb{R}\right)$ such that

$$\begin{aligned} \mathcal{L}u &= f && \text{in } \Omega \\ u &= g && \text{on } \Gamma_D \\ \mathbf{n} \cdot \mathbf{C}\,\mathbf{grad}\,u &= -h && \text{on } \Gamma_N \end{aligned} \tag{2}$$

where $n$ denotes the exterior unit normal vector to $\Omega$. The equations $(2)_2$, $(2)_3$ are called Dirichlet, respectively Neumann boundary conditions.

We call this formulation classical, since all equations are assumed to hold pointwise. But is the pointwise definiton sufficient to capture the most of physically meaningful solutions of such problems? The answer is no. There are much more physically meaningful solutions than one gets by solving this classical BVP. For example in (2) $\mathbf{C}$ was assumed to be $C^1$ to get the equation $(2)_1$ to be pointwise well defined. But in nature there are a lot of problems with simply continuous or even discontinuous conductivity C.
Another problem might arise on the boundary. Since the boundary conditions are just defined on $\Gamma_N$ and $\Gamma_D$, which are open in $\partial\Omega$, there might be some points $P \in \overline{\Gamma_D} \cap \overline{\Gamma_N}$, on which it is not specified which boundary condition to take.
Hence, we have to change the pointwise equations into equations in distributional sense to be able to capture the solutions mentioned above, i.e. to weaken the sense in which the equations of (2) hold.

## 2.2 The variational formulation of the second order elliptic problem

The discussion above leads us to a distributional reformulation of the classical second order elliptic BVP, the *variational formulation*:

$$\text{Find } u \in U: \ a(u,v) = f(v) \ \forall\, v \in V \tag{3}$$

This formulation can be constructed from equation $(2)_1$ by multiplying $v \in V$ and then integrating over the whole domain $\Omega$. The space $V$ is called *test space* and $U$ *trial space* which are always assumed to be Banach spaces. In this paper, we will always assume that $f \in L^2(\Omega)$ and set $U = V = H^1_{\Gamma_D}(\Omega)$, the *Sobolev space of square integrable functions*, defined as follows:

$$\begin{aligned} H^1(\Omega) &= \left\{v \in L^2(\Omega) |\ \mathbf{grad}\, v \in L^2(\Omega)\right\} \\ H^1_{\Gamma_D}(\Omega) &= \left\{v \in L^2(\Omega) |\ \mathbf{grad}\, v \in L^2(\Omega) \wedge T_1 v = 0 \text{ on } \Gamma_D\right\} \end{aligned} \tag{4}$$

where $T_1 \colon H^1(\Omega) \to H^{1/2}(\partial\Omega)$ is the *trace operator*, and $H^{1/2}(\partial\Omega)$ is given by:

$$\begin{aligned} H^{1/2}(\partial\Omega) = \{v \in L^2(\partial\Omega) |\ |v|^2_{H^{1/2}(\partial\Omega)} \\ = \int_\Omega \int_\Omega \frac{|v(\mathbf{x}) - v(\mathbf{y})|}{|\mathbf{x}-\mathbf{y}|^{d+1}} dS(\mathbf{x}) dS(\mathbf{y}) < \infty\} \end{aligned} \tag{5}$$

In our case, $a(u,v)$ and $f(v)$ will always have the form

$$\begin{aligned} a(u,v) &= \int_\Omega \mathcal{L}(u)(\mathbf{x}) v(\mathbf{x}) d\mathbf{x} + \int_{\Gamma_N} \mathbf{C}\,\mathbf{grad}\, u(\mathbf{x}) \cdot \mathbf{n} v(\mathbf{x}) dS(\mathbf{x}) \\ &= \int_\Omega -\operatorname{div}\left(\mathbf{C}\,\mathbf{grad}\, u(\mathbf{x})\right) v(\mathbf{x}) + \mathbf{b} \cdot \mathbf{grad}\, u(\mathbf{x}) v(\mathbf{x}) + c u(\mathbf{x}) v(\mathbf{x}) d\mathbf{x} \\ &\quad - \int_{\Gamma_N} h v(\mathbf{x}) dS(\mathbf{x}) \\ &= \int_\Omega \mathbf{C}\,\mathbf{grad}\, u(\mathbf{x}) \cdot \mathbf{grad}\, v(\mathbf{x}) + \mathbf{b} \cdot \mathbf{grad}\, u(\mathbf{x}) v(\mathbf{x}) + c u(\mathbf{x}) v(\mathbf{x}) dx \\ f(v) &= \int_\Omega f(\mathbf{x}) v(\mathbf{x}) d\mathbf{x} + \int_{\Gamma_N} h(\mathbf{x}) v(\mathbf{x}) dS(\mathbf{x}) - a(\tilde{g}(\mathbf{x}), v(\mathbf{x})) \end{aligned} \tag{6}$$

where $T_1 \tilde{g} = g$ on $\Gamma_D$.
The last line of the derivation of $a(u,v)$ in (6) is obtained by applying integration by parts (IBP) to the first term. By convention, the boundary term occuring from the IBP is written into $f(v)$, therefore we get an additional boundary term at the beginning of (6).
The additional term $a(\tilde{g}, v)$ in (6) ensures that the Dirichlet boundary conditions hold. Simply solving the BVP without this additional term would give us a solution $\tilde{u}$ which is equal to zero on $\Gamma_D$. To get now the demanded function $u$, which fulfills also $(2)_2$, we have to add some function $\tilde{g} \in H^1(\Omega)$ to $\tilde{u}$ which comes up to $g$ on $\Gamma_D$ and can be derived by applying $T_1^{-1}$ to $g$. Since this $\tilde{g}$ may not be zero in $\Omega$, we have to subtract $a(\tilde{g}, v)$ in $(2)_1$ to ensure that $u$ still satisfies the equation.

**Remark:** In this formulation, the boundary conditions are integrated into the variational formulation (6), such that we do not need to include any additional equations.

The Sobolev space $H^1_{\Gamma_D}(\Omega)$ guarantees existence and uniqueness of the solution. To derive this statement, let us first introduce some definitions.

A continuous sesquilinear form $a\colon V \times V \to \mathbb{R}$ on some Hilbert space $V$ is called $V$*-elliptic*, if

$$\exists \alpha > 0 \,:\, a(v,v) \leq \alpha \|v\|_V^2 \;\forall\, v \in V \tag{7}$$

The norm induced by such a $V$-elliptic sesquilinear form is called *energy norm*.

A continuous sesquilinear form $a\colon U \times V \to \mathbb{R}$, for $U$, $V$ Hilbert spaces, is meant to fulfill *the inf-sup conditions*, if

$$\exists \gamma > 0 :$$

$$\inf_{u \in U\setminus\{0\}} \sup_{v \in V\setminus\{0\}} \frac{|a(u,v)|}{\|u\|_U \|v\|_V} \geq \gamma \tag{8}$$

$$\forall\, v \in V\setminus\{0\} :$$

$$\sup_{u \in U\setminus\{0\}} |a(u,v)| > 0 \tag{9}$$

Further let $V'$ be the dual space of $V$, i.e. the space of all continuous linear functionals on $V$ and

$$\begin{aligned} A\colon U &\to V' \\ u &\mapsto a(u,\cdot) \end{aligned} \tag{10}$$

the operator associated with the sesquilinear form $a$. Then from (8) results injectivity of $A$ and closedness of $Im(A(U)) \subset V'$. Moreover (9) gives additionally the density of $Im(A(U)) \subset V'$ which yields together with the closedness the surjectivity of A. Thus we get bijectivity of A, which proves the existence and uniqueness of the solution $u \in U$ such that (3) holds. One can easily see, that if some sesquilinear form $a$ is $V$-elliptic, $a$ also satisfies the inf-sup conditions.

But what about the $H^1_{\Gamma_D}(\Omega)$-ellipticity of our $a$ from (6)? It follows directly from the Poincaré-Friedrichs inequalities. They state that for all choices of boundary conditions, in particular for the mixed conditions, we get for $\Omega \subset \mathbb{R}^d$, and $\mu(\Gamma_D) > 0$:

$$\exists\, C > 0 \,:\, \|u\|_{L^2(\Omega)} \leq C \|\,\mathbf{grad}\, u\|_{L^2(\Omega)} \;\forall u \in H^1_{\Gamma_D}(\Omega) \tag{11}$$

where $C$ is only depending on properties of the domain $\Omega$. This immediately involves the $H^1_{\Gamma_D}(\Omega)$-ellipticity of $a$.

## 2.3 Galerkin discretization of the variational formulation

The spaces $U$ and $V$ from problem (3) are mostly not finite dimensional. Hence, to be able to do numerical calculations on (3), we have to discretize $U$ and $V$. We do this by simply taking finite dimensional subspaces $U_N \subset U$ and $V_N \in V$ such that $dim(U_N) = m, dim(V_N) = n < \infty$. Moreover we require that the inf-sup conditions have to hold also in the discrete case to guarantee existence and uniqueness of the solution. The discrete problem, called *Galerkin discretization* now looks as follows:

$$\text{Find } u_N \in U_N \text{ such that } a(u_N, v_N) = f(v_N) \;\forall\, v_N \in V_N \tag{12}$$

where $a$, $f$ remain the same as in (3). By taking $\{\varphi_1, \varphi_2, ..., \varphi_m\}$ as a basis of $U_N$ and $\{\psi_1, \psi_2, ..., \psi_n\}$ of $V_N$ we can reformulate equation (12) as

$$\begin{aligned} &\mathbf{Au} = \mathbf{f} \\ &\mathbf{A} = (a(\phi_k, \psi_j))_{j=1,...,m;k=1,...,n} \in \mathbb{R}^{m,n} \\ &\mathbf{u} = (u_j)_{j=1,..,n} \in \mathbb{R}^n \\ &\mathbf{f} = (f(\psi_j))_{j=1,..,m} \in \mathbb{R}^m \end{aligned} \tag{13}$$

**A** is called *Galerkin Matrix*, **f** *load vector* and **u** is the vector with the coefficients of $u_N$ relatively to the basis $\{\varphi_1, \varphi_2, ..., \varphi_m\}$. In a concrete implementation of the FEM, this system would be solved numerically for a suitable basis which produces, in the best case, a very sparse matrix.
If one substitutes $u_N$ in (12) by $u - u_N$, where $u$ is solution of the variational problem (3), one gets (by using linearity of $a$ in the first argument):

$$a(u - u_N, v_N) = a(u, v_N) - a(u_N, v_N) = f(v_N) - f(v_N) = 0 \; \forall \, v_N \in V_N \tag{14}$$

This property is called *Galerkin orthogonality*. By further transformation one derives a first estimate of the error $u - u_N$ in energy norm. More precisely we get that the solution $u_N$ of the discrete variational problem is the best possible approximation of $u$ under all $w_N \in U_N$:

$$\begin{aligned} \|u - u_N\|_a^2 &= a(u - u_N, u - u_N) \\ &= a(u - u_N, u - w_N) + a(u - u_N, w_N - u_N) \\ &= a(u - u_N, u - w_N) \\ &\leq \|u - u_N\|_a \|u - w_N\|_a \; \forall w_N \in U_N \end{aligned} \tag{15}$$

where in the second equality, we used antilinearity in the second argument of $a$. The third equality uses Galerkin orthogonality (14) and the inequality we get by applying Cauchy-Schwarz. After reducing $\|u - u_N\|_a$ on both sides and taking the infimum over all $w_N \in U_N$, we get the desired estimate, called

**Lemma 1** *(Céas Lemma)*

$$\|u - u_N\|_a \leq \inf_{w_N \in U_N} \|u - w_N\|_a \tag{16}$$

This valuation builds the basis for most of the a posteriori error estimates, which we will derive later on.

## 2.4 Finite element space

In this section we will specify how the finite dimensional spaces $U_N$ and $V_N$ from the last section look like in practice and what bases would be the best choice.

The basic idea is to partition our domain $\Omega$ into a *mesh* and then defining the *finite element space* $\mathcal{S}_l^{\mathbf{P}}(\mathcal{M})$on it.

A *mesh* $\mathcal{M}$ of $\Omega \subset \mathbb{R}^d$ is a finite collection $K_{i\,i=1}^{M}$ with $M \in \mathbb{N}$ of open, non-degenerate intervals ($d = 1$) or polygons ($d = 2$) for which the following conditions hold

$$\begin{aligned} &i)\, \bar{\Omega} = \bigcup_{i=1}^{M} \bar{K}_i \\ &ii)\, K_i \cap K_j = \emptyset \Leftrightarrow i \neq j \\ &iii)\, \forall i,\, j \in \{1, ..., M\},\, i \neq j : \bar{K}_i \cap \bar{K}_j = \begin{cases} \emptyset \\ v \in \mathcal{V} \\ e \in \mathcal{E} \end{cases} \end{aligned} \tag{17}$$

where $\mathcal{V}$ is the set of all vertices and $\mathcal{E}$ the set of all edges of $\mathcal{M}$.

$$\mathcal{S}_l^{\mathbf{p}}(\mathcal{M}) := \{v \in C^l(\bar{\Omega}) :\, v_{|K_i} \in \mathcal{W}_{p_i}(K_i)\, \forall K_i \in \mathcal{M}\} \tag{18}$$

for $\mathcal{W}_k = \mathcal{P}_k,\ \mathcal{Q}_k$. The spaces $\mathcal{P}_k$ and $\mathcal{Q}_k$ are named space of *multivariate* respectively of *tensor product* polynomials of degree $k \in \mathbb{N}$ and defined in the following way:

$$\begin{aligned} \mathcal{P}_k(\mathbb{R}^d) &= \{\mathbf{x} \in \mathbb{R}^d \mapsto \sum_{\substack{\alpha \in \mathbb{N}_0^d \\ |\alpha| \leq p}} \kappa_\alpha \mathbf{x}^\alpha, \kappa_\alpha \in \mathbb{R}\} \\ \mathcal{Q}_k(\mathbb{R}^d) &= \{\mathbf{x} \in \mathbb{R}^d \mapsto \sum_{\substack{\alpha \in \mathbb{N}_0^d \\ |\alpha_i| \leq p \forall i \in \{1,2,..,d\}}} \kappa_\alpha \mathbf{x}^\alpha, \kappa_\alpha \in \mathbb{R}\} \end{aligned} \tag{19}$$

One can show that the following compatibility condition holds: Let $\Omega,\, \Omega_1, \Omega_2 \subset \mathbb{R}^d$ bounded Lipschitz domains with $\bar{\Omega} = \bar{\Omega_1} \cup \bar{\Omega_2},\ \Omega_1 \cap \Omega_2 = \emptyset$. Then for $u \in L^2(\Omega),\, u_{|\Omega_i} \in C^1(\bar{\Omega_i}),\, i = 1, 2$ it holds:

$$u \in H^1(\Omega) \Leftrightarrow u \in C(\bar{\Omega}) \tag{20}$$

Thus we get that $\mathcal{S}_0^{\mathbf{p}}(\mathcal{M}) \subset H^1(\Omega)$. If we take $V_N = U_N = \mathcal{S}_0^{\mathbf{p}}(\mathcal{M})$ (in notation from section 2.3) we get by $H^1(\Omega)$-ellipticity of $a$ that $\mathcal{S}_0^{\mathbf{p}}(\mathcal{M})$ is a valid space for the Galerkin discretization.

## 2.5 A priori estimates

In this section we shortly give an idea how the error and the FEM behaves asymptotically in the ideal case. To do this we are going to list some a priori estimates without giving any proofs, since we need it just to relate our numerical results with the theoretically best convergence behaviour. From this, we want to get an idea when to choose $h$- and when $p$-refinement. Further we will restrict ourselves to the one dimensional case, since all our computations will take place in 1D.

**Theorem 1** *Let $\Omega = (a,b) \subset \mathbb{R}$ be an interval and $\mathcal{M}$ be any mesh in $\Omega$. Assume that $u \in H^1(\Omega)$ satisfies*

$$u' \in H^{k_i}(K_i) \quad \forall i \in \{1,...,M\},\ k_i \geq 1. \tag{21}$$

*Then there is a function $s \in \mathcal{S}_1^{\mathbf{p}}(\mathcal{M})$ such that*

$$\|u' - s'\|^2_{L^2(\Omega)} \leq \sum_{i=1}^{M} \left(\frac{h_i}{2}\right)^{2s_i} \frac{(p_i - s_i)!}{(p_i + s_i)!} \|u'\|^2_{H_i^s(K_i)} \tag{22}$$

$$\|u - s\|^2_{L^2(\Omega)} \leq \sum_{i=1}^{M} \left(\frac{h_i}{2}\right)^{2t_i+2} \frac{(p_i - t_i)!}{p_i(p_i+1)(p_i + t_i)!} \|u'\|^2_{H_i^t(K_i)} \tag{23}$$

*where $0 \leq s_i, t_i \leq \min(p_i, k_i)$. (c.f. [8, p.76])*

Using Céas Lemma (16) and the continuity i.e. boundedness of the factors in (1) we get that the estimate (22) of the theorem above also bounds the error of the Galerkin solution in energy norm.
Taking $N = \dim \mathcal{S}_0^{\mathbf{p}}(\mathcal{M}) \sim \frac{p}{h}$ as a work measure, we get from the above theorem that for pure $h$-refinement for $u \in H^k(\Omega)$ and $h = \max_{i\in\{1,..,M\}} h_i$ and $p = \min_{i\in\{1,..,M\}} p_i$ we have:

$$\|u - u_N\|_a \leq C N^{-\min(p,k)} \tag{24}$$

and for pure p-refinement we get:

$$\|u - u_N\|_a \leq C N^{-k} \tag{25}$$

with $C$ independent of $h$ and $p$, but depending on $k$.
Moreover, for an analytic function $u$ in $\bar{\Omega}$ we get that

$$\|u - u_N\|_a \leq C\, \mathrm{e}^{-\kappa N}, \tag{26}$$

$C$ independent of $p$, by local analytical extension of $u$ into the power series on an ellipsis centered around every element $K_i$ (for more details see [6]).
For singular functions $u$ we don't get such a strong convergence, but combining $h$- and $p$- refinement usefully, one could attain the following convergence rate:

$$\|u - u_N\|_a \leq C\, \mathrm{e}^{-\kappa\sqrt{N}} \tag{27}$$

with $C$, $\kappa$ independent of $h$ and $p$ (for further information c.f. [8, 3.3.6, Thm. 3.36]).
From this result it follows directly that we get convergence for our FEM algorithm if we either refine the mesh or ingrease the polynomial degree. Additionally, one might recognize that for locally smooth functions (i.e. for functions $u \in H^{k_i}$ with $k_i$ big), it is more efficient to increase the polynomial degree $p_i$ than refining $h_i$.

# 3 Basic adaptive algorithm for FEM

From here on, we will restrict ourselves to the one dimensional case of (2) and (3) with homogeneous Dirichlet boundary conditions and $\mathbf{b}, c \equiv 0$:
Find $u \in H_0^1((a,b))$, for $a \leq b$, $a, b \in \mathbb{R}$ and $C$ piecewise constant

$$\int_a^b C(x)u'(x)v'(x)\,dx = \int_a^b f(x)v(x)dx \quad \forall\, v \in H_0^1((a,b)). \tag{28}$$

Now let us introduce the basic algorithm on which all the adaptive refinement strategies in FEM are based on.

**1. *Initialization*:** We begin with a coarse uniform decomposition of $\Omega$ into Elements $K_1, ..., K_M$ represented by the vectors $\mathbf{h}$ and $\mathbf{p}$, both of size $M$, containing the length of the intervalls $K_i$ and the local polynomial degree on $K_i$ for $i \in \{1, ..., M\}$. We call this decomposition $\mathcal{K}^{(0)}$ and set $j = 0$.

**2. *Galerkin solution*:** On the current grid $\mathcal{K}^{(j)}$, $j \geq 0$, we solve the linear system resulting from the discretized problem.

**3. *Error estimation*:** Based on the Galerkin solution we calculate an a posteriori error estimate $\epsilon$ and store the local errors in the vector $\epsilon$. If it holds that $\epsilon < tol$ for some tolerance $tol$ we are finished. Otherwise we continue with

**4. *Marking elements*:** Now we choose the elements needing to be refined using our local error estimates in $\epsilon$.
The most common ways to decide whether an element has to be refined or not are the *maximum strategy* and the *fixed energy fraction*. In fact we initialize a vector **mark** of length $M$ with

$$mark_i = \begin{cases} 1\ K_i \text{ has to be refined} \\ 0\ K_i \text{ has not to be refined} \end{cases}$$

**5. *Refinement of marked elements*:** Depending on our refinement strategy:

*Pure h-refinement*: We refine all marked elements $K_n$ by splitting $h_n$ into two parts $\delta h_n$ and $(1-\delta)h_n$ (usually $\delta = \frac{1}{2}$), i.e. changing the $n$-th entry of $\mathbf{h}$ into $\delta h_n$ and enlarging the length of $\mathbf{h}$ and $\mathbf{p}$ by one element, placed immediately after the $n$th entry and containing $(1-\delta)h_n$ respectively $p_n$.

*pure p-refinement*: We refine all marked elements $K_n$ by enlarging the $n$-th entry of $\mathbf{p}$ by $s$ (usually, $s = 1$ is chosen).

*automatic hp-refinement*: We need a criterion which decides automatically between $h$-refinement and p-refinement on the marked element $K_n$ in such a way that the desirable convergence mentioned in (26) resp. (27) is to expect. (See section 5 for details)
Finally we get a new refined decomposition $\mathcal{K}^{(j+1)}$, set $j = j + 1$ and continue with step 3.

Until now, the steps 1. and 2. have been discussed. The next section will focus on step 3, which is the basis and the most crucial step of adaptive $h$-, $p$- and $hp$-refinement. Step 4 and 5 will be discussed later on in section 5.

# 4 A posteriori estimates

In this section we want to focus on the crucial step of the adaptive algorithm, the a posteriori error estimates.
We are going to look at two different ways to estimate the energy error of the Galerkin solution, the first way derives an residual error estimator which can be directly computed after having derived the Galerkin solution $u_N$. The other one is an equilibrating method, this means that an equilibrated local BVP is solved to get the estimate.
For both estimators, we are going to show reliability, efficiency and locality. For us, locality is important since otherwise we are not able to do adaptive refinement.

An error estimator $\epsilon$ is *reliable* if

$$\exists\, C_{rel}: \quad \|u - u_N\|_a \leq C_{rel}\epsilon + h.o.t. \tag{29}$$

An error estimator $\epsilon$ is *efficient* if

$$\exists\, C_{eff}: \quad \epsilon \leq C_{eff}\|u - u_N\|_a + h.o.t. \tag{30}$$

where $h.o.t$ contains all functions converging in an higher order than $\|u-u_N\|_a$, i.e. converging $o(\|u-u_N\|_a)$ and $C_{eff}$, $C_{rel}$ only depend on problem and mesh parameters. (c.f. [9, p. 16, def. 1.2])

Moreover we derive more information about the dependencies of the constants in (29) and (30) to get a better understanding of the convergence behaviour.
Satisfying these two properties guarantees that the estimator bounds the error (reliability), behave in a similar way as the exact error does (efficiency and reliability) and gives a bound for the overestimation (efficiency).

## 4.1 Residual error estimation

In this subsection we fully discuss the properties of the following residual error estimator (c.f. [3, p. 1111, 1112]) :

$$\begin{aligned}
\epsilon_{K_i}^2 &= \nu_{K_i}^2 + \delta_{K_i}^2 && (31)\\
&= \frac{1}{C_i p_i(p_i+1)}\|\mathrm{res}_{K_i}\|_{L^2(K_i)}^2 \\
&\quad + \frac{h_i^2}{4C_i p_i^2}\|f - \Pi_{\mathcal{M}}^{\mathbf{q}} f\|_{L^2(K_i)}^2 \ \forall i \in \{1,2,...,M\} && (32)\\
\epsilon^2 &= \sum_{i=1}^{M} \epsilon_{K_i}^2 && (33)
\end{aligned}$$

where

$$\mathrm{res}_{K_i} := \Pi_{\mathcal{M}}^{\mathbf{p}} f + C_i u_N''. \tag{34}$$

### 4.1.1 Reliability of the residual error estimate

We first want to estimate the following term for arbitrary $v \in H_0^1(\Omega)$ and $v_N \in \mathcal{S}_0^{\mathbf{P}}(\mathcal{M})$.

$$\begin{aligned} a(u-u_N, v) &= a(u-u_N, v-v_N) && (35)\\ &= f(v-v_N) - a(u_N, v-v_N) && (36)\\ &= \int_\Omega (f + (Cu_N')')(v-v_N)\,dx + && (37)\\ &\qquad \sum_{z\in\mathcal{V}} (C(z^+)u_N'(z^+) - C(z^-)u_N'(z^-))(v - v_N(z)) \\ &= \int_\Omega (f + (Cu_N')')(v-v_N)\,dx && (38) \end{aligned}$$

in (36) Galerkin orthogonality is used and in (38) we exploit that

$$v(z) = v_N(z)\,\forall z \in \mathcal{V}, \tag{39}$$

if we choose $v_N$ in the following way:

$$v_N|_{K_j}(x) := \int_{\phi_j^{-1}(-1)}^{x} s_j(\nu)\,d\nu + v(\phi_j^{-1}(-1)) \tag{40}$$

where

$$s_j(\nu) = \sum_{i=0}^{p_j-1} a_i^j \sqrt{\frac{2}{h_j}} L_i(\phi_i(\nu)) \tag{41}$$

$$a_i^j = \frac{2i+1}{2} \int_{K_j} v'(\xi) \sqrt{\frac{2}{h_j}} L_i(\phi_j(\xi))\,d\xi, \tag{42}$$

which are well defined, using the completeness of $\{L_i(x)\}_{i=0}^\infty$ in $L^2((-1,1))$ respectively $\{\sqrt{2/h_j}L_i(\phi_j(x))\}_{i=0}^\infty$ in $L^2(K_j)$. Then by definition

$$v_N(\phi_j^{-1}(-1)) = v(\phi_j^{-1}(-1))\,\forall j \in \{1,...,M\} \tag{43}$$

and further $\forall j \in \{1,...,M\}$

$$v(\phi_j^{-1}(1)) - v(\phi_j^{-1}(-1)) = \int_{K_j} v'(\nu)\,d\nu = 2a_0^j \tag{44}$$

$$v_N(\phi_j^{-1}(1)) - v_N(\phi_j^{-1}(-1)) = \int_{K_j} v_N'(\nu)\,d\nu = 2a_0^j \tag{45}$$

by the orthogonality properties of the Legendre polynomials. And therefore we get the wanted property (39) of $v_N$.

In the next part we continue with the derivation of our residual error esti-

mate.

$$a(u-u_N,v) = \sum_{i=1}^{M} \int_{K_i} (f + C_i u_N'')(v - v_N)\,dx \tag{46}$$

$$= \sum_{i=1}^{M} \int_{K_i} (\Pi_{\mathcal{M}}^{\mathbf{q}} f + C_i u_N'' + f - \Pi_{\mathcal{M}}^{\mathbf{q}} f)(v - v_N)\,dx \tag{47}$$

$$= \sum_{i=1}^{M} \int_{K_i} \mathrm{res}_{K_i}(v - v_N)\,dx + \int_{K_i} (f - \Pi_{\mathcal{M}}^{\mathbf{q}} f)(v - v_N)\,dx \tag{48}$$

where $\Pi_{\mathcal{M}}^{\mathbf{q}} f$ is an arbitrary projection (for example the projection chosen in (165)) of $f$ onto $\mathcal{S}_0^{\mathbf{q}}(\mathcal{M})$ with $q_i \geq p_i + 1 \, \forall i \in \{1, ..., M\}$. We continue by the estimation of the first term. Using the bubble function

$$\omega_{K_i} := (\phi_i^{-1}(1) - x)(x - \phi_i^{-1}(-1)) \tag{49}$$

we get

$$|\int_{K_i} \mathrm{res}_{K_i}(x)(v - v_N)(x)\,dx| = |\int_{K_i} \sqrt{\omega_{K_i}(x)}\,\mathrm{res}_{K_i}(x) \frac{(v - v_N)(x)}{\sqrt{\omega_{K_i}(x)}}\,dx| \tag{50}$$

$$\leq \Big(\int_{K_i} \omega_{K_i}(x) |\,\mathrm{res}_{K_i}(x)|^2\,dx \int_{K_i} \frac{|(v - v_N)(x)|^2}{\omega_{K_i}(x)}\,dx\Big)^{1/2} \tag{51}$$

$$\leq \Big(\int_{K_i} \omega_{K_i}(x) |\,\mathrm{res}_{K_i}(x)|^2\,dx\Big)^{1/2} \frac{1}{\sqrt{C_i p_i (p_i+1)}} \| \sqrt{C_i}(v' - v_N')\|_{L^2(K_i)} \tag{52}$$

the last inequality is obtained using (40) and using that for $x \in K_i$ it holds:

$$(v - v_N)(x) = \int_{\phi_i^{-1}(-1)}^{x} \sum_{j=p_i}^{\infty} a_j^i \sqrt{\frac{2}{h_i}} L_j(\phi_i(t))\,dt \tag{53}$$

$$= \sum_{j=p_i}^{\infty} a_j^i \int_{\phi_i^{-1}(-1)}^{x} \sqrt{\frac{2}{h_i}} L_j(\phi_i(t))\,dt. \tag{54}$$

Further using (174) we get

$$\int_{\phi_i^{-1}(-1)}^{x} \sqrt{\frac{2}{h_i}} L_j(\phi_i(t))\,dt = \frac{\sqrt{h_i}}{\sqrt{2} j(j+1)} (\phi_i(x)^2 - 1) L_j'(\phi_i(x)), \tag{55}$$

and inserting this above:

$$\int_{K_i} \frac{|(v-v_N)(x)|^2}{\omega_{K_i}(x)}\,dx = \int_{K_i} \frac{\left(\sum_{j=p_i}^{\infty} \frac{a_j^i\sqrt{h_i}}{\sqrt{2}j(j+1)}(\phi_i(x)^2-1)L_j'(\phi_i(x))\right)^2}{(\phi_i^{-1}(1)-x)(x-\phi_i^{-1}(-1))}\,dx \tag{56}$$

$$= \sum_{j=p_i}^{\infty} \frac{(a_j^i)^2 h_i}{2j^2(j+1)^2} \int_{K_i} \frac{-4}{h_i^2}(\phi_i(x)^2-1)L_j'(\phi_i(x))^2\,dx \tag{57}$$

$$= \sum_{j=p_i}^{\infty} \frac{2(a_j^i)^2}{h_i j^2(j+1)^2} \int_{-1}^{1} (1-x^2)L_j'(x)^2\,dx \tag{58}$$

$$= \sum_{j=p_i}^{\infty} \frac{j(j+1)}{j^2(j+1)^2} \frac{2(a_j^i)^2}{(2j+1)} \tag{59}$$

$$\leq \frac{1}{C_i p_i(p_i+1)} \|\sqrt{C_i}(v'-v_N')\|_{L^2(K_i)}^2. \tag{60}$$

In the last inequality we make use of (171) and in (57) we use the definition of our transformation $\phi_i(x)$ (see (154)), to get that

$$\phi_i(x) - 1 = -2 + 2\frac{x - a - \sum_{l=1}^{i-1} h_l}{h_i} \tag{61}$$

$$= -\frac{2}{h_i}\left(a + \sum_{l=1}^{i} h_l - x\right) \tag{62}$$

$$= -\frac{2}{h_i}\left(\phi_i^{-1}(1) - x\right) \tag{63}$$

and analogously $\phi_i(x) + 1 = \frac{2}{h_i}\left(x - \phi_i^{-1}(-1)\right)$.
Moreover in (58) we exploit that

$$\int_{-1}^{1} (1-x^2)L_i'(x)L_j'(x)\,dx = \delta_{ij}\frac{2i(i+1)}{2i+1}. \tag{64}$$

With this, we arrive at equation (52).

Furthermore

$$\|\sqrt{C_i}(v'-v_N')\|_{L^2(K_i)} \leq \|C_i \sum_{j=p_i}^{\infty} a_j^i \sqrt{\frac{2}{h_i}} L_j(\phi_i(x))\|_{L^2(K_i)} \tag{65}$$

$$\leq \|\sqrt{C_i}v'\|_{L^2(K_i)}. \tag{66}$$

The inequalities hold by definition of $v_N$ as an integral over the truncated Legendre series of $v'$.

Continuing from (48) and repeating the same procedure as for the first term we

get

$$a(u-u_N,v) = \sum_{i=1}^{M} \int_{K_i} \mathrm{res}_{K_i}(v-v_N)\,dx + \int_{K_i} (f-\Pi^{\mathfrak{q}}_{\mathcal{M}} f)(v-v_N)\,dx \tag{67}$$

$$\leq \sum_{i=1}^{M} \big( \|\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}\,\|_{L^2(K_i)} + \|\sqrt{\omega_{K_i}}(f-\Pi^{\mathfrak{q}}_{\mathcal{M}} f)\|_{L^2(K_i)} \big) \cdot \frac{1}{\sqrt{C_i p_i (p_i+1)}} \|\sqrt{C_i} v'\|_{L^2(K_i)} \tag{68}$$

and therefore applying (68) for $v = u - u_N$ and using that $\omega_{K_i}(x) \leq h_i^2/4 \; \forall x \in K_i$ we get that the demanded estimator (32) bounds the error in energy norm:

$$\|u-u_N\|_a^2 \leq \sum_{i=1}^{M} ((\nu_{K_i} + \delta_{K_i}) \|u-u_N\|_{a,K_i}) \tag{69}$$

$$\leq \sqrt{\sum_{i=1}^{M} (\nu_{K_i}^2 + \delta_{K_i}^2)}\, \|u-u_N\|_a \tag{70}$$

where the last inequality follows by applying Cauchy-Schwarz for sums and shows reliability of our estimator $\epsilon$ with $C_{rel} = 1$.

### 4.1.2 Efficiency of the residual error estimator

To get our error estimator bounded from above by the energy error, we extend the the proof written down breathly in [3, p. 1112]. First we use that equations (48) hold for an arbitrary $v \in H_0^1(\Omega)$ and $v_N \in \mathcal{S}_1^{\mathfrak{P}}(\mathcal{M})$, such that (39) holds. For every $i \in \{1, 2, ..., M\}$ we set

$$v := \begin{cases} \omega_{K_i}\,\mathrm{res}_{K_i} & x \in K_i \\ 0 & \text{otherwise} \end{cases} \tag{71}$$

$$v_N :\equiv 0 \tag{72}$$

with $\omega_{K_i}$ defined as in (49). $v$ lies for sure in $H_0^1(\Omega)$ since it is piecewise smooth.

We get the following result:

$$\int_{K_i} \omega_{K_i} (\mathrm{res}_{K_i})^2\,dx = \int_{K_i} \big\{ C_i (u-u_N)'(\omega_{K_i}\,\mathrm{res}_{K_i})' - (f-\Pi^{\mathfrak{q}}_{\mathcal{M}} f)\omega_{K_i}\,\mathrm{res}_{K_i} \big\}\,dx \tag{73}$$

$$\leq \|u-u_N\|_{a,K_i} \|(\omega_{K_i}\,\mathrm{res}_{K_i})'\|_{L^2(K_i)} + \|\sqrt{\omega_{K_i}}(f-\Pi^{\mathfrak{q}}_{\mathcal{M}} f)\|_{L^2(K_i)} \|\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}\,\|_{L^2(K_i)} \tag{74}$$

using Cauchy-Schwarz inequality and that $C_i$ are constants which can be switched in and out of the integral without changing anything. Now we introduce an inverse estimate for $w \in \mathcal{P}_q$ for some $q \in \mathbb{N}$:

$$\|(\omega_{K_i} w)'\|_{L^2(K_i)} \leq 4(q+1) \|\sqrt{\omega_{K_i}} w\|_{L^2(K_i)} \tag{75}$$

which can be shown in a similar way as in $(56)-(60)$ (c.f. [3, p.1110]). Applying this estimate to $\mathrm{res}_{K_i} \in \mathcal{P}_{q_i}$ in (74), we get

$$\|\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}\,\|_{L^2(K_i)} \leq \left(4(q_i+1)\|u-u_N\|_{a,K_i} + \|\sqrt{\omega_{K_i}}(f-\Pi^{\mathbf{q}}_{\mathcal{M}}f)\|_{L^2(K_i)}\right) \tag{76}$$

$$\leq 4(\frac{q_i}{p_i}+1)p_i\left(\|u-u_N\|_{a,K_i} + \frac{h_i}{2p_i}\|(f-\Pi^{\mathbf{q}}_{\mathcal{M}}f)\|_{L^2(K_i)}\right) \tag{77}$$

exploiting that the bubble function satisfies $\omega_{K_i}(x) \leq h_i^2/4\ \forall x \in K_i$, $\frac{1}{p_i} \leq 1$ and $\frac{1}{q_i+1} \leq \frac{1}{p_i}$ after definition of $q_i$ (see paragraph following after (48)). By using that for arbitrary values $x, y \in \mathbb{R}$ it holds

$$x + y = \sqrt{x^2+2xy+y^2} \leq \sqrt{4x^2+4y^2} = 2\sqrt{x^2+y^2} \tag{78}$$

we arrive at our goal:

$$\|\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}\,\|_{L^2(K_i)} \leq 8(\frac{q_i}{p_i}+1)p_i\sqrt{\|u-u_N\|^2_{a,K_i} + \frac{h_i^2}{4p_i^2}\|(f-\Pi^{\mathbf{q}}_{\mathcal{M}}f)\|^2_{L^2(K_i)}} \tag{79}$$

$$= C_{eff}\sqrt{\|u-u_N\|^2_{a,K_i} + \delta^2_{K_i}} \tag{80}$$

which shows efficiency of the residual error estimator with constant

$$C_{eff} = 8 \cdot \max_{i\in\{1,\dots,M\}}\left\{\frac{q_i}{p_i}\right\} + 1$$

**Remark:** It is assumed that the oscillation term $\delta_{K_i}$ converges higher order than the error of the Galerkin discretization in energy norm (data saturation assumption). Computationally we enforce this by using the implicit interpolation of a high order quadrature for the integration or taking the $L^2$-projection onto the polynomial space $\mathcal{S}_0^{\mathbf{q}}(\mathcal{M})$, with $\mathbf{q}$ choosen big enough.

## 4.2 Equilibrated error estimator

To get an estimation of the error in energy norm, this estimator, described in [2], is a linear combination of the solutions of the new local boundary value problems containing projected and equilibrated terms:

For $i \in \{1, 2, ..., M-1\}$ find $\sigma_i \in RT^{p+1}_{-1}(\mathrm{supp}(\Psi_i))$:

$$\sigma_i' = r_i^K \quad \text{in } K \in \mathrm{supp}(\Psi_i) \tag{81}$$

$$\sigma_i^-(x_i) - \sigma_i^+(x_i) = (r_i^{x_i})^- - (r_i^{x_i})^+ \tag{82}$$

$$\sigma_i(x_{i-1}) = \sigma_{\omega_i}(x_{i+1}) = 0 \tag{83}$$

with

$$r_i^K := \Psi_i\left(\Pi(f) + Cu_N''\right) \tag{84}$$

$$(r_i^{x_i})^- - (r_i^{x_i})^+ := C(x_i)^-(u_N')^-(x_i) - C(x_i)^+(u_N')^+(x_i) \tag{85}$$

where $\Psi_i$ is the hat basis function associated with the vertex $\phi_i^{-1}(1) =: x_i$ ($x_0 := \phi_1^{-1}(-1)$), $\text{supp}(\Psi_i) = K_i \cup K_{i+1}$, $\Pi(f)$ is the $L^2$-orthogonal projection of $f$ onto $\mathcal{S}_0^{\mathbf{p}}(\mathcal{M})$ and

$$RT_{-1}^p(D) := \{\tau \in L^2(D) : \tau|_K \in \mathcal{P}_{p+1}, T \subset D, T \in \mathcal{M}\} \tag{86}$$

is the *broken Raviart-Thomas space* defined on a domain $D$.
Outside of $\text{supp}(\Psi_i)$ we extend $\sigma_i$ by the zero-function.

The error estimator coming out of this local problem looks as follows:

$$\epsilon_{hyp}^2 := \sum_{i=1}^{M} \epsilon_{hyp,K_i}^2 \tag{87}$$

$$\epsilon_{hyp,K_i}^2 := \|\frac{1}{\sqrt{C_i}}\sigma^{\Delta}\|_{L^2(K_i)}^2 + \frac{h_i^2}{4C_i p_i^2}\|f - \Pi(f)\|_{L^2(K_i)}^2 \tag{88}$$

$$\sigma^{\Delta} := \sum_{i=1}^{M-1} \sigma_i \tag{89}$$

We directly begin with showing reliability and efficiency. For the proof of efficiency and p-robustness, we are just going to give an idea of the construction, since the whole proof is quite long. We refer to the results in the paper of Braess [2].
P-robustness is a rare but very desirable property of error estimators, since from this we get an estimator for the error without any p-dependent constant. This means that we do not have a loss of efficiency for large p.

### 4.2.1 Reliability of the equilibrated error estimator

Reliability follows from a famous Theorem of Prager and Synge, called *hypercircle method* (c.f. [1, p. 141]).

**Theorem 2** *Let $v \in H_0^1(\Omega)$ for $\Omega = (a, b)$ satisfying $\sigma' + f = 0$.*
*Then it holds for the solution $u$ of* (28)*:*

$$\|u - v\|_a^2 + \|\frac{1}{\sqrt{C}}(Cu' - \sigma)\|_{L^2(\Omega)}^2 = \|\sqrt{C}(v' - \frac{1}{C}\sigma)\|_{L^2(\Omega)}^2. \tag{90}$$

**Proof:**

$$\|\frac{1}{\sqrt{C}}(Cv'-\sigma)\|_{L^2(\Omega)} = \int_\Omega \frac{1}{C}(Cv'-\sigma)^2\,dx \tag{91}$$

$$= \int_\Omega (v'-\frac{1}{C}\sigma)(Cv'-\sigma)\,dx \tag{92}$$

$$= \int_\Omega C(v'-\frac{1}{C}\sigma)^2\,dx \tag{93}$$

$$= \int_\Omega C(v'-u')(v'-\frac{1}{C}\sigma)\,dx + \int_\Omega C(u'-\frac{1}{C}\sigma)(v'-\frac{1}{C}\sigma)\,dx \tag{94}$$

$$= \|v-u\|_a^2 + 2\int_\Omega C(v'-u')(u'-\frac{1}{C}\sigma)\,dx + \|\sqrt{C}(u'-\frac{1}{C}\sigma)\|^2_{L^2(\Omega)} \tag{95}$$

where, using that $C$ is piecewise constant and applying integration by parts we get for the integral in (95):

$$\int_\Omega C(v'-u')(u'-\frac{1}{C}\sigma)\,dx = -\int_\Omega (v-u)(Cu''-\sigma')\,dx + (u(x)-v(x))(C(x)u'(x)-\sigma(x))|_{x=a}^b \tag{96}$$

$$= 0 \tag{97}$$

using that $Cu'' = -f = \sigma'$ and $v(a) = v(b) = u(a) = u(b) = 0$ by definition. Thus the theorem is proven (c.f. [1].

**Remark:** We notice that (90) also holds locally on the elements $K_j$, setting $v = u_N$ and using what is stated in (116).

Using that for our $\sigma^\Delta$ it holds by definition

$$(\sigma^\Delta)' = \sum_{i=1}^{M-1} \sigma_i' = \sum_{i=1}^{M-1} \Psi_i\,(\Pi(f) + Cu_N'') = \Pi(f) + Cu_N''. \tag{98}$$

If we use the hypercircle method with $\sigma := \sigma^\Delta - Cu_N'$, $f := \Pi(f)$ and $v := u_N$, we get

$$\|\tilde{u} - u_N\|_a \le \|\sqrt{C}(u_N' - \frac{1}{C}(\sigma^\Delta - Cu_N'))\|_{L^2(\Omega)} = \|\frac{1}{\sqrt{C}}\sigma^\Delta\|_{L^2(\Omega)} \tag{99}$$

where $\tilde{u}$ is the solution of (28) with right hand side $\Pi(f)$.

Applying that the inequality holds also locally we substitute $\Omega$ by $K_i$. Further using the results written down in (68) and derived there before by analyzing the residual error estimator

$$\|\sqrt{C_i}(u-\tilde{u})\|^2_{L^2(K_i)} \le \frac{h_i}{2\sqrt{C_i}p_i}\|f-\Pi(f)\|_{L^2(K_i)}\|\sqrt{C_i}(u-\tilde{u})\|_{L^2(K_i)}, \tag{100}$$

we arrive at our aim that the estimator is reliable with constant $C_{rel} = 1$:

$$\|u - u_N\|_a^2 = \|u - \tilde{u}\|_a^2 + \|\tilde{u} - u_N\|_a^2 \tag{101}$$

$$\leq \sum_{i=1}^{M} \left( \frac{h_i^2}{4C_i p_i^2} \|f - \Pi(f)\|_{L^2(K_i)}^2 + \|\frac{1}{\sqrt{C_i}} \sigma^{\Delta}\|_{L^2(K_i)} \right) \tag{102}$$

$$= \epsilon_{hyp}^2. \tag{103}$$

This error estimator makes use of the fact that we just need to solve the mixed problem above with *a polynomial right hand side* as derived by analyzing the residual error estimator. It makes life much easier, since we now can solve the problem pointwise on every element $K$ by just integrating over $r_{\omega_i}^K \in \mathcal{P}_{p+1}$ (this integration is exact for quadrature of order p+1), thus the result lies in $RT_{-1}^{p+1}(\text{supp}(\Psi_i))$. By fitting the constants (c.f. implementation found in 8.3.2) we get our $\sigma_i$. Existence of the $\sigma_i$ is then given by the fact, that

$$< r_i, 1 >= 0 < r_i, v > := a(\tilde{u} - u_N, \Psi_i v) \tag{104}$$

$$= \int_{\text{supp}(\Psi_i)} \Pi(f) \Psi_i v \, dx - a(\Psi_i u_N, v) \tag{105}$$

$$= \sum_{j=i}^{i+1} \left( \int_{K_j} r_i^{K_j} v \, dx \right) + (r_i^{x_i})^- v(x_i)^- - (r_i^{x_i})^+ v(x_i)^+ \tag{106}$$

using Galerkin orthogonality and that the constant function $1 \in \mathcal{S}_0^{\mathbf{p}}(\mathcal{M})$ (c.f. [2]).

### 4.2.2 Efficiency of the equilibrated error estimator

As already stated above, only an idea of the proof is given here. For more details, see [2], where the proof is explained in detail for the two-dimensional case.

We first have to show that the dual norm of the local residuals

$$\|r_i\|_* := \sup_{v \in H^1(\text{supp}(\Psi_i))/\mathbb{R}} \frac{< r, v >}{\|v'\|_{L^2(\text{supp}(\Psi_i))}} \tag{107}$$

is bounded by the energy norm of the error $\|\tilde{u} - u_N\|_{a,\text{supp}(\Psi_i)}$ with constants independent of $h_i$ and $p_i$, $i \in \{1, 2, ..., M\}$.
After that it is proven, that for our $r_i$ with $r_i^K \in \mathcal{P}p + 1(K)$ we get

$$< r_i, 1 >= 0 \quad \Longrightarrow \quad \inf_{\substack{\sigma \in RT_{-1,0}^{p+1} \\ \sigma' = r_i}} \|\sigma\|_{L^2(\text{supp}(\Psi_i))} \leq C \|r_i\|_* \tag{108}$$

with $C$ independent of $h_i$ and $p_i$, $i \in \{1, 2, ..., M\}$ and

$$RT_{-1,0}^{p+1} := \left\{ v \in RT_{-1}^{p+1} |\ v|_{\partial \,\text{supp}(\Psi_i)} = 0 \right\}. \tag{109}$$

Thus the result follows by first applying (101) and (100) and afterwards combining with the two statements (107), (108).

# 5 Marking and Refinement

## 5.1 Marking elements

In this section we explain in detail what is shortly mentioned in part 4 of our basic algorithm for adaptive FEM. Our aim is to mark those elements of our decomposition $\mathcal{K}^{(j)}$ which promise to decrease our error the most after being refined. The main considerations which give rise to the strategies are quite intuitive (c.f. [3, 3.2, (3)]).
Refinement has to be done on elements with big estimated local error $\epsilon_{K_{i_0}}$ compared to the error $\epsilon_{K_i}$ of the others. The two most common techniques are the *maximum strategy* and the *fixed energy fraction*.

### 5.1.1 Maximum strategy

For $\theta \in (0,1)$ set

$$mark_i = 1 :\Leftrightarrow \epsilon_{K_i} \geq (1-\theta) \max_{i \in \{1,..,M\}} \epsilon_{K_i} \tag{110}$$

### 5.1.2 Fixed energy fraction

For $\theta \in (0,1)$ define **mark** such that

$$\sum_{i=1}^{M} mark_i \epsilon_{K_i}^2 \geq \theta^2 \sum_{i=1}^{M} \epsilon_{K_i}^2 \tag{111}$$

To be efficient, **mark** should be chosen as small as possible fulfilling the property (111).
For hp-refinement, we are going to introduce a *weighted fixed energy fraction*, defined in the following way:
For $\theta \in (0,1)$ and weights $I_i$, $i \in \{1, ..., M\}$ define **mark** such that

$$\sum_{i=1}^{M} mark_i (I_i \epsilon_{K_i})^2 \geq \theta^2 \sum_{i=1}^{M} \epsilon_{K_i}^2 \tag{112}$$

## 5.2 Refinement of marked elements

For pure $h$- and $p$-refinement, the marking strategy is all we need.
But for the mixed $hp$-refinement we have to discuss how to find reasonable automatic decision criteria. Going back to the a priori error estimates written down in 2.5, we conclude that it might be reasonable to search for information about the local analyticity of our solution $u$.
The decision criterion we are going to discuss first does this by expanding $u$ locally into its Legendre series and estimating its Bernstein radius.
Further we are going to discuss another approach which generates a minimization problem out of which we get the marking of elements together with the most efficient refinement pattern.

### 5.2.1 Estimation of analyticity

The idea is to estimate the local regularity on $K_j$ of the exact solution $u$ by calculating the coefficients $b_i^j$ of its Legendre series and estimating the elemental Bernstein radius (The Bernstein radius $\rho_j = \frac{2(a+b)}{h_j}$, where $a$ is the length of the semi-major and $b$ the length of the semi-minor axes, is the radius of the ellipse on whose interior the Legendre series converges absolutely and uniformly on any closed set).
We are able to calculate the first Legendre coefficients $b_i^j$ for $i \in \{0, ..., p_j\}$ of $u$ since they are identical to those of $u_N$. The reason why follows from the fact that for $c(x) \equiv 0$ we have using the Galerkin orthogonality for $v|_{K_j}(x) = x \in \mathcal{S}_0^{\mathbf{p}}(\mathcal{M})$:

$$0 = a(u - u_N, v) \tag{113}$$
$$= \int_{K_j} C_j(u - u_N)'(x)\, dx \tag{114}$$
$$= C_j(u - u_N)(\phi_j^{-1}(1)) - C_j(u - u_N)(\phi_j^{-1}(-1)) \tag{115}$$

using that the homogeneous Dirichlet boundary conditions also hold for the Galerkin discretization $u_N$, we get:
$\forall j \in \{1, ..., M\}$

$$u(\phi_j^{-1}(1)) = u_N(\phi_j^{-1}(1)) \tag{116}$$
$$u(a) = u_N(a) = 0 \tag{117}$$

and therefore we can write $u_N$ in the form (40) since it follows by construction of $\mathcal{S}_0^{\mathbf{p}}(\mathcal{M})$:

$$\int_{K_j} C_j(u - u_N)'(x)v'(x)\, dx = 0 \ \forall v \in \mathcal{S}_0^{\mathbf{p}}(\mathcal{M}) \tag{118}$$
$$\Leftrightarrow \int_{K_j} C_j(u - u_N)'(x)v'(x)\, dx = 0 \ \forall v \in \mathcal{P}_{p_j}(K_j) \tag{119}$$
$$\Leftrightarrow \int_{K_j} C_j(u - u_N)'(x)w(x)\, dx = 0 \ \forall w \in \mathcal{P}_{p_j-1}(K_j) \tag{120}$$
$$\tag{121}$$

which implies

$$\Pi_{K_j}^{p_j-1}(u - u_N)'(x) = \Pi_{K_j}^{p_j-1}u'(x) - u_N'(x) = 0 \ \forall x \in K_j \tag{122}$$

and hence

$$\int_{\phi_j^{-1}(-1)}^{x} \Pi_{K_j}^{p_j-1}u'(\nu)\, d\nu = u_N(x) - u_N(\phi_j^{-1}(-1))$$
$$= u_N(x) - u(\phi_j^{-1}(-1)) \ \forall x \in K_j. \tag{123}$$

The last equality follows by (116).
Now applying integration by parts to (120) we get together with the nodal exactness of $u_N$ that

$$\int_{K_j} C_j(u - u_N)(x)w(x)\, dx = 0 \ \forall w \in \mathcal{P}_{p_j-2}(K_j). \tag{124}$$

Further, making use of the properties of the Legendre polynomials, we get that the first $p_i + 1$ Legendre coefficients of the solution $u$ are identical to those of $u_N$.

Let us now look back at the Bernstein radius, which is obtained by calculating

$$\limsup_{i\to\infty} |b_i^j|^{1/i} = \frac{1}{\rho_j} =: \theta_j, \tag{125}$$

but we have to settle for a computable approximation $\hat{\theta}_j$ of $\theta_j$. We will get $\hat{\theta}_j$ in the following way:
First we take the i-th power of equation (125) and then apply the logarithm:

$$\log |b_i^j| \sim i \log \frac{1}{\rho_j} \quad i \to \infty \tag{126}$$

Now we fit the slope $m_j$ in

$$|log|b_i^j|| = im_j + n_j \tag{127}$$

by linear regression to the already computed $y_i := log|b_i^j|$ for $i \in \{0, ..., p_j\}$ and get:

$$m_j = 6\frac{2\sum_{i=0}^{p_j-2} iy_i - (p_j-2)\sum_{i=0}^{p_j-2} y_i}{(p_j-1)((p_j-1)^2-1)} \tag{128}$$

which can be used for the approximation as:

$$\hat{\theta}_j = e^{-m_j} \tag{129}$$

Our completed algorithm now decides between $h$- and $p$-refinement based on an arbitrarily chooseable parameter $\theta \in (0, 1)$:

$$\begin{cases} p\text{-refinement} & \text{if } \hat{\theta}_j >= \theta \\ h\text{-refinement} & \text{otherwise} \end{cases}. \tag{130}$$

For more details on this decider, we refer to the article [7].

### 5.2.2 Minimization

The idea is to choose the most effective refinement pattern out of patterns $\{1, 2, ..., l\}$ for $l \in \{1, 2, ..., 5\}$ on every element which has to be refined.

1. **bisection:**
   subdivision of $K_j$ into $K'$ and $K''$ where $h' = h'' = \frac{1}{2}h_j$
2. **common p-refinement:**
   the polynomial degree $p_j$ on $K_j$ is increased by 1
3. **graded bisection:**
   subdivision of $K_j$ into $K'$ and $K''$ where $h' = 0.15h_j$ and $h'' = 0.85h_j$

4. **graded bisection:**
   subdivision of $K_j$ into $K'$ and $K''$ where $h' = 0.85h_j$ and $h'' = 0.15h_j$
5. **p-refinement:**
   the polynomial degree $p_j$ on $K_j$ is increased by 2

The is composed of two steps:

- one solves for every pattern $l \in \{1, ..., 5\}$ a local problem to get an index for its efficiency
- by a minimization problem we determine the most effective pattern on each element and which elements need to be refined

In the following we will discuss the two steps in detail:

**Step 1: Getting a local index for efficiency**

First we have to define the index of efficiency $\beta^l_{K_i}$ for every pattern $l \in \{1, 2, ..., 5\}$.

$$\beta^l_{K_i} := \frac{p_i}{\|\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}\|_{L^2(K_i)}} \sup_{\tilde{w}_N \in \tilde{V}^l_{N,K_i}} \left\{ \frac{\int_{K_i} \mathrm{res}_{K_i}\,\tilde{w}_N\,dx}{\|\tilde{w}_N\|_{L^2(K_i)}} \right\} \tag{131}$$

where $\tilde{V}^l_{N,K_i}$ is the new FES of functions with compact support in $K_i$, locally refined by pattern $l$. By few transformations (applying the same property as used in (52) to $\|\tilde{w}'_N\|_{L^2(K_i)}$) we can interpret the index as the angle between span$\{\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}\}$, which represents the span of our energy error indicating function (see (34)), and the new refined space times $\frac{1}{\sqrt{\omega_{K_i}}}$. This is an index of efficience of the refinement in the following way: The bigger the angle the smaller the projection of $\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}$ onto our new space, the smaller we expect the $L^2$-norm of the residual times the bubble function $\sqrt{\omega_{K_i}}$ in our new refined space to be. Moreover the estimate in (32) shows that a minimal norm of this type gives us a minimal bound from below for the energy error of the new Galerkin solution.

To compute $\beta^l_{K_i}$ we take advantage of the fact that the solution $\tilde{z}^l_N \in \tilde{V}^l_{N,K_i}$ of

$$\int_{K_i} (\tilde{z}^l_N)'(x)\tilde{w}'_N(x)\,dx = \int_{K_i} \mathrm{res}_{K_i}(x)\tilde{w}_N(x)\,dx \quad \forall \tilde{w}_N \in \tilde{V}^l_{N,K_i} \tag{132}$$

satisfies

$$\frac{\int_{K_i} \mathrm{res}_{K_i}\,\tilde{w}_N\,dx}{\|\tilde{w}'_N\|_{L^2(K_i)}} = \frac{\int_{K_i} (\tilde{z}^l_N)'\tilde{w}'_N\,dx}{\|\tilde{w}'_N\|_{L^2(K_i)}} \tag{133}$$

$$\leq \frac{\|(\tilde{z}^l_N)'\|_{L^2(K_i)}\|\tilde{w}'_N\|_{L^2(K_i)}}{\|\tilde{w}'_N\|_{L^2(K_i)}} \tag{134}$$

$$= \|(\tilde{z}^l_N)'\|_{L^2(K_i)} \tag{135}$$

$$= \frac{\int_{K_i} \mathrm{res}_{K_i}\,\tilde{z}^l_N\,dx}{\|(\tilde{z}^l_N)'\|_{L^2(K_i)}} \tag{136}$$

and thus by definition

$$\beta_{K_i}^l = \frac{p_i}{\|\sqrt{\omega_{K_i}}\,\mathrm{res}_{K_i}\|_{L^2(K_i)}}\|(\tilde{z}_N^l)'\|_{L^2(K_i)} \tag{137}$$

**Step 2: Minimization problem**

Let $\mathcal{A}^{(j)} \subset \mathcal{K}^{(j)}$ be the set containing elements needing to be refined in the $j^{th}$ refinement step, then the above mentioned minimization problem looks as follows:
Find $\mathcal{A}^{(j)} \subset \mathcal{K}^{(j)}$ and $(l_{K_i})_{i=1}^M$ such that

$$\sum_{i=1}^{M} \frac{\mathrm{dof}_{K_i}^{l_{K_i}}}{\beta_{K_i}^{l_{K_i}}} \longrightarrow \min, \tag{138}$$

under the constraint that the weighted fixed energy fraction holds

$$\sum_{K_i \in \mathcal{A}^{(j)}} \left(\beta_{K_i}^{l_{K_i}} \epsilon_{K_i}\right)^2 \geq \theta^2 \sum_{K_i \in \mathcal{K}^{(j)}} \epsilon_{K_i}^2 \tag{139}$$

where $\mathrm{dof}_{K_i}^l$ is the new number of local degrees of freedom after having applied pattern $l$ to the local FES and $\epsilon_{K_i}$ is a local error indicator. The parameter $\theta \in (0,1)$ can be chosen arbitrarily.
In practice, using that the summands in (138) are all positive by definition, we find the minimum by choosing $l_{K_i}$ in such a way that

$$l_{K_i} := \mathrm{argmin}_{l\in\{1,\ldots,5\}} \left\{ \frac{\mathrm{dof}_{K_i}^l}{\beta_{K_i}^l} \right\}. \tag{140}$$

Further we construct $\mathcal{A}^{(j)}$ in such a way that it is as small as possible, while still fulfilling the constraint.
In the case that the constraint can not be fulfilled, we will choose $\theta$ and $l_{K_i}$ in a way explained in the next paragraph. It would be favourable to make the constraint hold since one can show that satisfying the constraint implies uniform decrease of the error in energy norm as stated and shown in [3, p.1115 ff].

Thus we will choose $\theta \in (0,1)$ and $l_{K_i}$ in such a way that:

1. $\theta \in (0, \beta_0)$, if $\beta_0 > 0$
   where $\beta_0 := \min_{i\in\{1,..,M\}} \left(\min_{l\in\{1,..,5\}}(\beta_{K_i}^l)\right)$ and $l_{K_i}$ as explained above in (140).
2. $\theta \in (0, \beta_1 w_0)$, if $\beta_1 w_0 > 0$
   where $w_0 := \min_{i\in\{1,..,M\}} \left(\min_{l,l'\in\{1,..,5\}}(\frac{\mathrm{dof}_{K_i}^l}{\mathrm{dof}_{K_i}^{l'}})\right) - \epsilon$ for some small $\epsilon > 0$
   and $l_{K_i}$ as explained above in (140).
3. $\theta \in (0, \beta_1)$, if $\beta_1 > 0$
   where $\beta_0 := \min_{i\in\{1,..,M\}} \left(\max_{l\in\{1,..,5\}}(\beta_{K_i}^l)\right)$
   and $l_{K_i} := \mathrm{argmax}_{l\in\{1,..,5\}}(\beta_{K_i}^l)$.

where the list is sorted in descending priority. It is left to the reader to ensure that the conditions listed above in every option ensure the constraint to hold. If none of the conditions can be fulfilled, we continue by a global refinement step.

For more information about this *hp*-decider, see [3].

# 6 Numerical experiments and results

After the theoretical discussions, we are going to discuss the solution of the problem written down in (28) with $C = 1$ and $(a, b) = (0, 1)$ for the following right hand sides $f$:

$$f_1(x) = (1 + s\chi_{(v,w)}(x)) \sin(2\pi x) \qquad s, v, w \in \mathbb{R} \tag{141}$$

$$f_2(x) = \sin(2\pi \, \mathrm{e}^{\alpha x}) \qquad \alpha \in \mathbb{R}^+ \tag{142}$$

where

$$\chi_{(v,w)} := \begin{cases} 1 & x \in (v, w) \\ 0 & \text{otherwise} \end{cases}. \tag{143}$$

Looking at the a priori estimates written down in 2.5, we expect the adaptive FEM algorithm to converge exponentially in $\sqrt{N}$ for hp-refinement and algebraically in $N$ for pure $h$-refinement and pure $p$-refinement, where $N$ is the number of degrees of freedom.

## 6.1 Results for $f_1$

The function $f_1$ we have taken in our experiments to analize, how the algorithms behave at discontinuities.
For $f_1$, to be efficient, the algorithms should prefer $h$-refinement in regions around $v$ and $w$, since in $v$ and $w$ we have a discontinuity, and $p$-refinement in the other parts of $(0, 1)$, since the function is locally smooth there.

The results show that for $f_1$ all the algorithms react on the jumps in $v$ and $w$: Refinement is done mostly at the elements containing $v$ or $w$, where the local error is one of the largest. This is absolutely what we expect from theory and want the algorithms to do. But suprisingly, not only h-refinement has been applied at these discontinuities, but also p-refinement. In the results from all the algorithms, the highest local polynomial degree is found on the elements situated around $v$ and $w$.

The reason might be that the deciders often do not take the theoretically optimal decision. The decider solving the minimization problem is better in choosing a efficient refinement, as we see by comparing figures 1-4. A reasonable explanation could be, that the algorithm estimating the local analyticity of the function makes by construction a lot of approximations to compute some limits. The idea of the decider to solve a minimization problem, i.e. to compare the effect of different refinements on the error seems to be more reasonable and accurate. But we should have in mind that it is still only an approximation, which might cause errors. Thus, it is not surprising, knowing what the theoretical background tells us, that the convergence of the energy errors is quite a bit better for the minimization-decider. But there is no obvious difference between the algorithms using a different error estimator (c.f. exact errors in figure 5). The theoretically expected exponential convergence in $\sqrt{N}$ is given for all $hp$-adaptive algorithms, since the graphs describe more or less straight lines.

Analyzing the convergence plot in figure 5, we recognize that the equilibrated error estimator underestimates the error while the residual error estimator overestimates the error. From the theoretical point of view, both error estimators should overestimate the error with coefficient $C_{rel} = 1$. The residual error estimator behaves as expected, but with the equilibrated one, there might be something wrong. Since the behaviour of the estimator is similar to the exact error, we might think that the reason could be the data oscillation term, which has been neglected in the implementation. But neither taking higher order quadrature nor increasing the polynomial space, where we project the function $f$ on, redoes the underestimation of the equilibrated error estimator. Thus there might be a constant term missing in the implementation of the equilibrated error estimator which has not been found yet.

In spite of this underestimation, the algorithms using the equilibrated error estimator converge quite fast. The theoretically expected exponential convergence is given for all hp-adaptive algorithms, since the graphs in figure 5 describe more or less straight lines.

In figure 6 the convergence behaviour of algorithms combining the *residual error estimator* with the *weighted fixed energy fraction* and the *hp-decider solving the minimization problem* is plotted for different options of patterns (c.f. 5.2.2). We can see that both pure $h$- and $p$-refinement are not as fast converging as the $hp$-adaptive algorithms, again as the theoretical analysis predicts. But there is no obvious difference between the variuos algorithms using $h$- and $p$- refinement, except the last one (Pattern 1-5). This algorithm has, compared to the others, the additional choice to increase the local polynomial degree by 2.
Analyzing the refinement steps of this algorithm, we notice that the pattern 5 is chosen often, even on elements containing $v$ and $w$. This is in contrast to the theory which would recommend h-refinement there. As above, the decider prefers $p$-refinement, the reason is probably that the function is smooth resp. behaves nicely except in the two jumps in $v$ and $w$ (c.f. refinement behaviour of the smooth but highly oscillating function $f_2$, where tendencially more h-refinement has been chosen).

In the last figure containing data about the function $f_1$, we see the exact error plotted vs. the execution time of the different algorithms. The algorithm combining the *residual error estimator* with the *weighted fixed energy fraction* is the most efficient one. In general, the residual error estimator gives better results. The reason is probably the inefficient implementation of the equilibrated error estimator, since the convergence rates of the error vs. degrees of freedom are similar for both estimators.

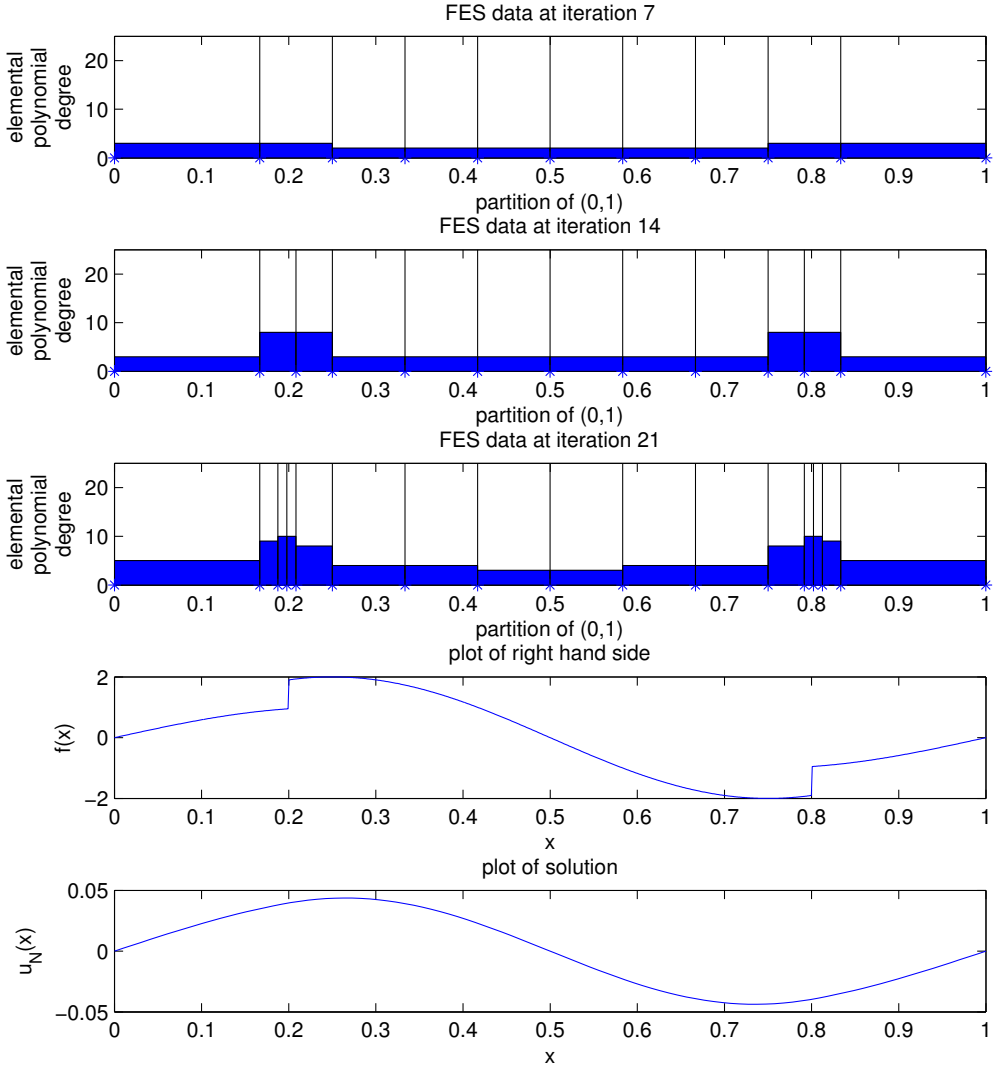


Figure 1: FES data of the adaptive algorithm combining the *residual error estimator* with the *maximum marking strategy* ($\theta = 0.2$) and the *hp decider estimating analyticity*($\theta = 0.3$) for $f_1$ with $v = 0.2$, $w = 0.8$, $s = 1$

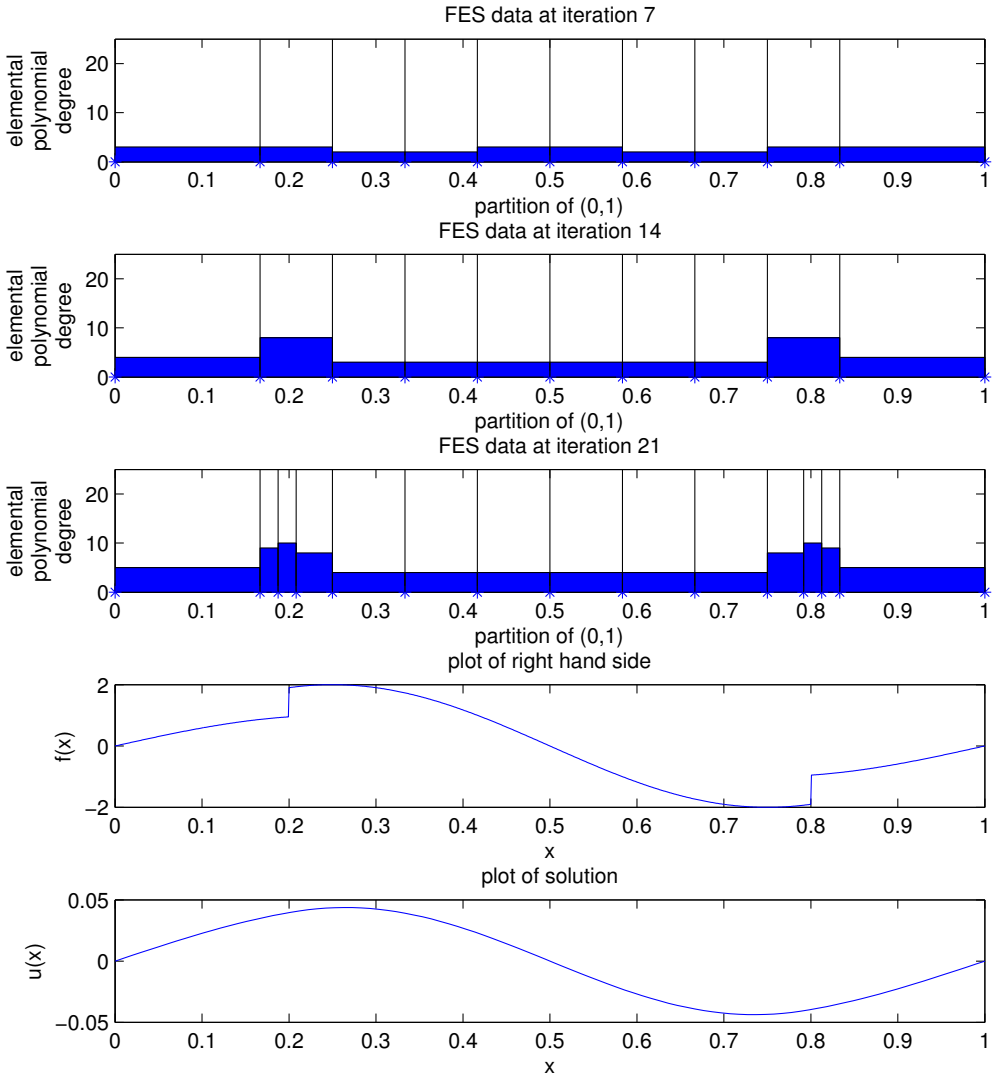


Figure 2: FES data of the adaptive algorithm combining the *equilibrated error estimator* with the *maximum marking strategy* ($\theta = 0.2$) and the *hp decider estimating analyticity*($\theta = 0.3$) for $f_1$ with $v = 0.2$, $w = 0.8$, $s = 1$

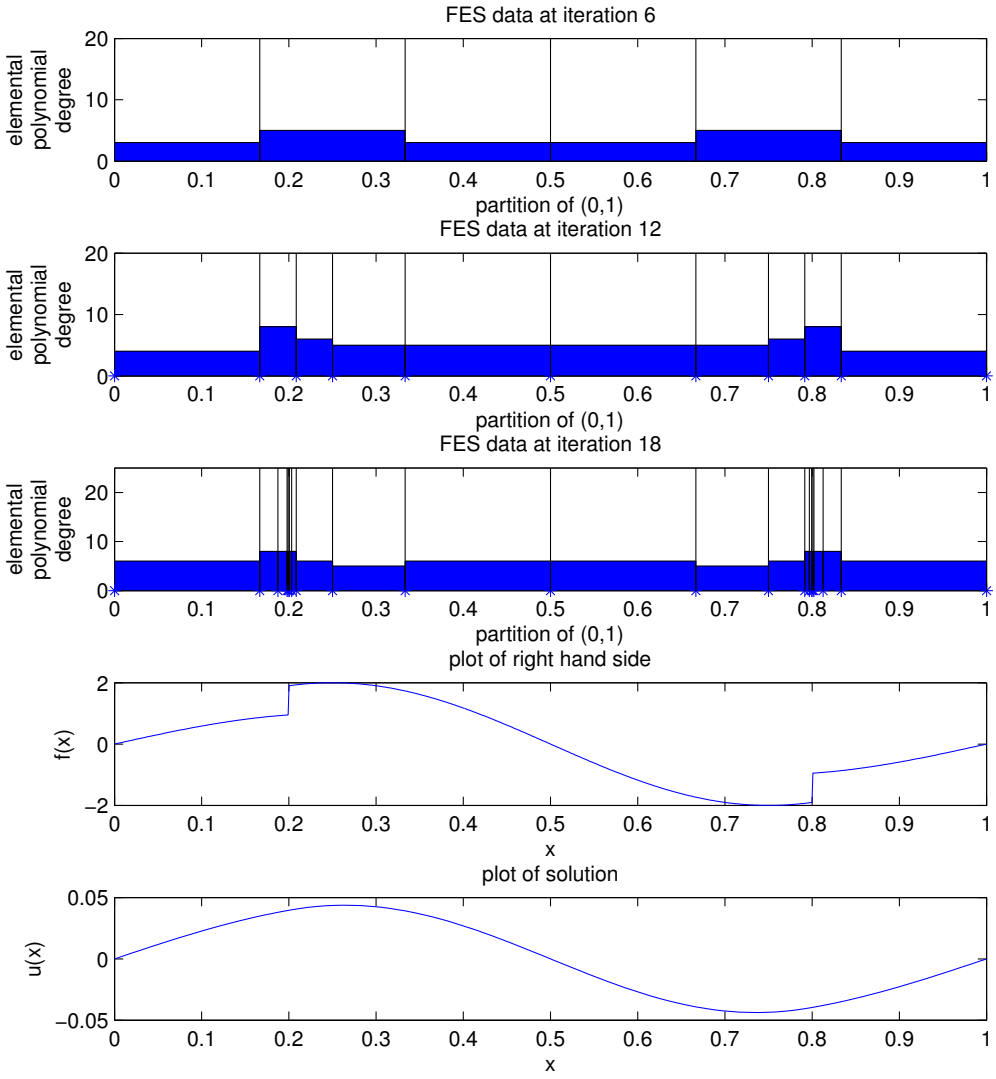


Figure 3: FES data of the adaptive algorithm combining the *residual error estimator* with the *weighted fixed energy fraction* ($\theta = 0.2$) and the *hp decider solving a minimization problem* for $f_1$ with $v = 0.2$, $w = 0.8$, $s = 1$

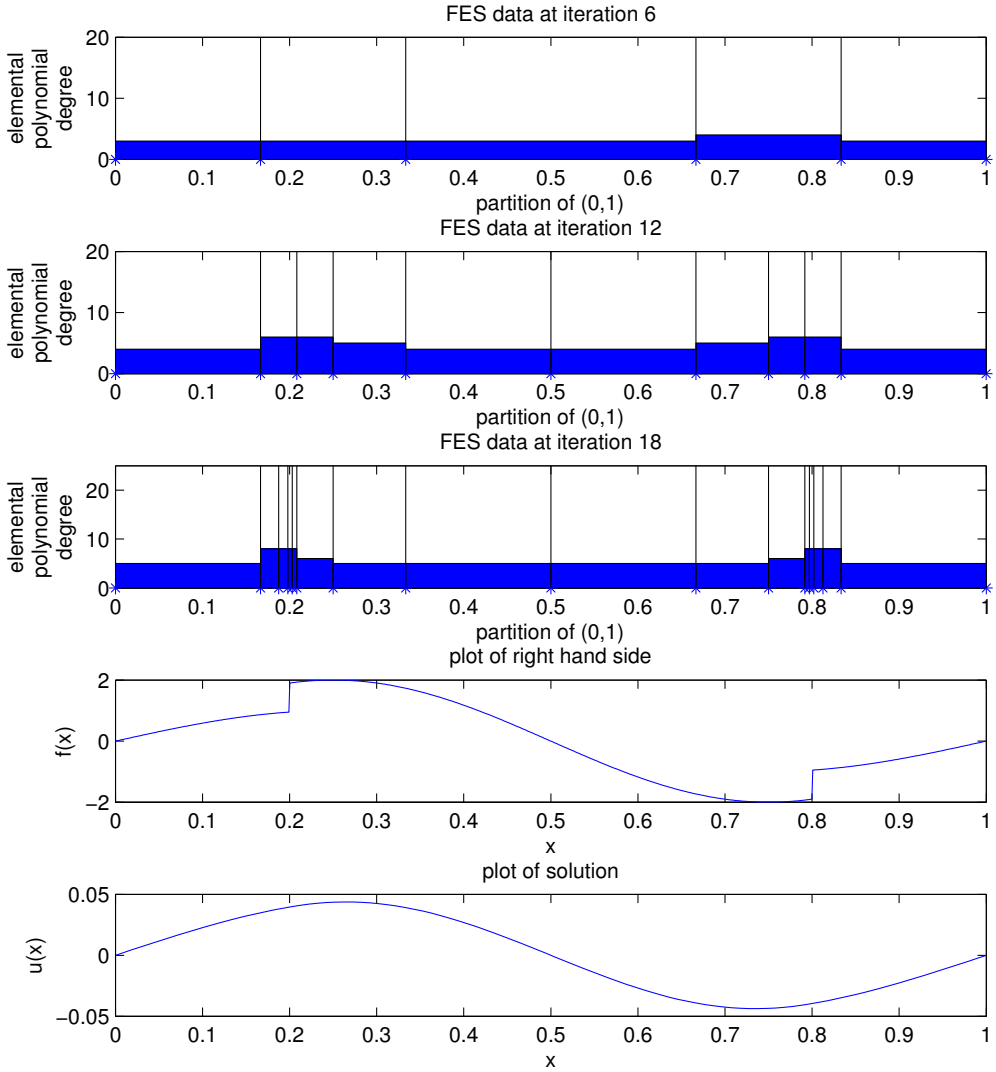


Figure 4: FES data of the adaptive algorithm combining the *equilibrated error estimator* with the *weighted fixed energy fraction* ($\theta = 0.2$) and the *hp decider solving a minimization problem* for $f_1$ with $v = 0.2$, $w = 0.8$, $s = 1$

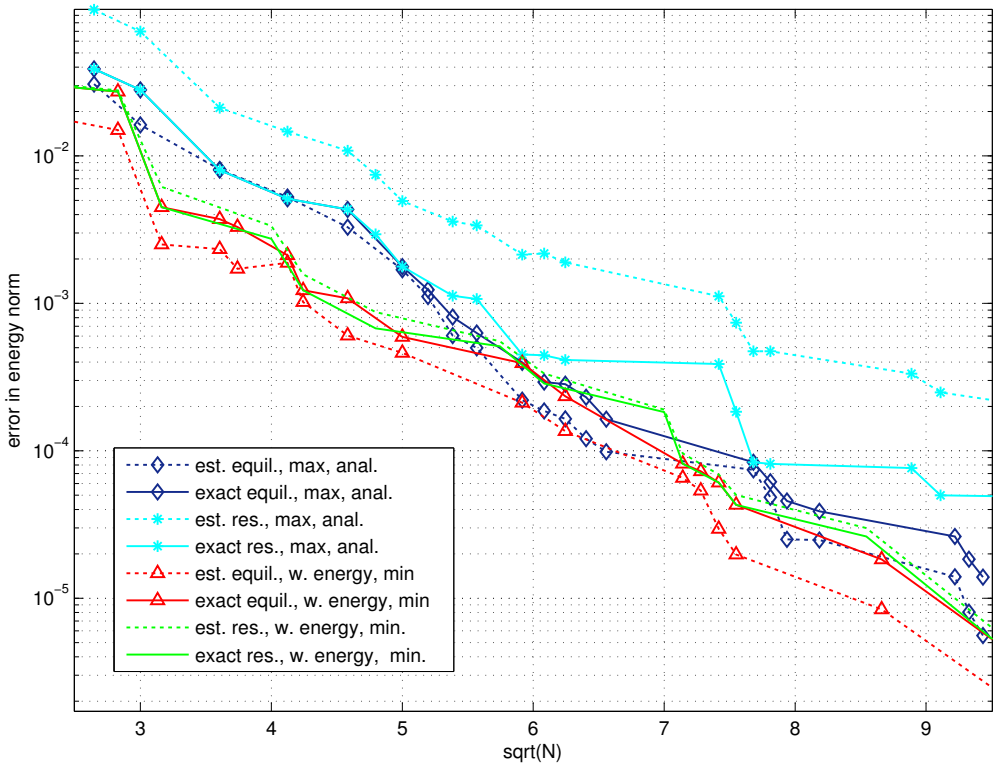


Figure 5: Convergency of the energy error for $f_1$ with $v = 0.2$, $w = 0.8$, $s = 1$, for the hp adaptive algorithms whose FES data is plotted above.
The abbreviations in the legend mean the following:
*est.*: estimated error in energy norm
*exact*: exact error in energy norm
*equil.*: equilibrated error estimator
*res.*: residual error estimator
*max*: maximum marking strategy
*w. energy*: weighted fixed energy fraction
*anal.*: hp decision by analyticity
*min.*: hp decision by solving the minimization problem

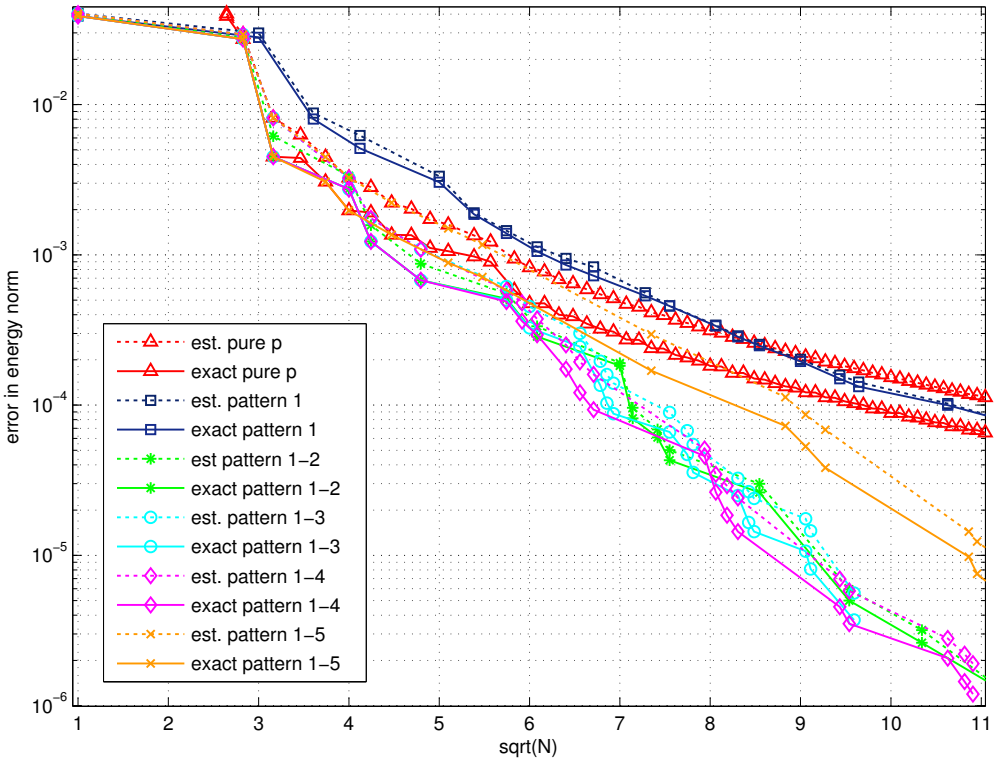


Figure 6: Convergency of the algorithm consisting of the residual error estimator, the weighted fixed energy fraction for marking elements and the hp decisioner which solves a minimization problem. Plotted is the convergence using different patterns (see first part of the subsection minimization) for $f_1$ with $v = 0.2$, $w = 0.8$ and $s = 1$.
The abbreviations in the legend mean the following:
*only p*: Only the common p-refinement is applied (algorithm uses only fixed energy fraction and then does p-refinement).
*pattern 1*: Only h-refinement (bisection) is applied (algorithm uses only fixed energy fraction and then does h-refinement).
*pattern 1-2*: In every step there is to choose between h- and p- refinement by solving the minimization problem.
*pattern 1-3*: As in in pattern 1-2 but additionally there is a pattern doing graded bisection ($h' = 0.15h$, $h'' = 0.85h$).
*pattern 1-4*: As in in pattern 1-3 but additionally there is a pattern doing another graded bisection ($h' = 0.85h$, $h'' = 0.15h$).
*pattern 1-5*: As in in pattern 1-4 but additionally there is a pattern doing another p-refinement ($p = p + 2$).

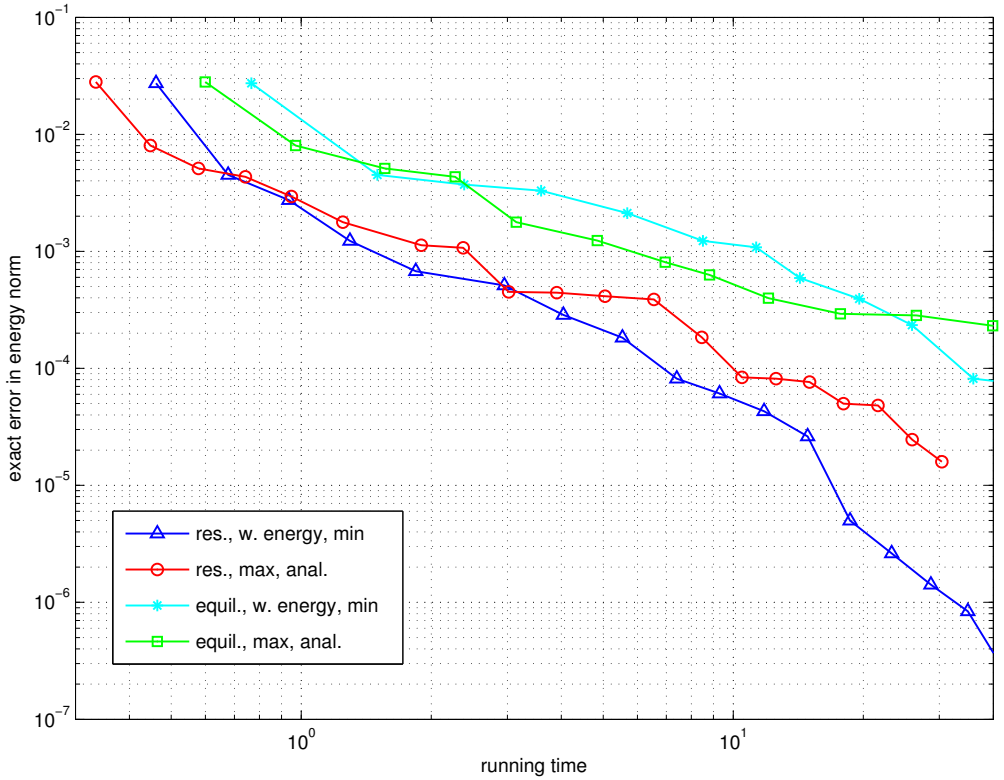


Figure 7: Efficiency of the algorithms in reducing the exact error in time for $f_1$ with $v = 0.2$, $w = 0.8$, $s = 1$, again for the hp adaptive algorithms whose FES data is plotted above.
The abbreviations in the legend mean the following:
*equil.*: equilibrated error estimator
*res.*: residual error estimator
*max*: maximum marking strategy
*w. energy*: weighted fixed energy fraction
*anal.*: hp decision by analyticity
*min.*: hp decision by solving the minimization problem

## 6.2 Results for $f_2$

The challenge for the algorithm is to get an accurate enough solution in spite of the high oscillation of $f_2$ near 1. $f_2$ is a smooth function and thus the *hp*-algorithms should do $p$-refinemnent the most to be efficient. Further it should notice the increasing oscillation of the function towards 1 and hence doing more and more refinement near 1.

The results nicely show that towards 1 there is indeed more refinement done than in the other regions. Near 1, the size of the elements is getting smaller, thus a lot of h-refinement has been applied. Nevertheless, doing $p$-refinement would be better, not just theoretically, as convergence plots confirm (c.f. convergence plot 13).

Again, with the decider estimating the local analyticity we get good (exponential convergence is again attained), but not as good convergence results as with the decider solving the minimization problem. Looking at the FES data, we observe that the analyticity-decider prefers h-refinement, whereas the other one does in general more p-refinement and only increases the h-refinement steps slightly near 1. That could be the reason for the better convergence of the second one.

Now we are going to examine the difference between the results of the two error estimators. Similarly to the case of $f_1$, the equilibrated error estimator underestimates the error. This underlines again the conclusion made during the analysis of $f_1$: there might be a constant missing in the implementation.

In figure 13 we see the convergence behaviour of algorithms combining the *residual error estimator* with the *weighted fixed energy fraction* and the *hp-decider solving the minimization problem* and having different options of patterns (c.f. 5.2.2). This time, the algorithm doing pure $p$-refinement steps converges best. This is exactly what we theoretically expect. Also the *hp*-algorithm who choses the refinement out of patterns $1-5$ converges better than the other hp-adaptive algorithms. Again, this is not surprising, since pattern 5, increasing the local polynomial degree by 2, would be the best choice ( $f_2$ is smooth). Nevertheless, there are only few differences between the exponential convergence behaviour of the *hp*-adaptive algorithms. Only the algorithm using pure $h$-refinement converges quite slowly and algebraically (we notice the slight decrease of the slope of the graphs marked by squares).

The graphs in figure 14, showing the efficiency of the algorithms, look quite different than for $f_1$. The convergence of the algorithms with the *hp*-decider estimating analyticity is better than the convergence of the others. This stands in contrast to the convergence of the exact energy error plotted relatively to the degrees of freedom. Moreover the algorithm using the equilibrated error estimator converges best.
An explication for this behaviour we find in the construction of the *hp*-deciders. The *hp*-decider estimating analyticity is much cheaper to compute than the other one. Therefore running time per iteration is quite a bit shorter. For the smooth function $f_2$, the faster running time per iteration wins over the algorithm's inefficient choice between h- and p-refinement. Thus, it is faster despite doing more refinement steps.

For $f_1$ this is not the case since we have discontinuities in $v$ and $w$. It makes a huge difference if the algorithm executes a p- or a h-refinement step. Doing h-refinement improves the error remarkably while applying p-refinement the error remains more or less the same. Thus it is much more important to choose the most efficient refinement pattern near discontinuities. It therefore pays off to use the more expensive but accurate minimization-decider.

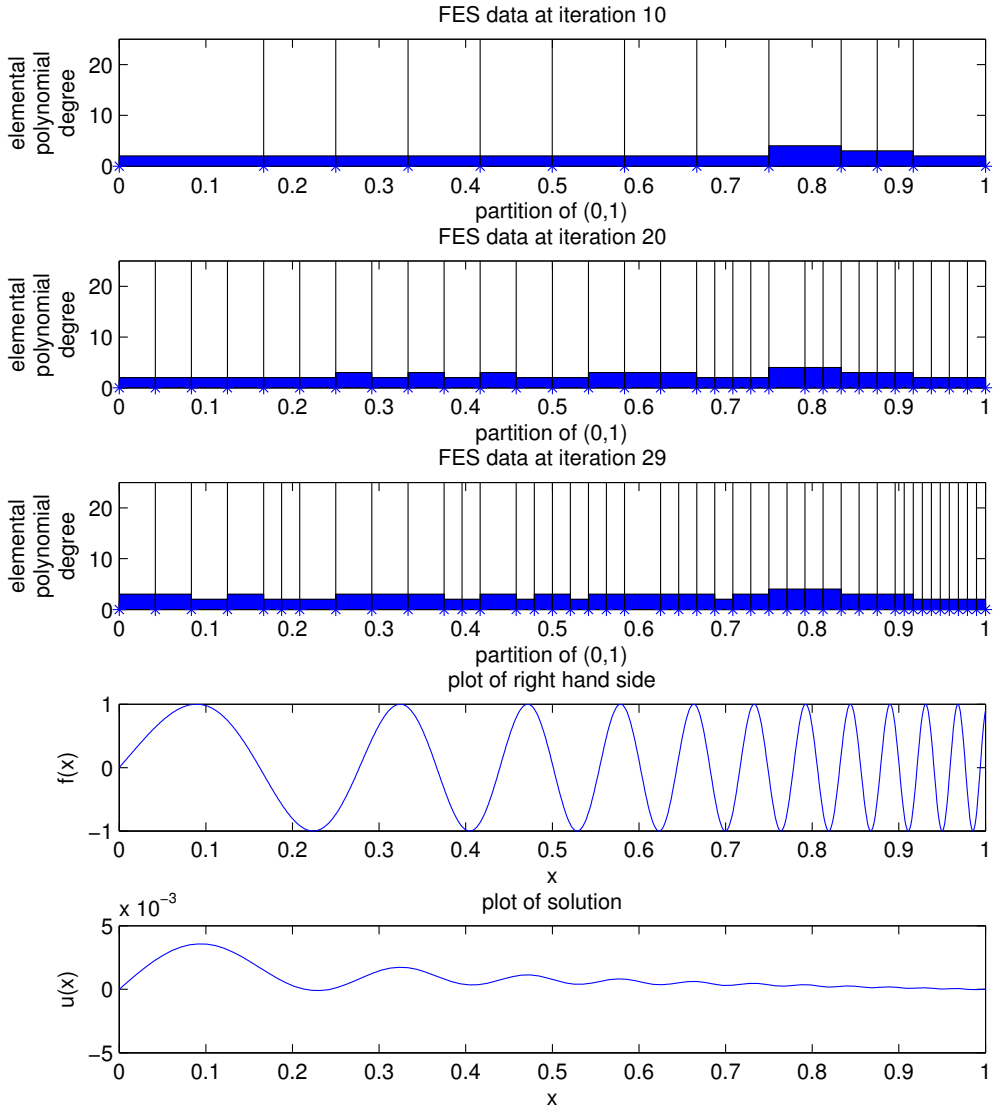


Figure 8: FES data of the adaptive algorithm combining the *residual error estimator* with the *maximum marking strategy* ($\theta = 0.2$) and the *hp decider estimating analyticity*($\theta = 0.3$) for $f_2$ with $\alpha = 2.5$

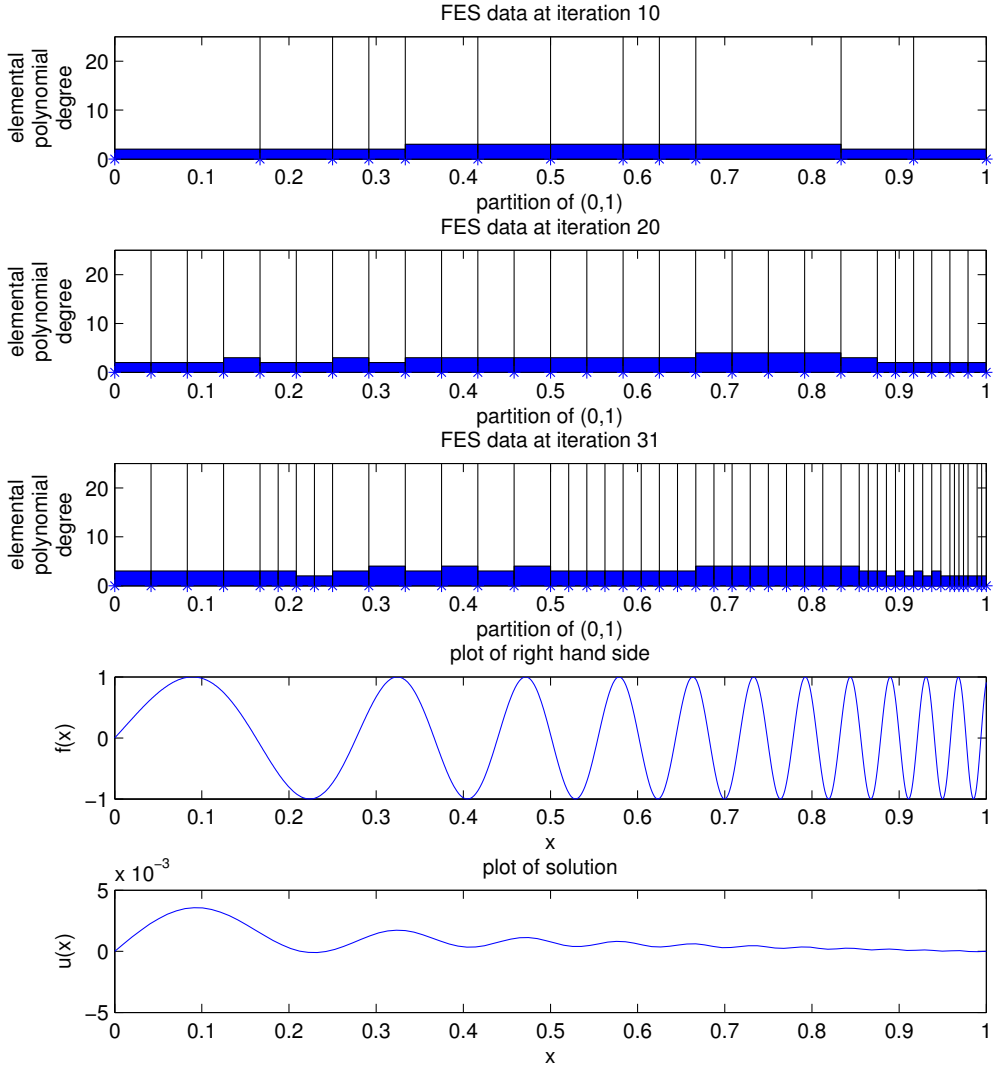


Figure 9: FES data of the adaptive algorithm combining the *equilibrated error estimator* with the *maximum marking strategy* ($\theta = 0.2$) and the *hp decider estimating analyticity*($\theta = 0.3$) for $f_2$ with $\alpha = 2.5$

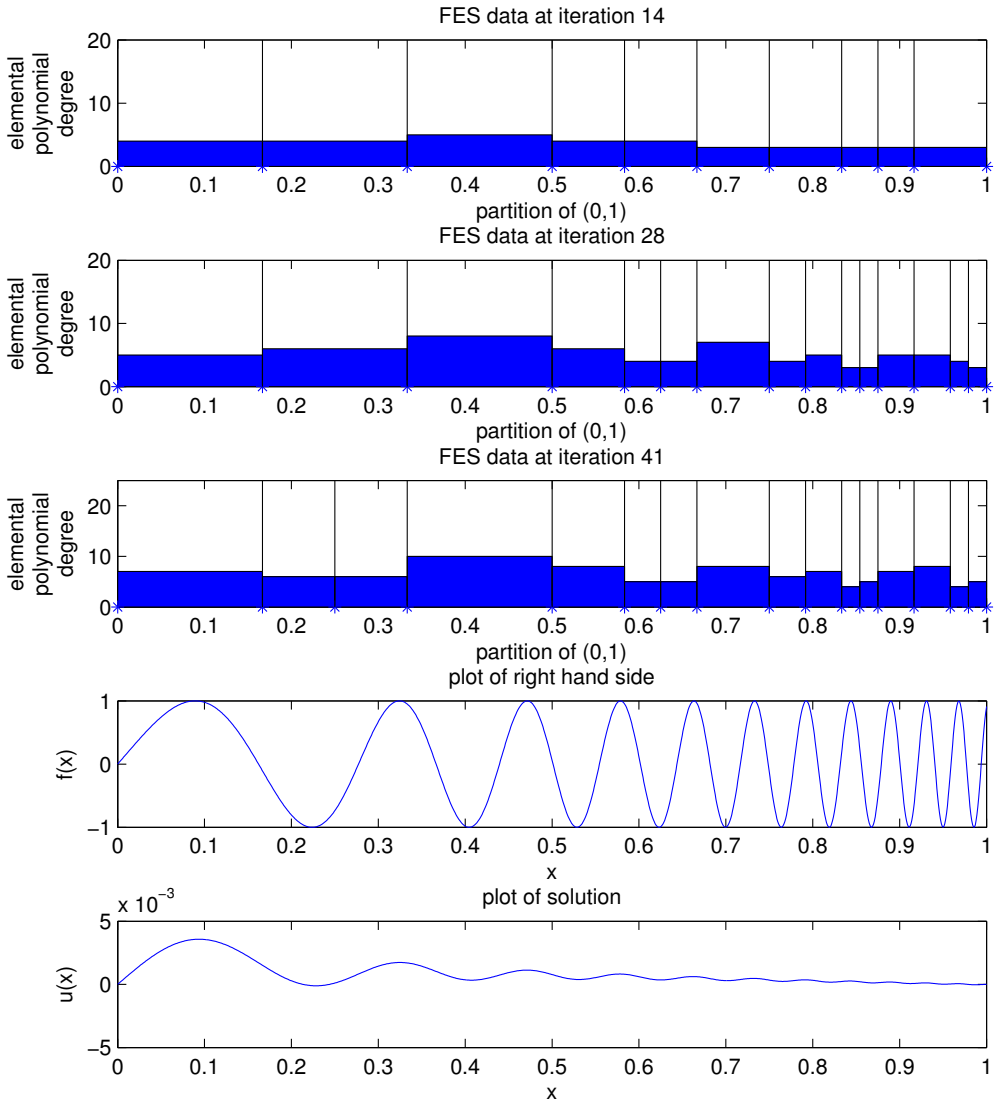


Figure 10: FES data of the adaptive algorithm combining the *residual error estimator* with the *weighted fixed energy fraction* ($\theta = 0.2$) and the *hp decider solving a minimization problem* for $f_2$ with $\alpha = 2.5$

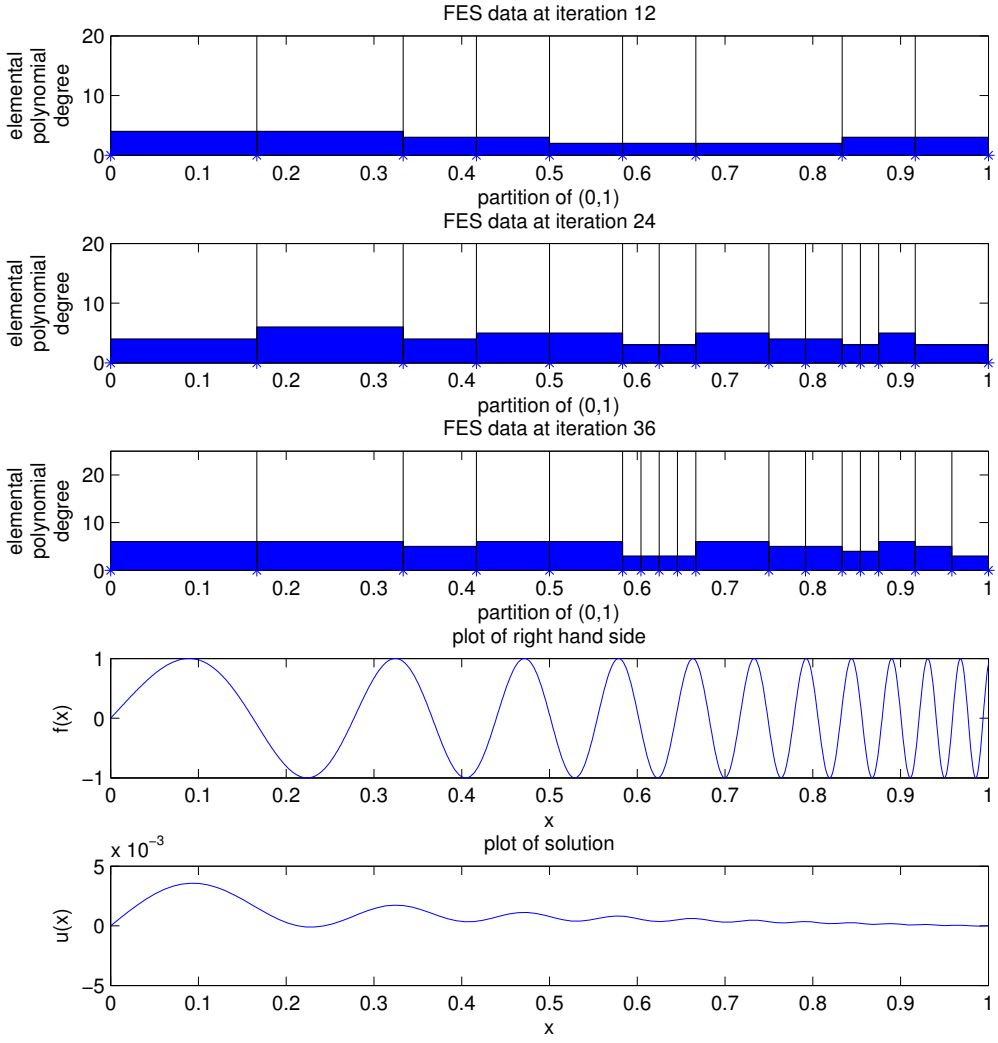


Figure 11: FES data of the adaptive algorithm combining the *equilibrated error estimator* with the *weighted fixed energy fraction* ($\theta = 0.2$) and the *hp decider solving a minimization problem* for $f_2$ with $\alpha = 2.5$

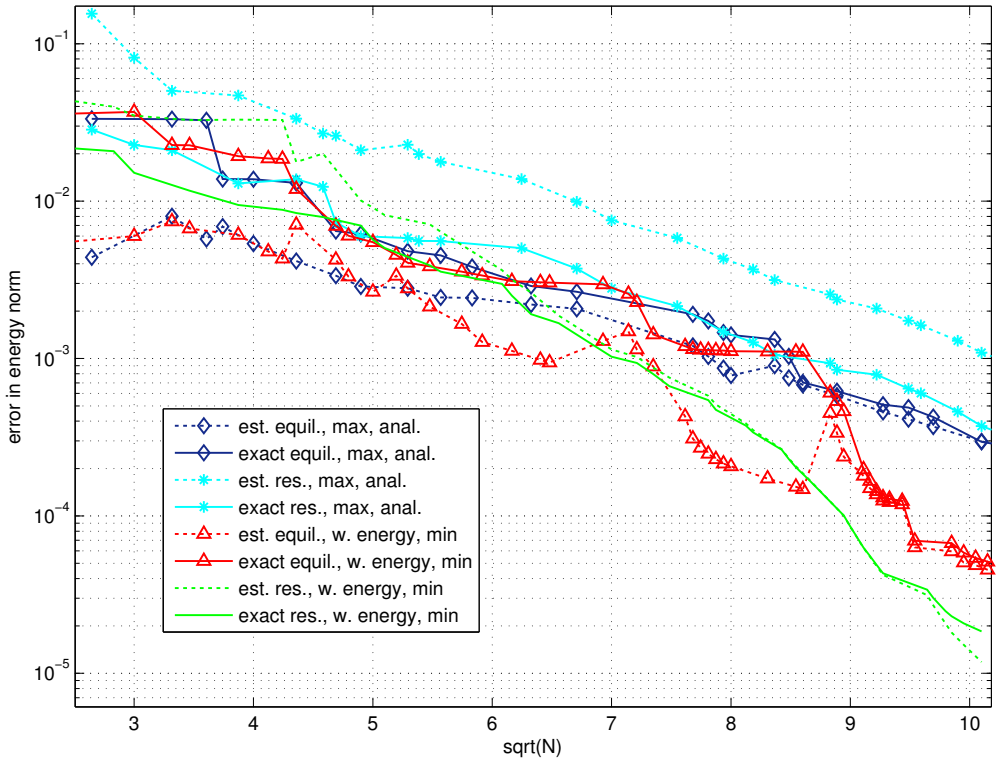


Figure 12: Convergency of the energy error for $f_2$ with $\alpha = 2.5$, for the hp adaptive algorithms whose FES data is plotted above.
The abbreviations in the legend mean the following:
*est.*: estimated error in energy norm
*exact*: exact error in energy norm
*equil.*: equilibrated error estimator
*res.*: residual error estimator
*max*: maximum marking strategy
*w. energy*: weighted fixed energy fraction
*anal.*: hp decision by analyticity
*min.*: hp decision by solving the minimization problem

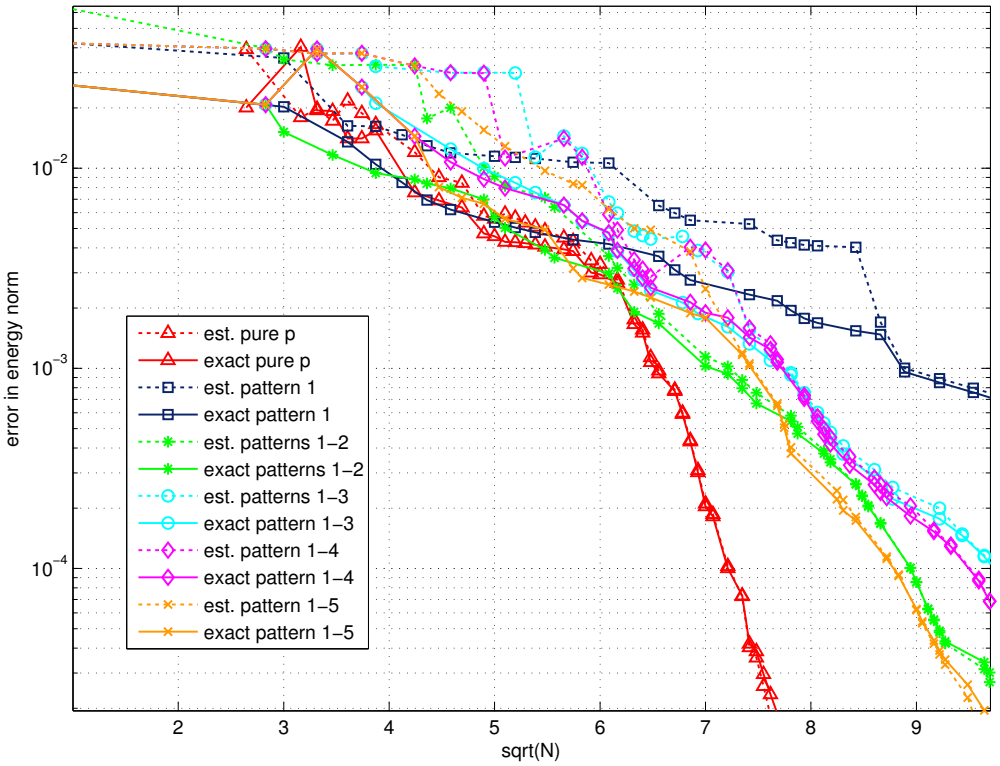


Figure 13: Convergency of the algorithm consisting of the residual error estimator, the weighted fixed energy fraction for marking elements and the hp decisioner which solves a minimization problem. Plotted is the convergence using different patterns (see first part of the subsection minimization) for $f_2$ with $\alpha = 2.5$.
The abbreviations in the legend mean the following:
*only p*: Only the common p-refinement is applied (algorithm uses only fixed energy fraction and then does p-refinement).
*pattern 1*: Only h-refinement (bisection) is applied (algorithm uses only fixed energy fraction and then does h-refinement).
*pattern 1-2*: In every step there is to choose between h- and p- refinement by solving the minimization problem.
*pattern 1-3*: As in in pattern 1-2 but additionally there is a pattern doing graded bisection ($h' = 0.15h$, $h'' = 0.85h$).
*pattern 1-4*: As in in pattern 1-3 but additionally there is a pattern doing another graded bisection ($h' = 0.85h$, $h'' = 0.15h$).
*pattern 1-5*: As in in pattern 1-4 but additionally there is a pattern doing another p-refinement ($p = p + 2$).

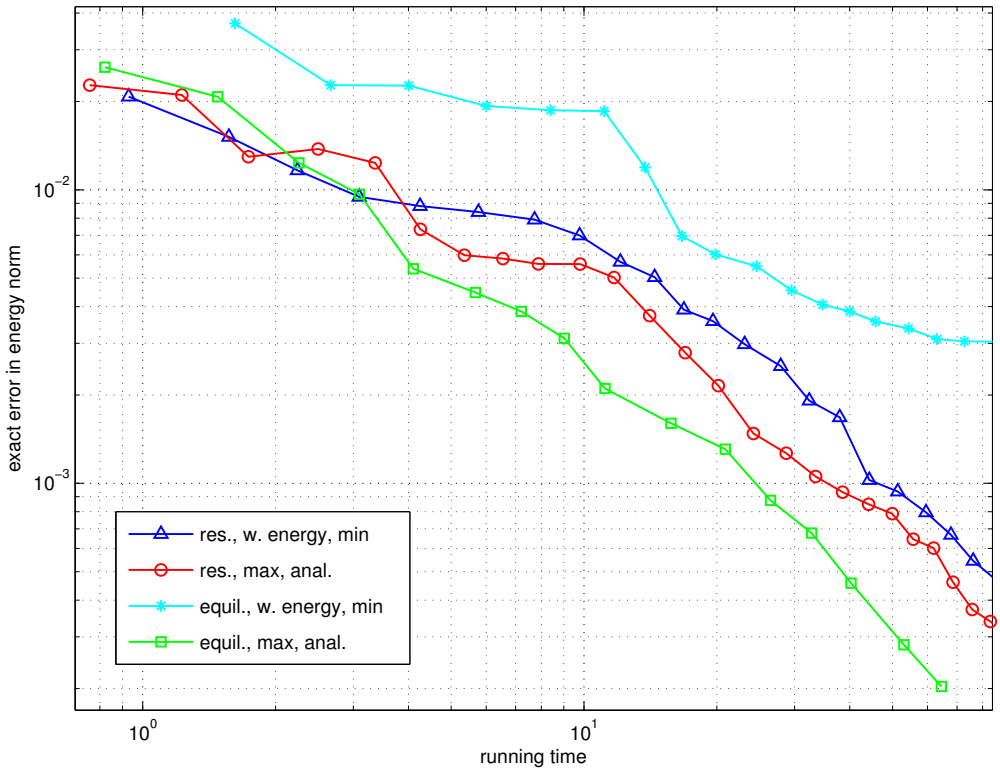


Figure 14: Efficiency of the algorithms in reducing the exact error in time for $f_2$ with $\alpha = 2.5$, again for the hp adaptive algorithms whose FES data is plotted above.
The abbreviations in the legend mean the following:
*equil.*: equilibrated error estimator
*res.*: residual error estimator
*max*: maximum marking strategy
*w. energy*: weighted fixed energy fraction
*anal.*: hp decision by analyticity
*min.*: hp decision by solving the minimization problem

# 7 Conclusion

Looking back we have theoretically derived and implemented different *hp*-adaptive algorithms, consisting of freely mixed combinations of two error estimators and two *hp*-deciders. Quite some time was needed for the implementation of the 1D-*hp*-FEM framework. Programming skills have been improved and a lot has been learned by planning this big framework.

The results of the implementations are satisfactory: All adaptive algorithms yield exponential convergence to the exact solution. The behaviour of the different error estimators is quite similar, although there remains an unexplained underestimation by the equilibrated error estimator. On the other hand we get noticable differences with the choice of the *hp*-deciders. Close inspection has shown that the decider estimating the analyticity of the solution can be recommended if the solution is expected to be quite smooth. In every case, the other decider is a more robust, albeit expensive choice.

Further investigations could search for the causes of the underestimation of the equilibrating error estimator, resp. find a probably missing constant. In general, the efficient implementation of the algorithms did not lie in the focus of this thesis and could be a natural choice for ongoing work.

# 8 Documentation of Implementation

## 8.1 General structure of the program

Here we will look at the main structure of the implementation. First there is a initialization of general parameters and variables.

```
%% Initialization

clear all;

% start timer
tstart = tic;

% Grid and PDE-coefficients
h = [1/3;1/3;1/3];
p = 2*ones(size(h));
a = 0;
C = ones(size(p));
c = zeros(size(p));

% Weight for marking elements, has to be in (0,1)
theta = 0.2;

% parameter, necessary for knapsack algorithm (used in fixed energy fraction)
nu = 0.1;

% parameter, used in hp-decider (analyticity)
gamma = 0.3;

% stores the number of h- or p-refinements
% o is initialized for the hp-decider (min), 1<=o<=5,
% for the other decider (analyticity) it is set to o=2.
o = 2;
number = zeros(1,o);

% tolerance and iteration control
tol = 10^(-5);
iter = 1;
maxiter = 30;
oo = 1;
```

Now, the right hand side is choosen and the according first derivative of the exact solution is calculated.

```
%% Initialization of function f_1

% right hand side
v = 0.2;
w = 0.8;
s = 1;
f = @(x) sin(2*pi*x).*(1+s*(x>=v).*(x<=w));

% exact solution and first derivative
ei = s*sin(2*pi*v)/(2*pi)^2;
en = s*sin(2*pi*w)/(2*pi)^2;
ek = s*cos(2*pi*v)/(2*pi);
el = s*cos(2*pi*w)/(2*pi);
e = sin(2*pi)/(2*pi)^2;
c22 = v*ek-ei;
c3 = el*w-en-c22;
```

```
c33 = -c3;
c2 = c3-el;
c1 = ek+c2;
u = @(x) sin(2*pi*x)/(2*pi)^2.*(1+s*(x>=v).*(x<=w))+...
    (x<=v).*c1.*x+(c2.*x+c22).*(x>=v).*(x<=w)+...
    (x>=w).*(c3.*x+c33);
u_diff = @(x) cos(2*pi*x)/(2*pi).*(1+s*(x>=v).*(x<=w))+c1*(x<=v)+c2*(x<=w).*(x>=v)+...
    c3*(x>=w);
```

or

```
%% Initialization of function f_2

% right hand side
alpha=2.5;
f = @(x) sin(2*pi*exp(alpha*x));

% first derivative of solution
const=-1/(2*pi*(1-exp(alpha)))*...
    (cos(2*pi)+2*pi*double(sinint(2*pi))-...
    cos(2*pi*exp(alpha))-2*pi*exp(alpha)*double(sinint(2*pi*exp(alpha))))+0.0029;
u_diff=@(x) -1/alpha*double(sinint(2*pi*exp(2.5*x))+const);
```

Then the first error estimation is evaluated.

```
%% First error estimation

% evaluates the the quadrature nodes for order of input
quad = gau_leg(max(p)+1);
quad2 = gau_leg(2*max(p)+5);

% computes the galerkin solution
[u_N,A] = calc_solution(a,h,p,f,C,c,quad,v,w);

% calculates the estimated local errors
[eps] = equilibrated_error_indicator(f,u_N,a,h,p,C,v,w,quad);

% global error
err = sqrt(sum(eps));

% store degrees of freedom
dof = sum(p)+1;

% calculates the exact error in energy norm
err_exact(iter) = exact_error_energy(u_diff,u_N,a,h,p,quad2,C,v,w)
```

where in row 11 there we insert the chosen estimator. In the code printed above the equilibrated error estimator is used. One can also substitute by

```
[eps] = error_indicator(a,u_N,p,h,f,quad,C,v,w);
```

to apply the residual error estimator.

After that, the loop checking the breaking criteria begins. Then the marking and refinement step and the error estimation of the refined FES is done:

```
%%% Iteration step

% slope, checking the breaking criteria
while(err(iter) > tol && iter <= maxiter)

%% marking elements
```

```
mark = marker_max(theta,eps);

%% hp-decider
marks = mark_h_p_analyt_ext(mark,gamma,u_N,h,p);

% actualization of the number of refinements
number(1,1) = number(1,1)+sum(marks(:,1));
number(1,2) = number(1,2)+sum(marks(:,2));

%% refinement step
p = p_increase(p,marks(:,2),1);
[h,p,c,C] = h_graded_bisection(h,p,c,C,marks(:,1),0.5);

% actual time is stored
t(iter)=toc(tstart);

% iteration is finished, new iteration begins
iter = iter +1;

%% error estimation
% calculates new nodes and weights of quadrature
quad = gau_leg(max(p)+1);
quad2 = gau_leg(2*max(p)+5);

% new Galerkin solution
[u_N,A] = calc_solution(a,h,p,f,C,c,quad,v,w);

% estimated new local error
eps=equilibrated_error_indicator(f,u_N,a,h,p,C,v,w,quad);

% global error
err(iter) = sqrt(sum(eps))

% new degrees of freedom is stored
dof(iter)=sum(p)+1;

% new exact error in energy norm
err_exact(iter)=exact_error_energy(u_diff,u_N,a,h,p,quad2,C,v,w)
```

for the *hp*-decider which estimates the analyticity and

```
%%% Iteration step

% slope, checking the breaking criteria
while(err(iter) > tol && iter <= maxiter)

%% marking elements and hp-decider
[mark,I] = marker_energy_loc(theta,eps,norm_res,nu,a,p,h,C,c,f,u_N,quad,v,w,o);

% actualization of the number of refinements and construction of marking vectors
marks = zeros(length(I),5);
for k = 1:length(p)
    for j = 1:o
        if mark(k) == 1 && I(k) == j
            marks(k,j) = 1;
            number(1,j) = number(1,j)+1;
        end
    end
end

%% refinement step
p = p_increase(p,marks(:,2),1);
p = p_increase(p,marks(:,5),2);
```

```
[h,p,c,C] = h_graded_bisection(h,p,c,C,...
    [marks(:,1),marks(:,3),marks(:,4)],[0.5,0.15,1-0.15]);

% actual time is stored
t(iter)=toc(tstart);

% iteration is finished, new iteration begins
iter = iter +1;

%% error estimation
% calculates new nodes and weights of quadrature
quad = gau_leg(max(p)+1);
quad2 = gau_leg(2*max(p)+5);

% new Galerkin solution
[u_N,A] = calc_solution(a,h,p,f,C,c,quad,v,w);

% estimated new local error and L2-norm of the residual, needed for the hp-decider
[norm_res,~] = error_indicator(a,u_N,p,h,f,quad,C,v,w);
eps=equilibrated_error_indicator(f,u_N,a,h,p,C,v,w,quad);

% global error
err(iter) = sqrt(sum(eps))

% new degrees of freedom is stored
dof(iter)=sum(p)+1;

% new exact error in energy norm
err_exact(iter)=exact_error_energy(u_diff,u_N,a,h,p,quad2,C,v,w)
```

for the *hp*-decider solving the minimization problem. Again, one can substitute the equilibrated error estimator by the residual one.

It follows the implementation of different algorithms, necessary for the computations.

## 8.2 Galerkin solution

The basis function used in the implementation are the 1D hierarchic shape functons $\{b_n\}_{n=1}^{Mp+1}$ for $(S)^0_{\mathbf{p}}(\mathcal{M})$ on $\Omega = (a, c)$, $a \leq c$.

To be able to define and understand this basis, we first introduce the following local shape functions $\{\widehat{b_n}\}_{n=1}^{p+1}$, defined on $(-1, 1)$:

$$\widehat{b_1}(x) = \frac{1-x}{2} = \frac{L_0(x) - L_1(x)}{2}, \tag{144}$$

$$\widehat{b_2}(x) = \frac{1+x}{2} = \frac{L_0(x) + L_1(x)}{2}, \tag{145}$$

$$\widehat{b_n}(x) = \sqrt{\frac{2n-3}{2}} \int_{-1}^{x} L_{n-2}(t)dt \tag{146}$$

$$= \frac{1}{\sqrt{2(2n-3)}} \left(L_{n-1}(x) - L_{n-3}(x)\right), \; p+1 \geq n \geq 3 \tag{147}$$

Then the symmetric mass matrix $\widehat{\mathbf{M}}$ and the matrix $\widehat{\mathbf{K}}$ defined below have the following form:

$$\widehat{\mathbf{M}} = (\int_{-1}^{1} \widehat{b}_i(x)\, \widehat{b}_j(x)\, dx)_{i,j=1}^{p+1} \tag{148}$$

$$\widehat{\mathbf{M}} = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} & -\frac{1}{\sqrt{6}} & \frac{1}{3\sqrt{10}} & 0 & \cdots & \cdots & & 0 \\ & \frac{2}{3} & -\frac{1}{\sqrt{6}} & -\frac{1}{3\sqrt{10}} & 0 & \cdots & \cdots & & 0 \\ & & \widehat{M}_{3,3} & 0 & \widehat{M}_{3,5} & \ddots & & & 0 \\ & & & \ddots & \ddots & \ddots & & & \vdots \\ & & & & \widehat{M}_{i,i} & 0 & \widehat{M}_{i,i+2} & \ddots & \vdots \\ & & & & & \ddots & \ddots & \ddots & 0 \\ & & & & & & \widehat{M}_{p-1,p-1} & 0 & \widehat{M}_{p-1,p+1} \\ & & & & & & & \widehat{M}_{p,p} & 0 \\ \text{sym} & & & & & & & & \widehat{M}_{p+1,p+1} \end{pmatrix}$$

where $\widehat{M}_{i,i} = \frac{2}{(2i-1)(2i-5)}$ and $\widehat{M}_{i,i+2} = \frac{-1}{(2i-1)\sqrt{(2i-3)(2i+1)}}$

$$\widehat{\mathbf{K}} = (\int_{-1}^{1} \widehat{b}_i'(x)\, \widehat{b}_j'(x)\, dx)_{i,j=1}^{p+1} \tag{149}$$

$$\widehat{\mathbf{K}} = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & 0 & \cdots & \cdots & 0 \\ & \frac{1}{2} & 0 & \cdots & \cdots & 0 \\ & & 1 & 0 & \cdots & 0 \\ & & & \ddots & \ddots & \vdots \\ & & & & \ddots & 0 \\ \text{sym} & & & & & 1 \end{pmatrix}$$

Now we want to work on the domain $\Omega = (a, b)$, where $a \leq b$ together with the following mesh:
For $M \in \mathbb{N}$:

$$\mathcal{M} = \{K_i\}_{i=1}^{M} \tag{150}$$

$$K_i := (a + \sum_{j=1}^{i-1} h_j, a + \sum_{j=1}^{i} h_j),\ i \in \{1, 2, ..., M\}, \tag{151}$$

$$\forall j\ :\ h_j > 0, \sum_{j=1}^{M} h_j = c - a \tag{152}$$

To get the 1D hierarchic shape functions for $(S)_{\mathbf{p}}^{0}(\mathcal{M})$ defined on this interval, we do the following transformations:
For $i \in \{1, 2, ..., M\}$:

$$\phi_i \colon K_i \to (-1, 1) \tag{153}$$

$$x \mapsto \phi_i(x) := -1 + 2\frac{x - a - \sum_{j=1}^{i-1} h_j}{h_i} \tag{154}$$

The basis functions $\{b^i\}_{i=1}^{\sum_{j=1}^{M} p_j+1}$ of the whole space $(S)^0_{\mathbf{p}}(\mathcal{M})$ are defined as follows:

The basis functions associated with node $i \in \{0,1,...,M\}$ are for $i \in \{0,1,...,M\}$:

$$b^{\sum_{j=0}^{i} p_j+1}(x) = \begin{cases} \widehat{b}_2(\phi_i(x)) & x \in K_i\,,\ (K_0 = \emptyset) \\ \widehat{b}_1(\phi_{i+1}(x)) & x \in K_{i+1}\,,\ (K_{M+1} = \emptyset) \\ 0 & \text{otherwise} \end{cases} \tag{155}$$

The basis functions associated with element $K_i$ are the following:
For $i \in \{1,2,...,M\}$ and $n \in \{2,3,...,p_i\}$:

$$b^{\sum_{j=0}^{i-1} p_j+n}(x) = \begin{cases} \widehat{b}_{n+1}(\phi_i(x)) & x \in K_i \\ 0 & \text{otherwise} \end{cases} \tag{156}$$

To make life easier, we define a local-global mapping $\gamma_i$ for $i \in \{1,2,...,M\}$:

$$\gamma_i \colon \{1,...,p_{K_i}\} \to \{1,2,... \sum_{j=1}^{M} p_j + 1\} \tag{157}$$

$$x \mapsto \gamma_i(x) = \begin{cases} \sum_{j=1}^{i-1} p_j + 1 & x = 1 \\ \sum_{j=1}^{i} p_j + 1 & x = 2 \\ \sum_{j=1}^{i-1} p_j + x + 1 & \text{otherwise} \end{cases} \tag{158}$$

The Galerkin matrix $\mathbf{A}^i$ of $K_i$ now is obtained by a linear combination of $\widehat{\mathbf{M}}$ and $\widehat{\mathbf{K}}$:

$$\mathbf{A}^i = \Big( \int_a^b \mathbf{C}(x) b^{\gamma_i(n)\prime}(x)\, b^{\gamma_i(k)\prime}(x)\, dx + \int_a^b c(x) b^{\gamma_i(n)}(x)\, b^{\gamma_i(k)}(x)\, dx \Big)_{n,k=1}^{p_i+1} \tag{159}$$

$$= \Big( C_i \int_{-1}^{1} \widehat{b^n}'(x)\, \widehat{b^k}'(x)\, \phi_i(x)'\, dx + c_i \int_{-1}^{1} \widehat{b^n}(x)\, \widehat{b^k}(x)\, \phi_i^{-1\prime}(x)\, dx \Big)_{n,k=1}^{p_i+1} \tag{160}$$

$$= \frac{2}{h_i} C_i \widehat{\mathbf{K}} + \frac{h_i}{2} c_i \widehat{\mathbf{M}} \tag{161}$$

where the $\frac{2}{h_i}$ respectively $\frac{h_i}{2}$ terms come from transformation of the integral (determinants of $\phi_i$ respectively $\phi_i^{-1}$).

The global Galerkin matrix $\mathbf{A}$ now is obtained by beginning with $\mathbf{A} = \mathbf{0}$ and then adding all local matrices $\mathbf{A}^i$ on the correct places (see numbering of bases in (156),(155)). We do this by using our local-global mappings $\gamma_i$ for $i \in \{1,2,...,M\}$ and get $\mathbf{A}$ by summing over all $i \in \{1,2,...,M\}$ and all $j,k \in \{1,2,...,p_i+1\}$

$$\mathbf{A}(\gamma_i(k), \gamma_i(j)) = \mathbf{A}(\gamma_i(k), \gamma_i(j)) + \mathbf{A}^i \tag{162}$$

The load vector $\mathbf{f}^i$ on the element $K_i$ one gets by evaluating

$$\mathbf{f}^i = (\int_{-1}^{1} f(\phi_i^{-1}(x))\,\widehat{b_n}(x)\,\phi_i^{-1\prime}(x)\,dx)_{n=1}^{p_i+1} \tag{163}$$

$$= (h_i/2 \sum_{k=1}^{\max_{j\in\{1,\dots,M\}}(p_i)+1} w_k f(\phi_i^{-1}(x_k))\,\widehat{b_n}(x_k))_{n=1}^{p_i+1}. \tag{164}$$

We use the Gauss-Legendre quadrature formula (computed by the Golub/Welsch algorithm cf. [5]) with $\max_{j\in\{1,\dots,M\}}(p_j)+1$ nodes, which gives us an order of at least $2\max_{j\in\{1,\dots,M\}}(p_j)+2$ which guarantees exact integration of polynomials of degree $2\max_{j\in\{1,\dots,M\}}(p_j)+1$. We write

$$f(\phi_i^{-1}(x)) = f^{K_i}_{\max(p_j)}(\phi_i^{-1}(x)) + (f(\phi_i^{-1}(x)) - f^{K_i}_{\max(p_j)}(\phi_i^{-1}(x))) \tag{165}$$

where $(f^{K_i}_{\max(p_j)}(\phi_i^{-1}(x)))$ is the projection of $f(\phi_i^{-1}(x))$ onto $\mathcal{P}_{2\max(p_j)+1}(K_i)$ produced by interpolation of f by Lagrange polynomials and nodes chosen at the zeros of $L_{\max p_j+1}(x)$, the $(\max p_j+1)^{\text{th}}$ Legendre polynomial.
We assume the energy norm of the second term to converge higher order than the error in energy norm.

The global load vector $\mathbf{f}$ we get by the same procedure as for $\mathbf{A}$:
For $i \in \{1,2,...,M\}$ and $j \in \{1,2,...,p_i+1\}$

$$\mathbf{f}(\gamma_i(j)) = \mathbf{f}(\gamma_i(j)) + \mathbf{f}^i \tag{166}$$

Finally we have implemented the whole linear system $\mathbf{Au} = \mathbf{f}$ and get $\mathbf{u}$ by simply solving it with the backslash operator in Matlab.

In this thesis, we work only with homogeneous Dirichlet boundary conditions, so we simply neglect the two basis functions which are associated with the nodes on the boundary of omega, i.e. we just solve the system

$$(a_{k,l})_{k,l=2}^{\sum_{j=1}^{M} p_j} (u_l)_{l=2}^{\sum_{j=1}^{M} p_j} = (f_l)_{l=2}^{\sum_{j=1}^{M} p_j} \tag{167}$$

and set $u_1, u_2 = 0$.
The ideas and formulas, not having an extra citation added, are found in the text above are found in [8] at p.$57-66$ or in [4].

Below, the implemented algorithms to solve the Galerkin problem and for evaluating the Galerkin solution and its derivatives are listed:

```
% This function calculates the Gauss-Legendre quadrature weight vector
% data.w and the vector containing the quadrature nodes data.x of length p,
% i.e. the Gauss-Legendre quadrature of order 2p on the interval (-1,1)


% Computations are after Golub/Welsh which showed that the eigenvalues of
% the matrix J computed below are the roots of the Legendre polynomial.

function data = gau_leg(p)

%initialization
```

```
b = 1:p;
b = b./sqrt(4*b.*b-1);
J = diag(b,-1)+diag(b,1);

% computation of eigenvectors and values
[ev,ew] = eig(J);
for i = 1:p+1
    ev(:,i) = ev(:,i)./norm(ev(:,i));
end

% calculate output arguments
data.x = diag(ew);
data.w = 2*(ev(1,:).*ev(1,:));
end
```

```
%This function computes the local mass matrix M_hat for 1D hierarchic
%shape funcions and initial value p (local polynomial degree).

function [M_hat] = local_M(p)
if p < 1
    error('input argument has to be p>=1',p)
else M_hat=zeros(p+1);
     M_hat(1:2,1:2) = [2/3,1/3;1/3,2/3];
     if p >= 2
         M_hat(1,3) = -1/sqrt(6);
         M_hat(2,3) = M_hat(1,3);
         M_hat(3,1) = M_hat(1,3);
         M_hat(3,2) = M_hat(1,3);
     end
     if p >= 3
         M_hat(1,4) = 1/(3*sqrt(10));
         M_hat(2,4) = -M_hat(1,4);
         M_hat(4,1) = M_hat(1,4);
         M_hat(4,2) = M_hat(2,4);
     end
     for i = 3:p+1
         M_hat(i,i) = 2/((2*i-1)*(2*i-5));
         if(p+1 >= i+2)
            M_hat(i,i+2) = -1/((2*i-1)*sqrt((2*i-3)*(2*i+1)));
            M_hat(i+2,i) = M_hat(i,i+2);
         end
     end
end
end
```

```
% This function computes the local matrix K_hat, containing inner product
% of derivatives of 1D hierarchic basis functions for initial value p (local
% polynomial degree).

function [K_hat] = local_K(p)
if p < 1
    error('input argument has to be p>=1',p)
else K_hat = eye(p+1);
     K_hat(1:2,1:2) = [0.5,-0.5;-0.5,0.5];
end
```

```
% This function evaluates the 'number'th local 1D-hierarchic basis
% function  in x.

function y = local_b(number,x)

if number == 1
```

```
    y = (1-x)/2;
elseif number == 2
    y = (1+x)/2;
else
    y = (legendre_pol(number-1,x)-legendre_pol(number-3,x))...
        /sqrt(2*(2*number-3));
end
end
```

```
% This function evaluates the transformation phi_inv:(-1,1)-->Ki
% in x.
% The needed inputs are:
% i           number of element
% x           evaluating point
% a           from omega=(a,b)
% h           vector containing h(i), the length of the
%             element/interval i

function y = phi_inv(i,x,a,h)
    y = h(i)*0.5.*(x+1)+a+sum(h(1:i-1));
end
```

```
% This function computes the corresponding index of the global
% basis functions from the vector j, containing indices of the local
% basisfunctions of the ith element.
% Needed input:
% p      the vector containing the local polynomial degrees.
% i        index of the element
% j        index of local basis function (can be vector of numbers)
% p        vector containing local polynomial degrees

function [globalnumber] = gamma_(i,j,p)
for k = 1:length(j)
    if j(k) > p(i)+1
        error('j␣has␣to␣be␣in␣{1,2,...,p(i)+1}');
    elseif j(k) == 1
        globalnumber(k) = sum(p(1:i-1))+1;
    elseif j(k) == 2
        globalnumber(k) = sum(p(1:i))+1;
    else globalnumber(k) = sum(p(1:i-1))+ j(k)-1;
    end
end
end
```

```
% This function computes the global Galerkin matrix A for
% a(u,v)= int(C(x)u'(x)v'(x)+c(x)u(x)v(x)dx) for
% the 1D hierarchic shape functions and the following
% initial values:
% p           vector with entries p(i), the local polynomial
%             degree on element i.
% h           vector with entries h(i), the length of the
%             element/intervall i
% C           density vector, containing the elemental density
%             given from the PDE
% c           given from the PDE, is anyway 0 in our computations, but
%             has been implemented, since at this time, it was not yet
%             known if it would be used.
% bd          given from the PDE, is anyway 0 in our computations,
%             since we always have homogeneous dirichlet boundary
%             conditions, but at time of implementation it has not yet
%             been known, if it would be used.
```

```
function [A] = global_galerkin(p,h,C,c,bd)
if(nargin == 4)
    bd = 0;
end
if length(p) ~= length(h)
    error('length of p and h have to be equal',p,h)
end
if length(h)==1 && p==1
     A = 1;
     return
end
B=zeros(sum(p)+1);
M_hat = local_M(max(p));
K_hat = local_K(max(p));
for i = 1:length(p)
    B(gamma_(i,1:p(i)+1,p),gamma_(i,1:p(i)+1,p)) = ...
    B(gamma_(i,1:p(i)+1,p),gamma_(i,1:p(i)+1,p)) + ...
    C(i) * 2/h(i) * K_hat(1:p(i)+1,1:p(i)+1) + ...
    c(i) * h(i)/2 * M_hat(1:p(i)+1,1:p(i)+1);
    %where the 2/h(i) and h(i)/2 come from transformation and c(i),
    %C(i) from the bilinearform a(v,w)
end
if bd == 0
    A = B(2:end-1,2:end-1);
else A = B;
end
end
```

```
% This function computes the Galerkin load vector F for
% p        vector containing local polynomial degrees
% h        vector containing h(i) lenght of i.element/intervall
% f        right hand side of DGL, function handle
% v,w      data of f_1, position of jumps, needed to get accurate
%          results for the quadrature
% quad     containing weights quad.w and quadrature nodes quad.x
%          of gauss-legendre quadrature of order p(i)+1
% a        a from omega = (a,b)
% bd       given from the PDE, is anyway 0 in our computations, since we
%          always have homogeneous dirichlet boundary conditions, but at
%          time of implementation it has not yet been known, if it would
%          be used.

function F = load_f(a,p,h,quad,f,v,w,bd)
if nargin==7
    bd=0;
end

L = zeros(sum(p)+1,1);
for k = 1:length(p)
    for i = 1:p(k)+1
    q = a+sum(h(1:k-1));
    g = @(x) f(x)*local_b(i,-1+2*(x-q)/h(k));
    L(gamma_(k,i,p),1) = L(gamma_(k,i,p),1)+...
                         integrate_on_element(k,g,a,h,v,w,quad);
    end
end
if bd == 0
    if length(h) == 1 && p(1) == 1
        F = 0; return
    end
    F = L(2:end-1);
else F = L;
end
```

```
end
```

```
% This function solves the Galerkin problem and returns the
% coefficients of u_N relatively to
% the 1d hierarcical shape functions.
% Input data:
% p      vector with entries p(i), the local polynomial
%        degree on element i.
% h      vector with entries h(i), the length of the
%        element/intervall i
% f      right hand side of DGL, function handle
% v,w    data of f_1, position of jumps, needed to get accurate
%        results for the quadrature
% quad   containing weights quad.w and quadrature nodes quad.x
%        of gauss-legendre quadrature of order p(i)+1
% a      a from omega = (a,b)
% C      density vector, containing the elemental density given from
%        the PDE
% c      given from the PDE, is anyway 0 in our computations, but has
%        been implemented, since at this time, it was not yet known,
%        if it would be used.

function [coeff_uN,A] = calc_solution(a,h,p,f,C,c,quad,v,w)
    % get global Galerkin matrix
    A = global_galerkin(p,h,C,c);
    % compute right hand side
    F = load_f(a,p,h,quad,f,v,w);
    % compute projection in terms of global shape functions
    coeff_uN = A\F;
    if length(h) == 1 && p(1)==1
        coeff_uN = [coeff_uN;0];
    else coeff_uN = [0;coeff_uN;0];
    end
end
```

```
% This function evaluates the Legendre polynomial of Order p at the entries
% of the vector x.

function y = legendre_pol(p,x)
if p < 0
  error('p␣has␣to␣be␣>0',p)
elseif p == 0
  y = ones(size(x));
elseif p == 1
  y = x;
elseif p >= 2
  y1 = ones(size(x));
  y0 = x;
  for i=2:p
    y = ((2*i-1)/i)*x.*y0 - ((i-1)/i)*y1;
    y1 = y0;
    y0 = y;
  end
end

end
```

```
% This function evaluates the first derivative of the number_th
% 1d hierarchical shape function in x.
% Needed input:
% number  index of the hierarchical shape function, the
%                derivative shoud be evaluated of.
```

```
% x   evaluation point/ vector

function y = first_derivative_localbasis(number,x)

% initialization
y = zeros(size(x));

% Evaluate the derivative of the local shape function with index number % at the vector x
if number == 1
  y = -1/2*ones(length(x),1);
elseif number == 2
  y = 1/2*ones(length(x),1);
else
      y = sqrt((2*number-3)/2)*legendre_pol(number-2,x);
end

end
```

```
%This function calculates the 2nd derivative of the lokal 1D hierarchic
%shape functions by using the identity for derivatives of legendre
%polynomials and the fact that we can write every lokal basis function
%in terms of sums of legendre polynomials.

%The function returns the vector y, containing the values of the
%derivative of b_number evaluated in x.

%For this evaluation it does not matter what boundary condition we have,
%since the 2nd derivative of the hat functions is everywhere 0.
%Input arguments:
%number      number of local basis function
%x           vector in which the 2nd derivative of u should be evaluated
%            in.

function y = second_derivative_localbasis(number,x)

%first calculate the 2nd derivative of the local shape functions with index
%number at the vector x

%Initialization
y = zeros(size(x));

if number ~= 1 && number ~= 2
    y = sqrt((2*number-3)/2)./(x.^2-1).*(number-2).*...
        (x.*legendre_pol(number-2,x)-legendre_pol(number-3,x));
end
end
```

```
% This function evaluates the Galerkin solution uN in x (can be vector)
% using the coordinates u relatively to 1D hierarchic shape functions.
% Needed are:
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) lenght of i.element/intervall
% u          the coordinates relatively to 1D hierarchic shape functions
% x          vector containing points on which uN has to be evaluated
% a          a from omega =(a,b)

function y = evaluate_uN(p,h,u,x,a)
y = zeros(size(x));
for i = 1:length(p)
    for j = 1:p(i)+1
        l = gamma_(i,j,p);
        y = y + u(l).*(sum(h(1:i-1))<x).*...
```

```
                (x<=sum(h(1:i))).*local_b(j,-1+2*(x-a-sum(h(1:i-1)))/h(i));
    end
end
end
```

```
% This function evaluates first derivative of the Galerkin solution u_N
% in x (can be vector)
% using the coordinates u relatively to 1D hierarchic shape functions.
% Needed are:
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) lenght of i.element/intervall
% u          the coordinates relatively to 1D hierarchic shape functions
% x          vector containing points on which uN has to be evaluated
% a          a from omega =(a,b)

function y = evaluate_uN_derivative(p,h,u,x,a)
y = zeros(size(x));
for i = 1:length(p)
    for j = 1:p(i)+1
        l = gamma_(i,j,p);
        y = y + u(l).*(sum(h(1:i-1))<x).*...
            (x<=sum(h(1:i))).*...
            first_derivative_localbasis(j,-1+2*(x-a-sum(h(1:i-1)))/h(i))*2/h(i);
    end
end
end
```

```
% evaluate_uN_derivative(p,h,u,x,a)
% This function evaluates the Galerkin solution uN in a gridpoint x
% taking the limit from the element i lying nearby x,
% using the coordinates u relatively to 1D hierarchic shape functions.
% Needed are:
% i          index of a neighbouring element of node x from which the limit of the
%            galerkin solution u_N to x is taken
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) lenght of i.element/intervall
% u          the coordinates relatively to 1D hierarchic shape functions
% x          gridpoint on which the limit of uN on the element i has to be evaluated
% a          a from omega =(a,b)

function y = evaluate_uN_derivative_limit(i,p,h,u,x,a)
y = zeros(size(x));
    for j = 1:p(i)+1
        l = gamma_(i,j,p);
        y = y + u(l).*...
            first_derivative_localbasis(j,-1+2*(x-a-sum(h(1:i-1)))/h(i))*2/h(i);
    end
end
```

```
% This function evaluates the second derivative of the Galerkin solution
% u_N in x (can be vector)
% using the coordinates u relatively to 1D hierarchic shape functions.
% Needed are:
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) lenght of i.element/intervall
% u          the coordinates relatively to 1D hierarchic shape functions
% x          vector containing points on which uN has to be evaluated
% a          a from omega =(a,b)

function y = evaluate_uN_second_derivative(p,h,u,x,a)
y = zeros(size(x));
for i = 1:length(p)
```

```
    for j = 1:p(i)+1
        l = gamma_(i,j,p);
        y = y + u(l).*(sum(h(1:i-1))<x).*...
            (x<=sum(h(1:i))).*...
            second_derivative_localbasis(j,-1+2*(x-a-sum(h(1:i-1)))/h(i))...
            *(2/h(i))^2;
    end
end
end
```

```
% This function integrates the function handle g over the k_th element
% of the mesh starting in a and being defined by the vector h.
% Moreover in elements where v and w are contained, the integration is
% seperately calculated on (x_{k-1},v) and (v,x_k) respectively
% (x_{k-1},w) and (w,x_k) or (x_{k-1},v) and (v,w) and (w,x_k).
% This separation is necessary for the function f_1, since there are
% jumps in v and w. If we would not do this separation, we would need high
% order quadrature to get good enough integration results.
% Needed input:
% k      index of element
% g      function handle of function which has to be integrated over
% h      vector containing h(i) length of i-th element/interval
% a      a from omega = (a,b)
% v,w    data of f_1, position of jumps, needed to get accurate
%        results for the quadrature
% quad   containing weights quad.w and quadrature nodes quad.x
%        of gauss-legendre quadrature of order p(i)+1

function y = integrate_on_element(k,g,a,h,v,w,quad)

y=0;

% testing if v and w lie in k.element
if a+sum(h(1:k-1))+eps < v && a+sum(h(1:k))-eps >w

%integrate over partitions of k.element by v,w
    q = a+sum(h(1:k-1));
    z = [v-q;w-v;q+h(k)-w];
    for j = 1:3
        for m = 1:length(quad.x)
            o = phi_inv(j,quad.x(m),q,z);
            y = y+g(o)*quad.w(m)*z(j)/2;
        end
    end

% testing if only v lies in k.element
elseif a+sum(h(1:k-1))+eps < v && a+sum(h(1:k))-eps > v

%integrate over partitions of k.element by v
    q = a+sum(h(1:k-1));
    z = [v-q;q+h(k)-v];
    for j = 1:2
        for m = 1:length(quad.x)
            o = phi_inv(j,quad.x(m),q,z);
            y = y+g(o)*quad.w(m)*z(j)/2;
        end
    end

% testing if only w lies in k.element
elseif a+sum(h(1:k-1))+eps < w && a+sum(h(1:k))-eps > w

%integrate over partitions of k.element by w
    q = a+sum(h(1:k-1));
```

```
    z = [w-q;q+h(k)-w];
    for j = 1:2
        for m = 1:length(quad.x)
            o = phi_inv(j,quad.x(m),q,z);
            y = y+g(o)*quad.w(m)*z(j)/2;
        end
    end
else
    for m=1:length(quad.x)
        o=phi_inv(k,quad.x(m),a,h);
        y=y+g(o)*quad.w(m)*h(k)/2;
    end
end
end
```

```
% This function evaluates the exact error in energy norm,
% taking the difference of the derivative of the exact solution
% and the Galerkin solution and calculating its energy norm
% (high order quadrature is used).
% Input data:
% u_diff    function handle of the derivative of the exact solution
% p         vector containing the elemental polynomial degree
% h         vector containing h(i) length of i-th element/interval
% u_N       the coordinates relatively to 1D hierarchic shape functions
% f         right hand side of DGL, function handle
% v,w       data of f_1, position of jumps, needed to get accurate
%           results for the quadrature
% quad      containing weights quad.w and quadrature nodes quad.x
%           of gauss-legendre quadrature of order p(i)+1
% a         a from omega = (a,b)
% C         density vector, containing the elemental density given
%           from the PDE

function err = exact_error_energy(u_diff,u_N,a,h,p,quad,C,v,w)
err = 0;
g = @(x) (u_diff(x)-evaluate_uN_derivative(p,h,u_N,x,a))^2;
for k = 1:length(h)
    g_loc = @(x) C(k)*g(x);
    err = err+integrate_on_element(k,g_loc,a,h,v,w,quad);
end
err = sqrt(err);
end
```

## 8.3 Error estimators

```
% This function calculates the value of the L2(K_k)-projection of
% the function f at point x.
% Input:
% x       evaluation  point
% k       number of element, in which x is contained
% p       vector containing the elemental polynomial degree
% h       vector containing h(i) length of i-th element/interval
% f       right hand side of DGL, function handle
% v,w     data of f_1, position of jumps, needed to get accurate
%         results for the quadrature
% quad    containing weights quad.w and quadrature nodes quad.x
%         of gauss-legendre quadrature of order p(i)+1
% a       a from omega = (a,b)

y = evaluate_proj(k,f,h,p,a,v,w,quad,x)
```

```
17 % initialization
   y = zeros(size(x));
19
   % calculates the Legendre coefficients
21 a_proj = orth_proj_K(k,f,h,p,a,v,w,quad);

23 % evaluation in point x
   for i = 1:length(a_proj)
25 y = y+a_proj(i)*legendre_pol(i-1,-1+2*(x-a-sum(h(1:k-1)))/h(k))...
       *sqrt(2/h(k));
27 end
   end
```

```
   % This function calculates the first p(k) coefficientsis of the legendre series
 2 % interpolating f.
   % They are needed to calculate the L2(K_k)-orthogonal projection of f
 4 % onto P_{p(k)-1}(K_k), since it is calculated interpolating f by the first
   % p Legendre polynomials.
 6 % Needed input:
   % k          element, on which projection should be done
 8 % p          vector containing the elemental polynomial degree
   % h          vector containing h(i) length of i-th element/interval
10 % f          right hand side of DGL, function handle
   % v,w        data of f_1, position of jumps, needed to get accurate
12 %            results for the quadrature
   % quad       containing weights quad.w and quadrature nodes quad.x
14 %            of gauss-legendre quadrature of order p(i)+1
   % a          a from omega = (a,b)
16
   function a_proj = orth_proj_K(k,f,h,p,a,v,w,quad)
18
   % initialization
20 a_proj=zeros(p(k),1);

22 % calculation
   for i = 0:p(k)-1
24     g = @(x)(2*i+1)*0.5*sqrt(2/h(k))*f(x)*...
           legendre_pol(i,-1+2*(x-a-sum(h(1:k-1)))/h(k));
26     a_proj(i+1) = integrate_on_element(k,g,a,h,v,w,quad);
   end
28 end
```

### 8.3.1 Residual error estimator

```
   %This function computes the eps for the residual error estimator
 2 %Needed intput:
   % p          vector containing the elemental polynomial degree
 4 % h          vector containing h(i) length of i-th element/interval
   % u_N        the coordinates relatively to 1D hierarchic shape
 6 %            functions
   % f          right hand side of DGL, function handle
 8 % v,w        data of f_1, position of jumps, needed to get accurate
   %            results for the quadrature
10 % quad       containing weights quad.w and quadrature nodes quad.x
   %            of gauss-legendre quadrature of order p(i)+1
12 % a          a from omega = (a,b)
   % C          density vector, containing the elemental density given from the PDE
14
   function [norm_res,eps] = error_indicator(a,u_N,p,h,f,quad,C,v,w)
16
   % Initialization
```

```
eps=zeros(size(p));
norm_res=zeros(size(p));

for k = 1:length(h)
        q = a+sum(h(1:k-1));
        g = @(x) zeros(size(x));
        for i = 1:p(k)+1
            g = @(x) g(x)+u_N(gamma_(k,i,p))*...
                        second_derivative_localbasis(i,-1+2*(x-q)/h(k));
        end

        % here we use the L2-projection of f to evaluate the error
        % estimator.
        g = @(x) C(k)*(C(k)*g(x)*4/h(k)^2+...
    evaluate_proj(k,f,h,2*p+5,a,v,w,quad,x))^2.*omeg(k,x,h);
        norm_res(k) = norm_res(k)+integrate_on_element(k,g,a,h,v,w,quad);
end
eps = 1./(C.*p.*(p+1)).* norm_res;
end
```

### 8.3.2 Equilibrated error estimator

```
%This function computes the eps for the equilibrated error estimator
%Needed input:
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) length of i-th element/interval
% u_N        the coordinates relatively to 1D hierarchic shape
%            functions
% f          right hand side of DGL, function handle
% v,w        data of f_1, position of jumps, needed to get accurate
%            results for the quadrature
% quad       containing weights quad.w and quadrature nodes quad.x
%            of gauss-legendre quadrature of order p(i)+1
% a          a from omega = (a,b)
% C          density vector, containing the elemental density given from the PDE

function epsilon = equilibrated_error_indicator(f,u_N,a,h,p,C,v,w,quad)

% initialization
epsilon = zeros(size(h));
xk_1 = a;
xk = xk_1+h(1);
resk_1 = @(x) zeros(size(x));

% calculating projection of f onto K_1
proj_f = @(x) evaluate_proj(1,f,h,p+1,a,v,w,quad,x);

% calculating estimator by integrating the residual on every element
res = @(x) evaluate_uN_second_derivative_local(1,p,h,u_N,x,a);
for k = 1:length(h)-1
    res_T = @(x) (C(k)*res(x)+proj_f(x))*...
        local_b(2,-1+2*(x-a-sum(h(1:k-1)))/h(k));
    q = a+sum(h(1:k-1));
    p1 = @(x) integrate_on_element(1,res_T,q,x-q,v,w,quad);
    resk = @(x) (resk_1(x)+p1(x))^2;

% calculating L2-norm of the local error estimate consisting of fluxes
% sigma_{k-1} and sigma_k
    intk = integrate_on_element(k,resk,a,h,v,w,quad);
    epsilon(k) = 1/C(k)*intk;

% initialization of data on new element
```

```
    xk_1 = xk;
    xk = xk+h(k+1);

% calculating the new residual and integrating it
    res = @(x) evaluate_uN_second_derivative_local(k+1,p,h,u_N,x,a);
    proj_f = @(x) evaluate_proj(k+1,f,h,p,a,v,w,quad,x);
    res_T = @(x) (C(k+1)*res(x)+proj_f(x))*...
        local_b(1,-1+2*(x-a-sum(h(1:k)))/h(k+1));
    q = a+sum(h(1:k));
    p2 = @(x) integrate_on_element(1,res_T,q,x-q,v,w,quad);

% evaluating the constants
    r_vertex = C(k+1)*evaluate_uN_derivative_limit(k+1,p,h,u_N,xk_1,a)-...
        C(k)*evaluate_uN_derivative_limit(k,p,h,u_N,xk_1,a);
    resk_1 = @(x) p2(x)+p1(xk_1)+r_vertex;
end

% calculates the error estimation for the last element
resk = @(x) resk_1(x)^2;
intk = integrate_on_element(length(h),resk,a,h,v,w,quad);
epsilon(end) = 1/C(end)*intk;
end
```

## 8.4 Marking strategies

```
% This function decides weather an element has to be refined or not by the
% maximum strategy. It returns an vector of length of number of elements,
% containing 1 for elements to be refined, 0 otherwise.
% Input data:
% theta       weight theta E (0,1) (theta =1: no element would be marked
%                                   theta =0: all elements would be marked)
% eps         local error indikator, ie. vector containing the estimated
%             local element errors

function mark=marker_max(theta,eps)

% initialization
mark = zeros(size(eps));

% checking if eps(k) is to big i.e. has to be refined (has to => mark(k)=1)
for k = 1:length(eps)
    if eps(k) >= (1-theta)*max(eps)
        mark(k) = 1;
    end
end
end
```

```
% This function performs the fixed energy fraction marking strategy
% using an knapsack algorithm.
% Input data:
% theta   parameter in (0,1) ( 0= no element is going to be marked;
%             1= all elements are going to be marked}
% nu    parameter in (0,1), regulates number of iterations
% eps   vector containing local estimated error

function mark = knapsack2(theta,nu,eps)

% initialization
s= 0;
tau = 1;
max_eps=max(eps);
mark = zeros(size(eps));
```

```
eps_ges=sum(eps);

% knapsack algorithm
while s < theta^2*eps_ges
    tau=tau-nu;
    for k=1:length(mark)
        if mark(k)==0
            if eps(k) > tau*max_eps
                mark(k) = 1;
                s = s + eps(k);
            end
        end
    end
end
end
```

```
% This function performs the weighted fixed energy fraction marking strategy
% using a knapsack algorithm. The algorithm is needed for the hp-decider which
% solves the minimization problem.
% Input data:
% theta     parameter in (0,1) ( 0= no element is going to be marked;
%                                1= all elements are going to be marked}
% nu        parameter in (0,1), regulates number of iterations
% eps       vector containing local estimated errorfunction
% frac      weights for the weighted fixed energy fraction.
% p         vector containing the elemental polynomial degree

mark = knapsack(theta,nu,eps,frac,p)

% initialization
s = 0;
tau = 1;
e = eps.*frac.^2;
max_e = max(e);
mark = zeros(size(p));
eps_ges = sum(eps);

% knapsack algorithm
while s < theta^2*eps_ges
    tau = tau-nu;
    for k = 1:length(mark)
        if mark(k) == 0
            if e(k) > tau*max_e
                mark(k) = 1;
                s = s + e(k);
            end
        end
    end
end
end
```

## 8.5 *hp*-decider

### 8.5.1 Estimation of analyticity

```
% This function returns the marking matrix (first column for h-, second for
% p-refinement) generated by the hp-decider estimating the regularity by extending the
% solution by legendre series
%(convergence radius >= theta --> smooth enough --> p-refinement
% otherwise h-refinement)
% Needed data:
```

```
% p          vector containing the elemental polynomial degree
% mark       vector of the same length as p with entry 1 if element sould be refined
%                                                    0 otherwise
% h          vector containing h(i) length of i-th element/interval
% u_N        the coordinates relatively to 1D hierarchic shape functions
% theta      parameter in (0,1)

function h_or_p = mark_h_p_analyt_ext(mark,theta,u_N,h,p)

% initialization
h_or_p=zeros(length(mark),2);

% calculation
for k=1:length(mark)
    if mark(k)==1
        theta_el=inv_bernsteinrad(u_N,h,p,k);
        if theta_el<=theta
            h_or_p(k,2)=1;
        else h_or_p(k,1)=1;
        end
    end
end
end
```

```
% This function estimates the inverse of the local bernstein radius of u_N
% on the k_th element. The algoritm is needed for the hp-decider
% estimating analyticity.
% Needed input data:
% k          index of element on which the inverse of the local bernstein
%            radius of u_N should be calculated
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) length of i-th element/interval
% u_N        the coordinates of the solution of the BVP
%            relatively to 1D hierarchic shape functions

function theta_el = inv_bernsteinrad(u_N,h,p,k)
a = zeros(p(k)+1,1);
l = sqrt(h(k)/2);

a(1) = l*0.5*(u_N(gamma_(k,1,p))+u_N(gamma_(k,2,p)));
a(2) = l*0.5*(u_N(gamma_(k,2,p))-u_N(gamma_(k,1,p)));
for i = 2:p(k)
    a(i+1) = l*1/sqrt(2*(2*i-1))*u_N(gamma_(k,i+1,p));
    a(i-1) = a(i-1)-l*1/sqrt(2*(2*i-1))*u_N(gamma_(k,i+1,p));
end
y = abs(log(abs(a)));
num = 0:p(k);
m = 6*(2*num*y-p(k)*sum(y))/(p(k)+1)/((p(k)+1)^2-1);
theta_el = exp(-m);
end
```

### 8.5.2 Minimization

```
% This function is marker and decider at one time. It stands for the
% hp-decider which solves the minimization problem and returns a
% vector mark of the same length as h, which contains 1, if the
% element has to be refined, 0, otherwise.
% I is a matrix, of size length(h)xo, which contains 1 in (k,j) if
% the k-th. element should be refined by pattern j.
% Input data:
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) length of i-th element/interval
```

```
% u_N        the coordinates relatively to 1D hierarchic shape
%            functions
% f          right hand side of DGL, function handle
% v,w        data of f_1, position of jumps, needed to get accurate
%            results for the quadrature
% quad       containing weights quad.w and quadrature nodes quad.x
%            of gauss-legendre quadrature of order p(i)+1
% a          a from omega = (a,b)
% C          density vector, containing the elemental density given
%            from the PDE
% c          given from the PDE, is anyway 0 in our computations, but
%            at time of implementation it has not yet been known, if
%            it would be used.
% theta      parameter for marking elements in (0,1)
% nu         parameter for marking elements, needed to use knapsack.m
%            algorithm
% o          indicates that the patterns 1-o can be chosen to do the
%            refinement step
% eps        vector containing the local estimated error in energy
%            norm
% norm_res   vector containing the elemental L2-norm of the residual,
%            calculated in algorithm error_indicator.m
% bd         given from the PDE, is anyway 0 in our computations,
%            since we always have homogeneous dirichlet boundary
%            conditions, but at time of implementation it has not yet
%            been known, if it would be used.

function [mark,I] = marker_energy_loc(theta,eps,norm_res,nu,a,p,h,C,c,f,...
  u_N,quad,v,w,o,bd)
if nargin == 15
    bd = 0;
end
if o > 5
    error('o␣has␣to␣be␣<=5',o)
end

% initialization
normelres = norm_res;
beta_ = zeros(length(p),o);
omega = zeros(length(p),o);
frac = zeros(size(p));
I = zeros(size(p));
I_max = zeros(size(p));
minq = zeros(size(p));
maxq = zeros(size(p));

% loop: iteration over the patterns 1 to o
for n = 1:o

% initialization, setting to zero in every step
    normz2 = zeros(size(p));

    for i = 1:length(p)

% the local bvp from which we get beta_i^n, the index of efficiency of pattern n
% on element i is solved and the L2- norm of the derivative is stored in norm2_z(i).
        [z,p_loc,h_loc] = solve_loc(i,n,h,p,a,u_N,C,c,quad,f,v,w);
        q = a+sum(h(1:i-1));
        for j = 1:length(h_loc)
            g = @(x) zeros(size(x));
            for m = 1:p_loc(j)+1
                g = @(x) g(x)+z(gamma_(j,m,p_loc))*...
```

```
                    first_derivative_localbasis(m,-1+2*(x-q-sum(h_loc(1:j-1)))...
      /h_loc(j))*2/h_loc(j);
            end
            g = @(x)g(x).^2;
            normz2(i) = normz2(i)+...
                integrate_on_element(j,g,q,h_loc,v,w,quad);
        end
    end

% the local degrees of freedom are stored in omega
    if n==1 || n==3 || n==4
        omega(:,n)=2*p+1;
    elseif n==2
        omega(:,n)=p+2;
    elseif n==5
        omega(:,n)=p+3;
    end

% the index beta is computed and stored
    beta_(:,n) = p.*sqrt(normz2)./sqrt(normelres);
end

% the minimization problem is solved and different conditions are checked.
% This is done to get the constraint fulfilled if possible (cf. theory).
for k = 1:length(p)
    [frac(k),I(k)] = min(omega(k,:)./beta_(k,:));
    frac(k) = beta_(k,I(k));
    minq(k) = min(beta_(k,:));
    [maxq(k),I_max(k)] = max(beta_(k,:));
    %[max_val(k),I_max(k)] = max(omega(k,:)./beta_(k,:));
end
minminq = min(minq);
minmaxq = min(maxq);
oldmin = 10;
for k = 1:o
    for m = 1:o
        if min(omega(:,k)/omega(:,m)) <= oldmin
            oldmin = min(omega(:,k)/omega(:,m));
        end
    end
end
oldmin = oldmin-2.2204e-016;
if minminq > 0
    if theta >= minminq
    theta = minminq/2;
    end
elseif oldmin > 0 && minmaxq > 0
      if theta >= oldmin*minmaxq
          theta = oldmin*minmaxq*1/2;
      end
elseif minmaxq > 0
    if theta >= minmaxq
        theta = minmaxq/2;
    end
    I = I_max;
    frac = maxq;
else mark = ones(size(h)); return;
end
mark = knapsack(theta,nu,eps,frac,p);
end
```

```
% omeg is a weight function, used in our weighted fixed energy fraction marker.
% Needed input data:
```

```
% k       number of element
% x       value in which we want to evaluate w
% h       vector containing elemental/intervall sizes

function y = omeg(k,x,h)
y = (sum(h(1:k))-x).*(x-sum(h(1:k-1)))*(x<=sum(h(1:k))).*(x>sum(h(1:k-1)));
end
```

```
% This function solves a local BVP for the hp-decider solving a minimization problem.
% Is needed to get the values beta_i^n, indicating the efficiency of the refinement
% step on element k by pattern n.
%Needed input:
% k          index of element on which the local problem should be solved
% n          index of pattern for which the local problem should be solved
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) length of i-th element/interval
% u_N        the coordinates relatively to 1D hierarchic shape functions
% f          right hand side of DGL, function handle
% v,w        data of f_1, position of jumps, needed to get accurate results
%            for the quadrature
% quad       containing weights quad.w and quadrature nodes quad.x
%            of gauss-legendre quadrature of order p(i)+1
% a          a from omega = (a,b)
% C          density vector, containing the elemental density given from the PDE
% c          given from the PDE, is anyway 0 in our computations, but has been
%            implemented, since at this time, it was not yet known, if it would be used.

function [z,p_loc,h_loc]=solve_loc(k,n,h,p,a,u_N,C,c,quad,f,v,w)

if n == 1 || n == 3 || n == 4
    if n == 1
        h_loc = 0.5*[h(k);h(k)];
    elseif n == 3
        h_loc = [0.15*h(k);0.85*h(k)];
    else h_loc = [0.85*h(k);0.15*h(k)];
    end
    p_loc = [p(k);p(k)];
    A = global_galerkin(p_loc,h_loc,C(k)*ones(2,1),c(k)*ones(2,1));
    f = load_f_res(k,p,h,a,u_N,h_loc,p_loc,C,quad,f,v,w);
else
    if n == 2
        p_loc = p(k)+1;
    else p_loc = p(k)+2;
    end
    h_loc = h(k);
    A = global_galerkin(p_loc,h_loc,C(k),c(k));
    f = load_f_res(k,p,h,a,u_N,h_loc,p_loc,C,quad,f,v,w);
end
z = A\f;
z = [0;z;0];
end
```

```
% This function computes the load vector F needed for the hp-decider
% solving the minimization problem.
% Input data:
% k          element the load vector has to be evaluated on
% p          vector containing the elemental polynomial degree
% h          vector containing h(i) length of i-th element/interval
% p_loc      vector containing the polynomial degrees of the locally refined element k
% h_loc      vector containing the the lengths of the new elements
%            of the locally refined element k
% u_N        the coordinates relatively to 1D hierarchic shape functions
```

```
% f          right hand side of DGL, function handle
% v,w        data of f_1, position of jumps, needed to get accurate results
%            for the quadrature
% quad       containing weights quad.w and quadrature nodes quad.x
%            of gauss-legendre quadrature of order p(i)+1
% a          a from omega = (a,b)
% C          density vector, containing the elemental density given from the PDE
% bd         given from the PDE, is anyway 0 in our computations,
%            since we always have homogeneous dirichlet boundary
%            conditions, but at time of implementation it has not yet
%            been known, if it would be used.

function F = load_f_res(k,p,h,a,u_N,h_loc,p_loc,C,quad,f,v,w,bd)
if nargin == 12
    bd = 0;
end
L = zeros(sum(p_loc)+1,1);
q = a+sum(h(1:k-1));
g = @(x) zeros(size(x));
for i = 1:p(k)+1
    g = @(x) g(x)+u_N(gamma_(k,i,p))*...
        second_derivative_localbasis(i,-1+2*(x-q)/h(k));
end
g = @(x) C(k)*g(x)*4/h(k)^2+f(x);
for j = 1:length(p_loc)
    for i=1:p_loc(j)+1
        g_loc = @(x) g(x).*local_b(i,-1+2*(x-q-sum(h_loc(1:j-1)))/h_loc(j));
        L(gamma_(j,i,p_loc),1) = L(gamma_(j,i,p_loc),1)+...
            integrate_on_element(j,g_loc,q,h_loc,v,w,quad);
    end
end
if bd == 0
    if length(h) == 1 && p(1) == 1
        F=0; return
    end
    F=L(2:end-1);
else F=L;
end
end
```

Moreover c.f. 8.4 to see the implementation of the marker included in the mean implementation.

## 8.6 *hp*-refinement step

```
% This function does the p-refinement, i.e. increases the local polynomial
% degree on marked elements by s.
% Input data:
% p          vector containing the elemental polynomial degree
% mark_p     vector of the same length as p with entry 1 if element
%            should be p-refined, 0 otherwise

function [p_new] = p_increase(p,mark_p,s)

for k = 1:length(p)
    if mark_p(k) == 1
        p(k)=p(k)+s;
    end
end
p_new = p;
end
```

```
% This function does the h-refinement step by enlarging the vector h by 1 and
% writing the new elemental lengths into h.
% Further, the vectors p, c, and C have to actualized.
% Input data:
% p           vector with entries p(i), the local polynomial
%             degree on element i.
% h           vector with entries h(i), the length of the
%             element/intervall i
% C           density vector, containing the elemental density given from the PDE
% c           given from the PDE, is anyway 0 in our computations, but
%             has been implemented, since at this time, it was not yet
%             known, if it would be used.
% mark_h      matrix of size (length(p),length(sigma)) with entry
%             1 in mark_h(i,j) if element it should be h-refined with
%             weight sigma(j), 0 in mark_h(i,j) otherwise
% sigma       vector containing weights in (0,1)
%             for grading the bisection: h'= sigma*h, h'' = (1-sigma)*h.

function [h_new,p_new,c_new,C_new] = h_graded_bisection(h,p,c,C,mark_h,sigma)
if nargin == 5
    sigma = 0.15;
end
n = length(sigma);
for k = 1:n
if sigma(n) >= 1 || sigma(n) <= 0
    error('the␣entries␣of␣sigma␣have␣to␣be␣chosen␣in␣(0,1)',sigma)
end
end
h_new = zeros(sum(sum(mark_h))+length(h),1);
p_new = zeros(size(h_new));
c_new = zeros(size(p_new));
C_new = zeros(size(p_new));
k=1;
s=0;
while k <= length(h)
    h_new(k+s) = h(k);
    c_new(k+s) = c(k);
    C_new(k+s) = C(k);
    p_new(k+s) = p(k);
    for i=1:n
        if(mark_h(k,i) == 1)
            h_new(k+s) = h(k)*sigma(i);
            h_new(k+s+1) = h(k)*(1-sigma(i));
            c_new(k+s+1) = c(k);
            C_new(k+s+1) = C(k);
            p_new(k+s+1) = p(k);
            s = s+1;
        end
    end
    k = k+1;
end
end
```

# 9 Appendix

## 9.1 Properties of the Legendre polynomials

The *Legendre polynomials* are recursively defined in the following way:

$$L_0(x) = 1 \tag{168}$$
$$L_1(x) = x \tag{169}$$
$$L_{n+1}(x) = \frac{2n+1}{n+1} x L_n(x) - \frac{n}{n+1} L_{n-1}(x) \; \forall n \in \mathbb{N} \setminus \{0, 1\} \tag{170}$$

Some important properties of the Legendre polynomials are:
orhtogonality:

$$\int_{-1}^{1} L_n(x) L_m(x) dx = \begin{cases} \frac{2}{2n+1} & \text{if } n = m \\ 0 & \text{otherwise} \end{cases} \tag{171}$$

properties of differentiation:

$$L_n(x) = \frac{L'_{n+1}(x) - L'_{n-1}(x)}{2n+1} \tag{172}$$

furthermore the Legendre polynomials fulfill the following differential equation:

$$(x^2 - 1) L'_n(x) = n(x L_n(x) - L_{n-1}(x)) \tag{173}$$
$$= n(n+1) \int_{-1}^{x} L_n(t)\, dt \tag{174}$$

For all the formulas stated in this section cf. [8, APPENDIX C.2].

# References


[1] D. Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer, 2007.

[2] Dietrich Braess, Veronika Pillwein, and Joachim Schberl. Equilibrated residual error estimates are p-robust. *Computer Methods in Applied Mechanics and Engineering*, 198(13-14):1189 – 1197, 2009. HOFEM07 - International Workshop on High-Order Finite Element Methods, 2007.

[3] W. Dörfler and V. Heuveline. Convergence of an adaptive hp finite element strategy in one space dimension. *Applied numerical mathematics*, 57(10):1108–1124, 2007.

[4] C. Gittelson and D. Kressner. *NAPDE Exercises: Programming Project 2*. ETH Zurich, 2009.

[5] G.H. Golub and J.H. Welsch. *Calculation of Gauss quadrature rules*. Stanford University of California, 1967.

[6] R. Hiptmair and C. Schwab. *Lecture Notes: Numerical Methods for Elliptic and Parabolic Boundary Value Problems*. ETH Zurich, 2008.

[7] P. Houston and E. Süli. A note on the design of hp-adaptive finite element methods for elliptic partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 194(2-5):229 – 243, 2005. Selected papers from the 11th Conference on The Mathematics of Finite Elements and Applications.

[8] C. Schwab. *p-and hp-finite element methods: Theory and applications in solid and fluid mechanics*. Oxford University Press, USA, 1998.

[9] Z.-C. Shi. *Recent advances in adaptive computation*. Zhejiang University, 2004.