Duality based error estimation for electrostatic force computation

Bachelor's Thesis

Author:
Simon Pintarelli

Supervisor:
Prof. Ralf Hiptmair

July 2010

Contents

Chapter 1

Introduction

The electric field E can be written as $E = -\nabla u$, where is u is the electrostatic potential.

$$\nabla \cdot E = -\nabla \cdot \nabla u = -\Delta u = \frac{\rho(x)}{\epsilon_0} \quad \text{(first Gauss' law)} \tag{1.1}$$

Since $\rho(x) = 0$, for $x \in \Omega$ we arrive at

$$-\Delta u = 0 \qquad x \in \Omega \tag{1.2}$$

$$u_{|\Gamma_i} = \begin{cases} g_1 & x \text{ on } \Gamma_1 \\ g_2 & x \text{ on } \Gamma_2 \end{cases}. \tag{1.3}$$

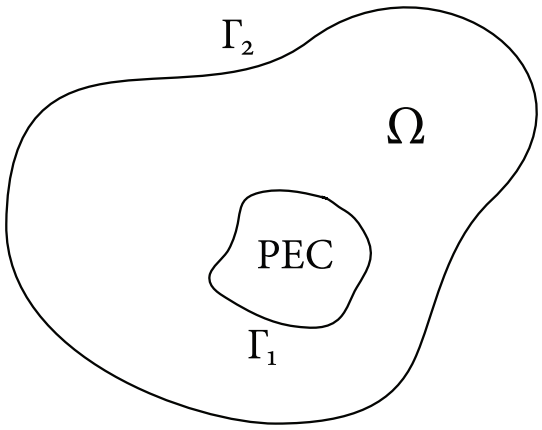


Figure 1.1: PEC (perfect electric conductor) enclosed by a potential g_2 on Γ_2

1.1 Finite element approximation

Multiplication (eq. 1.2) by $v \in H_0^1(\Omega)$ and integration by parts gives

$$\int_\Omega \nabla u \cdot \nabla v \ \mathrm{d}x =: \mathrm{a}(u, v) = 0,$$

and the finite element formulation is then to find $u \in H^1(\Omega)$, such that

$$\mathrm{a}(u, v) = 0, \quad \forall v \in H_0^1(\Omega). \tag{1.4}$$

Existence and uniqueness The symmetric bilinear a form is bounded and V-elliptic. The right hand side $f \equiv 0$ is in $L_2(\Omega)$. Thus it follows from the Lax-Milgram lemma that (eq. 1.4) has a unique solution. [3]

1.2 Force computation

Maxwell stress tensor T

$$T = \nabla u \cdot \nabla u^T - \frac{1}{2}\|\nabla u\|^2 \mathbf{I}$$

The force acting on the inner body is given by the integral of the stress tensor T over Γ_1.

$$\begin{aligned} F(u) &= \int_{\Gamma_1} T \cdot n \, \mathrm{d}\sigma \\ &= \int_{\Gamma_1} (\nabla u^T \cdot n)\nabla u - \frac{1}{2}(\nabla u^T \nabla u) n \, \mathrm{d}\sigma \end{aligned}$$

For later use it is shown that the divergence of T vanishes

$$\begin{aligned} (\nabla \cdot T)_j &= \sum_{i=1}^{3} \frac{\partial}{\partial x_i} \left(\frac{\partial u}{\partial x_j}\frac{\partial u}{\partial x_i} - \frac{1}{2}\sum_{k=1}^{3}\left(\frac{\partial u}{\partial x_k}\right)^2 \delta_{ij} \right) \\ &= \sum_{i=1}^{3} \left(\frac{\partial u}{\partial x_j}\frac{\partial^2 u}{\partial x_i^2} + \frac{\partial^2 u}{\partial x_i \partial x_j}\frac{\partial u}{\partial x_i} \right) - \frac{1}{2}\frac{\partial}{\partial x_j}\sum_{k=1}^{3}\left(\frac{\partial u}{\partial x_k}\right)^2 \\ &= \frac{\partial u}{\partial x_j} \underbrace{\sum_{i=1}^{3}\frac{\partial^2 u}{\partial^2 x_i}}_{=0} + \sum_{i=1}^{3}\frac{\partial^2 u}{\partial x_i \partial x_j}\frac{\partial u}{\partial x_i} - \frac{1}{2}\sum_{i=1}^{3}\frac{\partial}{\partial x_j}\left(\frac{\partial u}{\partial x_i}\right)^2 \\ &= \sum_{i=1}^{3}\frac{\partial^2 u}{\partial x_i \partial x_j}\frac{\partial u}{\partial x_i} - \frac{\partial u}{\partial x_i}\frac{\partial^2 u}{\partial x_i \partial x_j} = 0, \end{aligned}$$

therefore we find by applying Gauss' theorem and with insertion of a cutoff function Ψ

$$\begin{aligned} F &= \int_{\Gamma_1} T \cdot n \, d\sigma = \int_{\Gamma_1} T \cdot n\Psi \, d\sigma \\ &= \int_{\Gamma_1} div(T\Psi) \, d\sigma \\ &= \int_{\Omega_1} divT \cdot \Psi + T \cdot \nabla\Psi \, dx \\ &= \int_{\Omega_1} T(u) \cdot \nabla\Psi \, dx = F(u). \end{aligned}$$

This is true for $\Psi_{|\Gamma_1} = 1$, $\Psi_{|\Gamma_2} = 0$ and $\Psi \in H^1$, $\nabla\Psi \in L^\infty$, i.e Ψ lives in the space $W^{1,\infty}$.

The above is known as the eggshell method [5]. One of its advantages is that the shape of the eggshell is actually free and the shell needs not to be in contact with the object of interest. This means that the eggshell can be placed such that it does not include the singularities in electromagnetic fields.

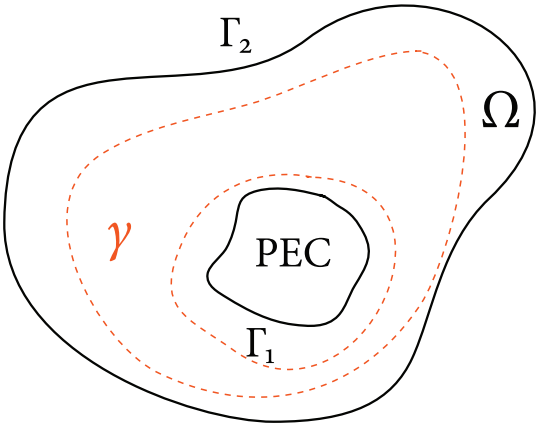


Figure 1.2: Eggshell γ

Chapter 2

Error estimation

2.1 Duality based error estimation

The discretization of (eq. 1.2) seeks an approximation $u_h \in V_h$ in a finite element subspace $V_h \subset V$

$$a(u_h, v_h) = 0 \quad \forall v_h \in V_h. \tag{2.1}$$

The dual (or adjoint) problem is

$$a(v, z) = J(v) \quad \forall v \in V. \tag{2.2}$$

Where J is our output functional or quantity of interest, in this work the force exerted on the inner body. The dual problem in its discretized form is

$$a(v_h, z_h) = J(v_h) \quad \forall v_h \in V_h.$$

From this we obtain a formula for the error in the linear output functional J,

$$\begin{aligned} J(e) &= a(e, z) = a(e, z - \psi_h) \\ &= (f, z - \phi_h) - a(u_h, z - \psi_h) =: \rho(u_h)(z - \psi_h), \quad \psi_h \in V_h. \end{aligned} \tag{2.3}$$

Cell-wise integration by parts implies

$$\begin{aligned} \rho(u_h)(z - \psi_h) &= \sum_{K \in \mathbb{T}_h} \{(f + \Delta u_h, z - \psi_h)_K - (\partial_n u_h, z - \psi_h)_{\partial K}\} \\ &= \sum_{K \in \mathbb{T}_h} \left\{(f + \Delta u_h, z - \psi_h)_K + \frac{1}{2}([\partial_n u_h], z - \psi_h)_{\partial K}\right\}, \end{aligned} \tag{2.4}$$

where $[\partial_n u_h]$ denotes the jump of $\partial_n u_h$ across the edges. For two neighboring cells $K, K' \in \mathbb{T}_h$ with common edge Γ and unit normal vector n pointing from K to K', we set

$$[\partial_n u_h] = [\nabla u_h \cdot n] := (\nabla u_{h|K' \cap \Gamma} - \nabla u_{h|K \cap \Gamma}) \cdot n.$$

$$
\begin{aligned}
R_{h|K} &:= f + \Delta u_h \\
r_{h|K} &:= \begin{cases} \frac{1}{2}[\partial_n u_h] & \text{if } \Gamma \subset \partial K \backslash \partial\Omega \\ 0 & \text{if } \Gamma \subset \partial\Omega \end{cases}
\end{aligned} \tag{2.5}
$$

Based on the previous results the a posteriori error representation reads,

$$
J(e) = \sum_{K \in \mathbb{T}_h} \left\{ (R_h, z - \psi_h)_K + (r_h, z - \psi_h)_{\partial K} \right\}, \tag{2.6}
$$

with an arbitrary $\psi_h \in V_h$. From the Cauchy-Schwarz inequality we get an upper bound for the error in the output functional

$$
|J(e)| \leq \eta_\omega := \sum_{K \in \mathbb{T}_h} \rho_K \omega_K, \tag{2.7}
$$

where the cell residuals ("smoothness indicators") ρ_K and weights ("influence factors") ω_K are given by

$$
\begin{aligned}
\rho_K &:= (\|R_h\|_K^2 + h_K^{-1} \|r_h\|_{\partial K}^2)^{1/2} \\
\omega_K &:= (\|z - \psi_h\|_K^2 + h_K \|z - \psi_h\|_{\partial K}^2)^{1/2} \\
h_K &:= \operatorname{diam}(K)
\end{aligned} \tag{2.8}
$$

2.1.1 Linearization of the output functional

The quantity of interest $F(u)$ is nonlinear, thus it must be linearized before we can solve the adjoint problem. For later use we compute the Gateaux derivative [6],

$$
\begin{aligned}
\mathrm{D}\, F(u)(v) &= \lim_{t \to 0} \frac{1}{t} \int_\Omega - \left(T(u + tv) - T(u) \right) \cdot \nabla \Psi \ \mathrm{d}x \\
&= \lim_{t \to 0} \frac{1}{t} \int_\Omega - (\nabla u \nabla u^T + t \nabla u \nabla v^T + t \nabla v \nabla u^T + t^2 \nabla v \nabla v^T \\
&\quad - \frac{1}{2} \left[(\nabla u, \nabla u) + 2t (\nabla u, \nabla v) + t^2 (\nabla v, \nabla v) \right] \mathbf{I} \\
&\quad - \nabla u \nabla u^T + \frac{1}{2} (\nabla u, \nabla u) \mathbf{I}) \cdot \nabla \Psi \ \mathrm{d}x \\
&= \int_\Omega - \left(\nabla u \nabla v^T + \nabla v \nabla u^T - (\nabla u, \nabla v) \mathbf{I} \right) \cdot \nabla \Psi \ \mathrm{d}x
\end{aligned} \tag{2.9}
$$

Since $F : \mathbb{R} \to \mathbb{R}^2$, two adjoint problems need to be solved. Find $z_1, z_2 \in V = H_0^1(\Omega)$ such that

$$
\begin{aligned}
a(v, z_1) &= \left[\mathrm{D}\, F(u)(v) \right]_{x_1} \qquad \forall v \in V \\
a(v, z_2) &= \left[\mathrm{D}\, F(u)(v) \right]_{x_2} \qquad \forall v \in V,
\end{aligned} \tag{2.10}
$$

where

$$
\begin{aligned}
\left[\mathrm{D}\, F(u)(v) \right]_{x_1} &= - \int_\Omega \partial_x v (\partial_x u \partial_x \Psi + \partial_y u \partial_y \Psi) + \partial_y v (\partial_x u \partial_y \Psi - \partial_y u \partial_x \Psi) \ \mathrm{d}x \\
\left[\mathrm{D}\, F(u)(v) \right]_{x_2} &= - \int_\Omega \partial_x v (\partial_y u \partial_x \Psi - \partial_x u \partial_y \Psi) + \partial_y v (\partial_x u \partial_x \Psi + \partial_y u \partial_y \Psi) \ \mathrm{d}x
\end{aligned}
$$

and Ψ is the cutoff function defined in (sec. 1.2). Please note that

$$F(u_h) - F(u) = \mathrm{D}\,F(u)(u_h - u) + O(\|u_h - u\|^2).$$

2.2 Practical error estimators

Important properties of an error estimator are:

- Sharpness of the approximate error representation $E(u_h)$
- Effectivity of the approximate local error indicators η_K

In practice (2.7) cannot be evaluated because it contains the exact solution of a variational problem z, which is in general unknown. The solution z can either be approximated by a finite element solution or the inequality (2.7) can be further estimated by using a priori analysis in the form of bounds for z in certain Sobolev norms.

Effectivity index The effectivity index I_{eff} is a measure for the sharpness of the error representation.

$$I_{\text{eff}} := \left| \frac{E(u_h)}{J(e)} \right| \tag{2.11}$$

An effectivity index of one is optimal, $I_{\text{eff}} \gg 1$ indicates over-estimation and $I_{\text{eff}} \ll 1$ under-estimation.

2.2.1 Approximation by a higher-order method

One possibility is to solve the dual problem by using biquadratic finite elements yielding the approximation $z_h^{(2)} \in V_h^{(2)}$. This estimator usually has a effectivity index close to one, but the computational cost is high. In most cases it is sufficient to employ linear finite elements and to construct a patchwise biquadratic interpolation.

`Name: EST1`

$$E^{(1)}(u_h) := \sum_{K \in \mathbb{T}_h} \left\{ \left(R_h, z_h^{(2)} - I_h z_h^{(2)} \right)_K + \left(r_h, z_h^{(2)} - I_h z_h^{(2)} \right)_{\partial K} \right\} \tag{2.12}$$

the corresponding local error indicator is then:

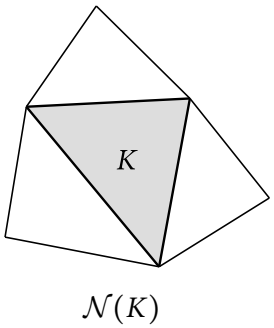
$$\eta_K^{(1)} = \left| \left(R_h, z_h^{(2)} - I_h z_h^{(2)} \right)_K + \left(r_h, z_h^{(2)} - I_h z_h^{(2)} \right)_{\partial K} \right| \tag{2.13}$$

2.2.2 Approximation by higher-order interpolation

`Name: EST2`

The computational cost can be reduced by solving the adjoint problem with linear finite elements instead of quadratic ones. The approximation to the exact solution z is then obtained by a patchwise higher-order interpolation to $\mathcal{S}^2$:

$$I_{2h}^{(2)} z_h := \arg\min_{v \in S^2} \|z_h - v\|_{L^2(\mathcal{N}(K))} \tag{2.14}$$



$\mathcal{N}(K)$

Figure 2.1: Patch for element K

$$E^{(2)}(u_h) := \sum_{K \in \mathbb{T}_h} \left\{ \left(R_h, I_{2h}^{(2)} z_h - z_h \right)_K + \left(r_h, z_h^{(2)} - I_{2h}^{(2)} z_h - z_h \right)_{\partial K} \right\} \tag{2.15}$$

and the corresponding local error indicator

$$\eta_K^{(2)} = \left| \left(R_h, I_{2h}^{(2)} z_h - z_h \right)_K + \left(r_h, z_h^{(2)} - I_{2h}^{(2)} z_h - z_h \right)_{\partial K} \right|. \tag{2.16}$$

2.2.3 Approximation by difference quotients

`Name: EST3`

We use the cell-wise interpolation estimate [3]

$$\omega_K = \|z - I_h z\|_K + h_K^{1/2} \|z - I_h z\|_{\partial K} \leq C h_K^2 |z|_K \tag{2.17}$$

According to Becker & Rannacher the two-seminorm of z is then replaced by a suitable second-order difference quotient.

$$E^{(3)}(u_h) := c_I \sum_{K \in \mathbb{T}_h} \rho_K h_K^{3/2} \|[\partial_n z_h]\|_{\partial K} \tag{2.18}$$

$$\eta_K^{(3)} = c_I h_K^{3/2} \rho_K \|[\partial_n z_h]\|_{\partial K} \tag{2.19}$$

The constant is choosen to be $c_I \approx 0.1 \ldots 1$. Usually strong over-estimation is observed.

The previous results were taken from [2].

2.2.4 Gradient recovery based estimator

`Name: EST4 (ErrEst_GOAL from LehrFEM)`

This estimator is basically identical with the previous one. Here the two-seminorm of z is computed by gradient-recovery.

$$E^{(4)}(u_h) := c \sum_{K \in \mathbb{T}_h} \rho_K(u_h) h_K^2 |z_h|_{2,K} \tag{2.20}$$

$$\eta_K^{(4)} = \rho_K(u_h) h_K^2 |z_h|_{2,K} \tag{2.21}$$

where

$$\rho_K(u_h) := \|f + \Delta u_h\|_{L^2(K)} + \frac{1}{2} h_K^{-1/2} \|[\partial_n u_h]_{\partial K}\|_{L^2(\partial K)}.$$

Usually strong over-estimation is observed.

2.3 Refinement process

1. Computation of elementwise error indicators $\eta_K^{(i)}$
2. Mark all elements with $\eta_K^{(i)} > TOL/M$, where M is the number of elements in the mesh and TOL is the desired tolerance.
3. Sort the marked elements in descending order.
4. Refine a fraction $\theta \in]0, 1]$ of the first (sorted) marked elements, the parameter θ is used to prevent from over-refinement. It turned out that $\theta = 0.6$ is a good choice.

2.4 Convergence properties

Theorem 2.4.1. *Provided that the problem is sufficiently regular, i.e. $z, u \in H^2(\Omega)$, the error in the output functional converges with $O(h^2)$. [1]*

Proof: We set $e^* = z - z_h$ and $e = u - u_h$ and use the relation (eq. 2.10)

$$\begin{aligned} a(e, z) &= \mathrm{D}\, F(u)(e) \\ a(e, e^*) &= \mathrm{D}\, F(u)(e) \quad \text{(by Galerkin-orthogonality)} \end{aligned}$$

Thus the error in the linearized output functional is represented by $a(u - u_h, z - z_h)$. And

$$a(z - z_h, u - u_h) \leq |z - z_h|_{1,\Omega} \, |u - u_h|_{1,\Omega} \leq Ch^2 \, |z|_{2,\Omega} \, |u|_{2,\Omega} \, .$$

Chapter 3

RESULTS

3.1 Model problems

For the model problem M02 an analytical solution exists. For all other examples considered a reference solution on a very fine mesh (> 500′000 elements) was computed in order to obtain convergence rates. For better illustration the isolines of the electric potentials are plotted in the following pictures.

Model problem: M1

The general solution for the Poisson equation on the circular annulus is given by

$$u(\phi, r) = \frac{1}{2}a_0 + b_0 \log(r) + \sum_{n=1}^{\infty} (a_n r^n + b_n r^{-n}) \cos(n\phi) + (c_n r^n + d_n r^{-n}) \sin(n\phi).$$

, after equating coefficients

$$a_0 = 3,\ a_2 = -\frac{1}{30},\ b_0 = -\frac{3}{2\log 2},\ b_2 = \frac{8}{15},\ c_1 = -\frac{1}{3},\ d_1 = \frac{4}{3}.$$

The exact force is $F(u) = \left[-\frac{8}{45}\pi + \frac{2}{\log(4)}, 0\right]$.

$$\begin{cases} \Delta u & = 0, \quad \text{in } \Omega \\ u_{\Gamma_1} & = \frac{1}{2}\left(3 + \cos(2\phi) + 2\sin(\phi)\right) \\ u_{\Gamma_2} & = 0 \end{cases}$$

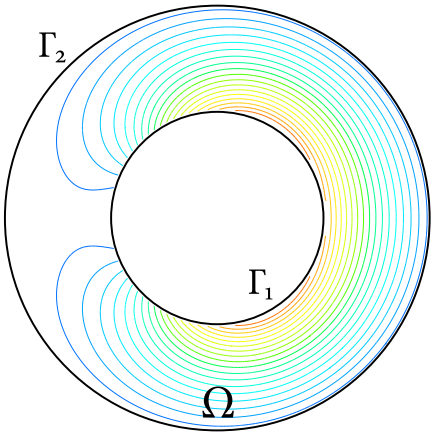


Figure 3.1: Domain M1, $r_i = 1$, $r_o = 2$

Model problem: M2

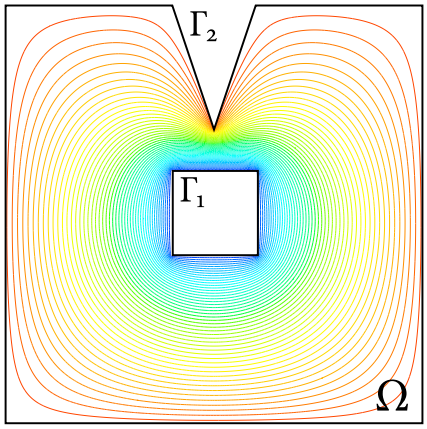


Figure 3.2: Domain M2

$$\begin{cases} \Delta u & = 0 \qquad x \in \Omega \\ u_{|\Gamma_1} & = 0 \\ u_{|\Gamma_2} & = 1 \end{cases}$$

Model problem: M3

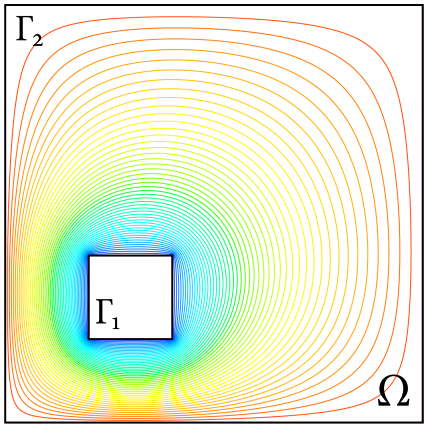


Figure 3.3: Domain M3

$$\begin{cases} \Delta u & = 0 \qquad x \in \Omega \\ u_{|\Gamma_1} & = 0 \\ u_{|\Gamma_2} & = 1 \end{cases}$$

Model problem: M4

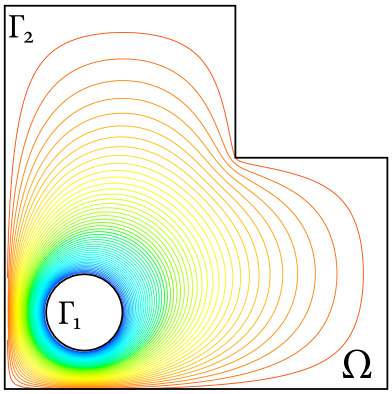


Figure 3.4: Domain M4

$$\begin{cases} \Delta u & = 0 \qquad x \in \Omega \\ u_{|\Gamma_1} & = 0 \\ u_{|\Gamma_2} & = 1 \end{cases}$$

Model problem: M5

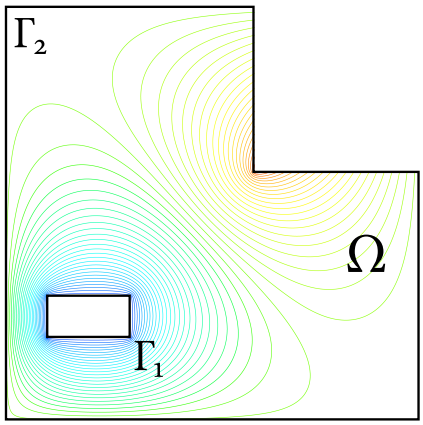


Figure 3.5: Domain M5

$$\begin{cases} \Delta u & = 0 \qquad x \in \Omega \\ u_{|\Gamma_1} & = 0 \\ u_{|\Gamma_2} & = g(x) \end{cases}$$

where

$$g(x) = \begin{cases} 1 + 5(x-1)(y-1) & x \text{ on reentrant corner} \\ 1 & x \text{ not on reentrant corner} \end{cases}$$

The convergence rates are added to the legend entries of the loglog-error plots. They were computed by fitting a first-order polynomial to all data points.

3.2 Numerical results

From (sec. 1.2) we know that the shape of the eggshell is free as long as it encloses the body where the force is computed on. Estep et al. [4] showed that goal-oriented refinement for a quantity of interest that has compact support may result in a dense mesh around its "effective domain of influence" and anywhere else the mesh can be relatively coarse. Thus one might hope that choosing the eggshell, such that $F(u)$ is compactly supported in Ω, a fine mesh only in a small region around its support could already give very accurate results. This is why, for every example considered, I used an eggshell extended on the entire domain and one that is not. Since two adjoint problems are solved (x,y-direction), their element error indicators are combined to $\eta_K = \sqrt{(}\eta_{K,x}^2 + \eta_{K,y}^2)$.

Error distribution plots In addition to the convergence rates, the elementwise error indicators are decomposed in weights ω_k and resiudals ρ_K and plotted in colormaps.
Estimator 1,2

$$\rho_K = \|R_h\|_K + \|r_h\|_{\partial K}$$
$$\omega_K = \|z_h^{(2)} - z_h\|_K$$

Estimator 3

$$\rho_K = \left(\|R_h\|_K^2 + \frac{1}{h_K}\|r_h\|_{\partial K}\right)^{1/2}$$
$$\omega_K = \|[\partial_n z_h]\|_{\partial_K}$$

Estimator 4

$$\rho_K = \|f + \Delta u_h\|_{L^2(K)} + \frac{1}{2}h_K^{-1/2}\|[\partial_n u_h]_{\partial K}\|_{L^2(\partial K)}$$
$$\omega_K = h_K^2 \left|z_h\right|_{2,K}$$

3.2.1 Model problem M1

Effect of the eggshell width

In order to study the effect of the eggshell width, the shell is placed in contact with the body and the convergence rates are computed for various widths on uniformly refined meshes. As it can be seen in (fig. 3.6) the diameter of the eggshell should be sufficiently large.

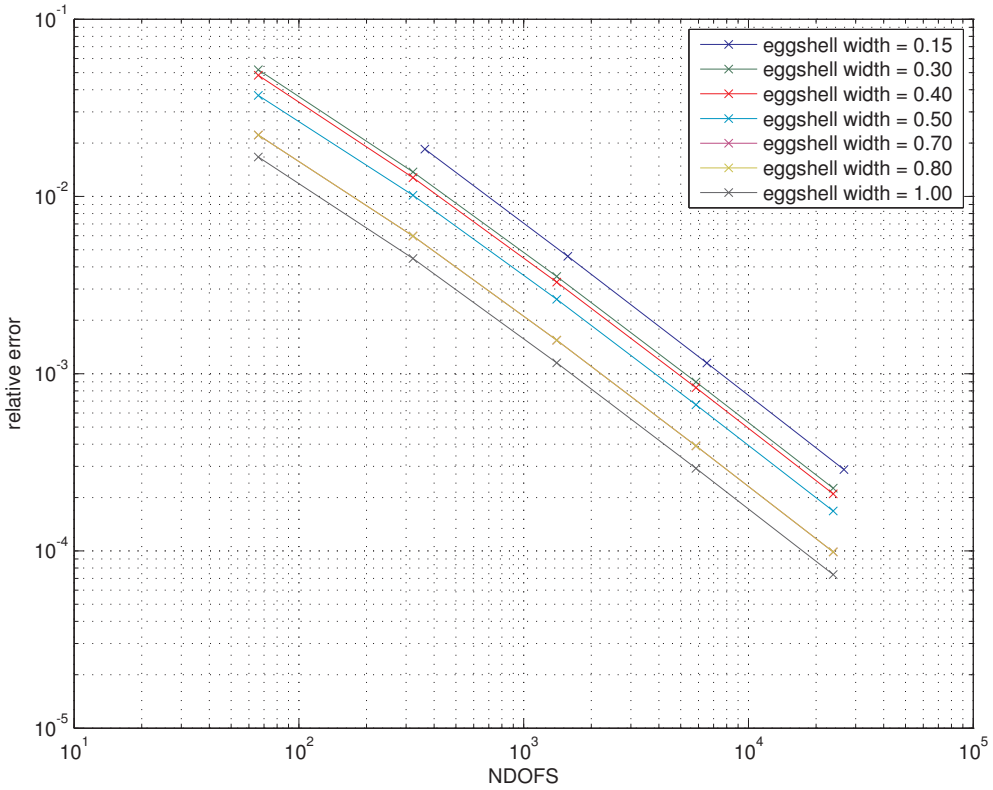


Figure 3.6: Convergence results wrt. eggshell width (model problem M1), width=1 corresponds to the entire domain

Force computation on entire domain

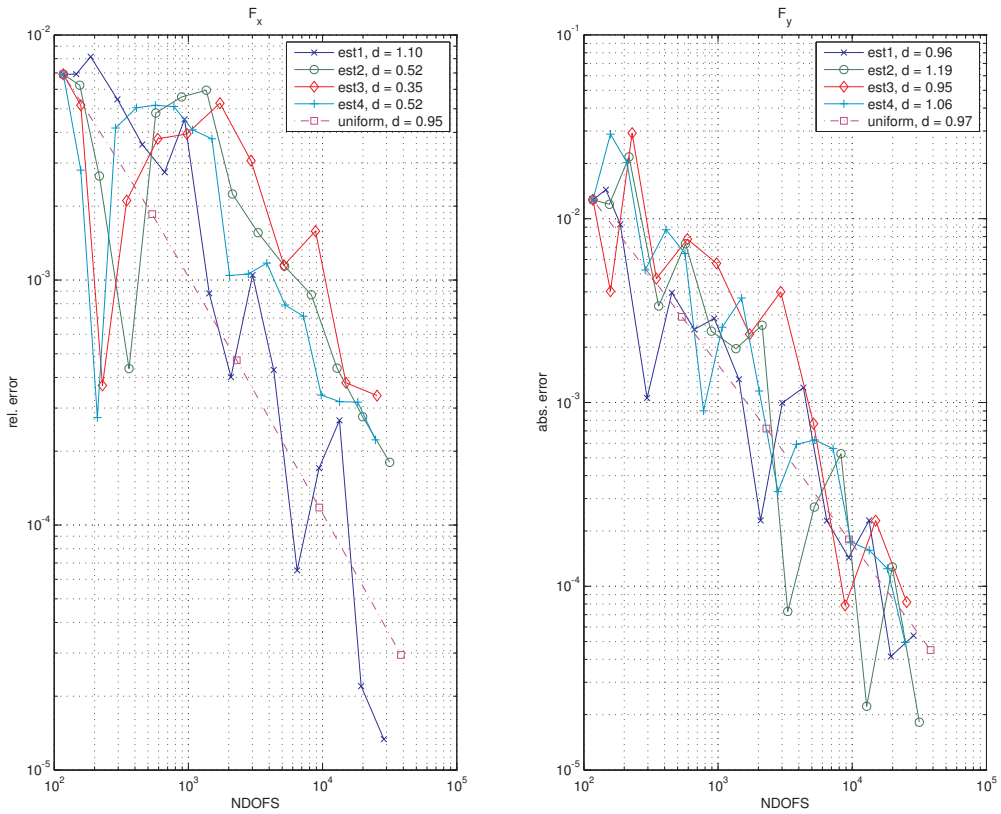
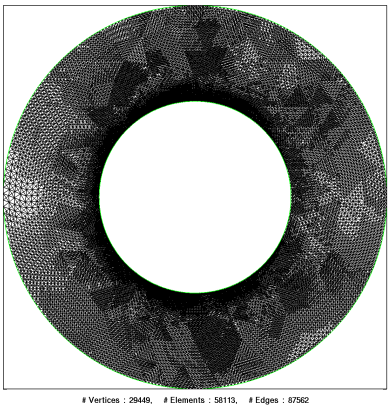
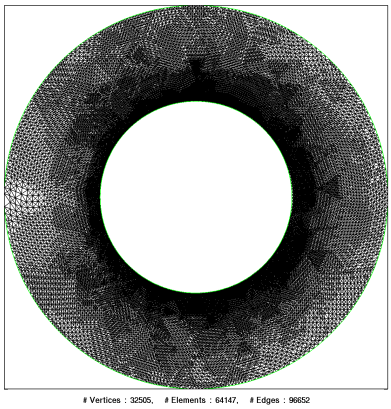


Figure 3.7: convergence rates (model problem M1, force computation on entire domain)

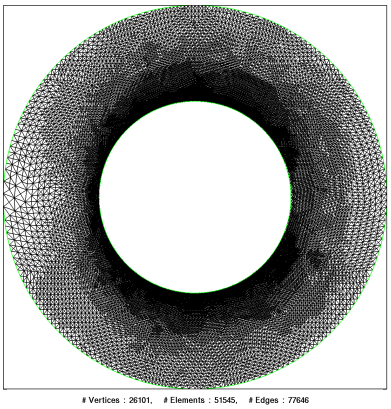
The optimal convergence rate of $O(h^2) = O(N^{-1})$ is already achieved with uniform refinement. Hence there is no benefit from the error estimators in this example.



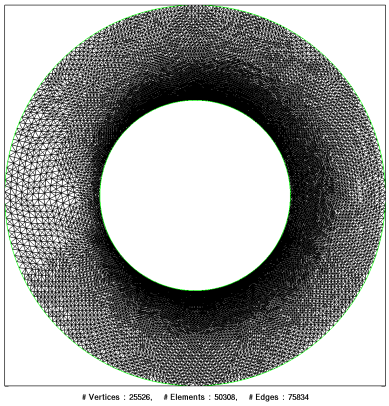
(a) Estimator 1



(b) Estimator 2



(c) Estimator 3



(d) Estimator 4

Figure 3.8: Refined meshes, (M1, force computation on entire domain)

Effectivity indices The effectivity indices for the estimators 3,4 are omitted from now on because they always have $I_{\text{eff}} \gg 1$. Both estimators have effectivity indices close to one. Underestimation

Est1	ndofs	66	141	271	502	913	1608	2889	5030	8610	14525	23312
	I_{eff}	1.958	2.140	0.668	0.933	2.870	1.373	2.420	1.435	0.453	10.941	0.841

Est2	ndofs	66	149	297	554	1064	2019	3740	6891	12866	23358
	I_{eff}	2.539	3.513	1.467	1.576	0.633	1.659	4.957	20.007	2.294	2.228

Table 3.1: effectivity indices (M1, force computation on entire domain)

can occur because of the approximation z. Thus one must be careful if they are used as a stopping criteria.

3.2.2 Model problem M2

Compact shell

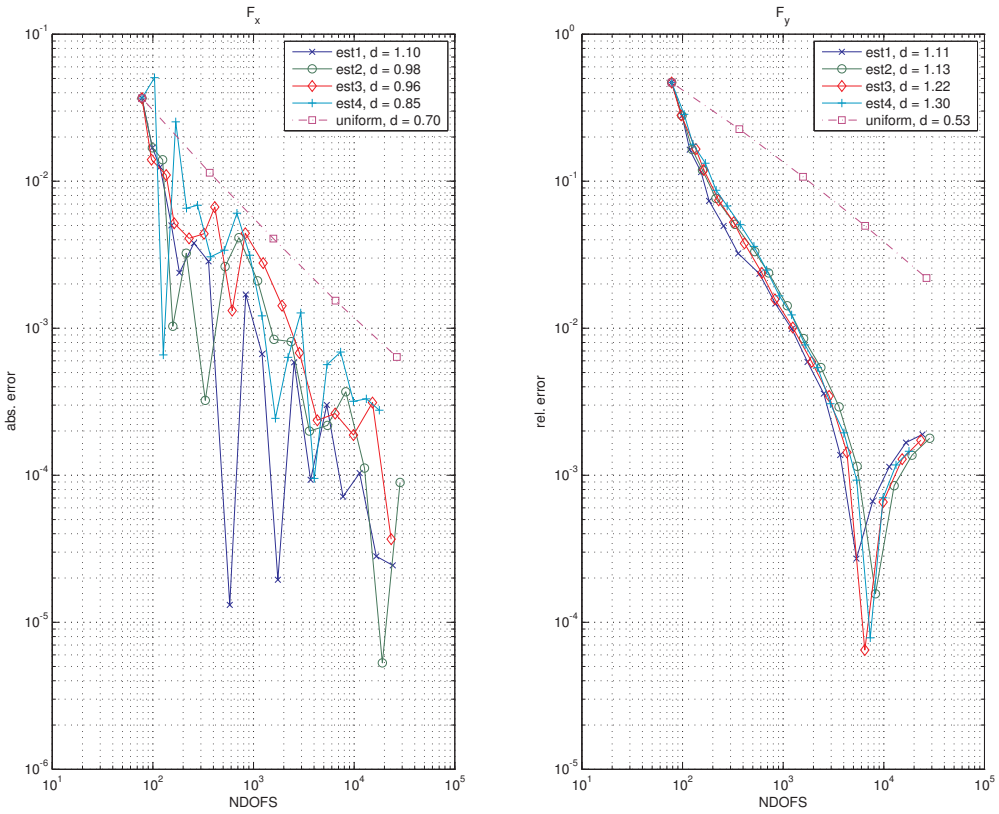
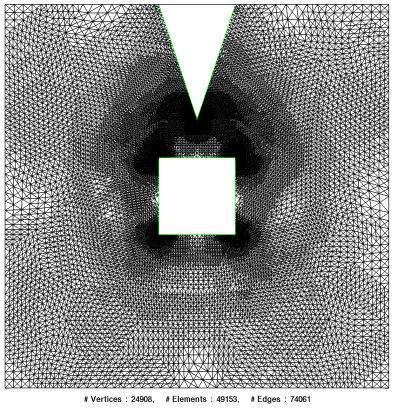
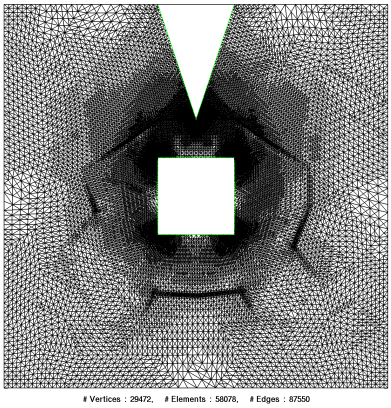


Figure 3.9: Convergence rates (M2, compact eggshell)

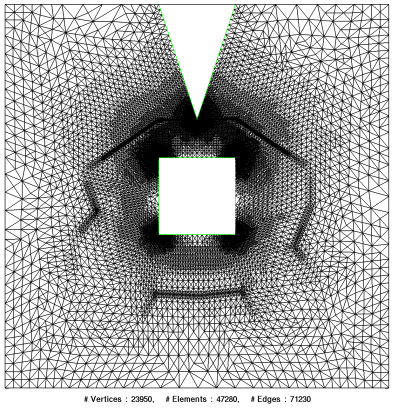
Because of symmetry reasons the force in x-direction is zero in this example. There are oscillations in the error in x-direction but on low level. In y-direction the estimators 2,3 achieve the optimal convergence rate. It must be noted that the estimators 1,2 show no improvement compared to the explicit residual based error estimator in the convergence rates.



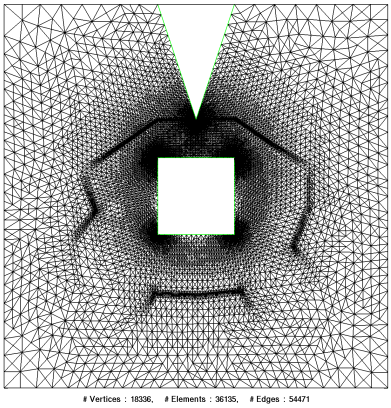
(a) Estimator 1



(b) Estimator 2



(c) Estimator 3



(d) Estimator 4

Figure 3.10: Refined meshes (M2, compact eggshell)

Effectivity indices Again estimators 1,2 have effictivity indices close to one, but under-estimation

Est1	ndofs	183	381	714	1268	2230	3877	6663	11485	19649	33404
	I_{eff}	0.829	0.816	0.799	0.818	0.839	0.869	0.932	1.028	1.253	1.760
Est1	ndofs	183	384	742	1368	2523	4635	8416	15240	27481	
	I_{eff}	0.508	1.623	1.674	1.511	1.381	1.891	1.633	2.199	2.120	

Table 3.2: Effectivity indices (M2, compact shell)

can occur.

Force computation on entire domain

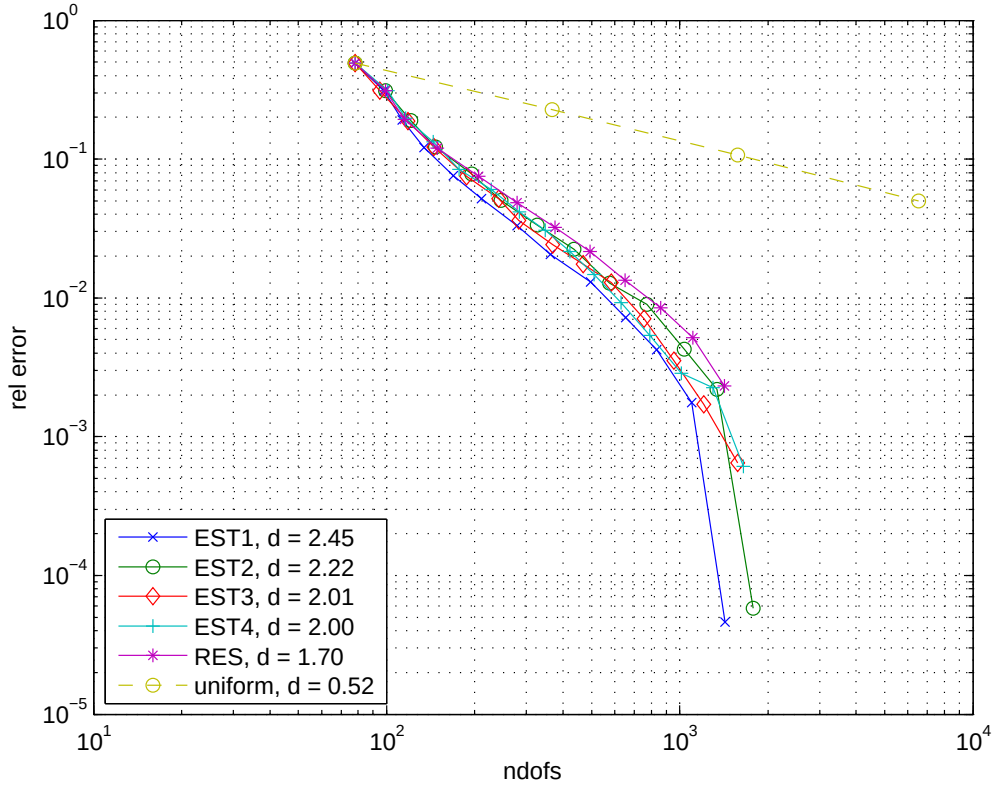
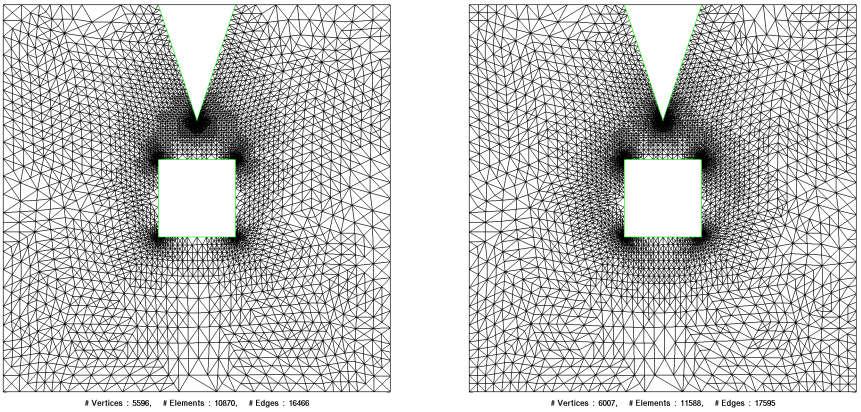


Figure 3.11: Convergence rates (M2, force computation on entire domain)

As in the previous example there are oscillations in the errors in x-direction. The convergence rates in y-direction are better than before.



(a) Estimator 1

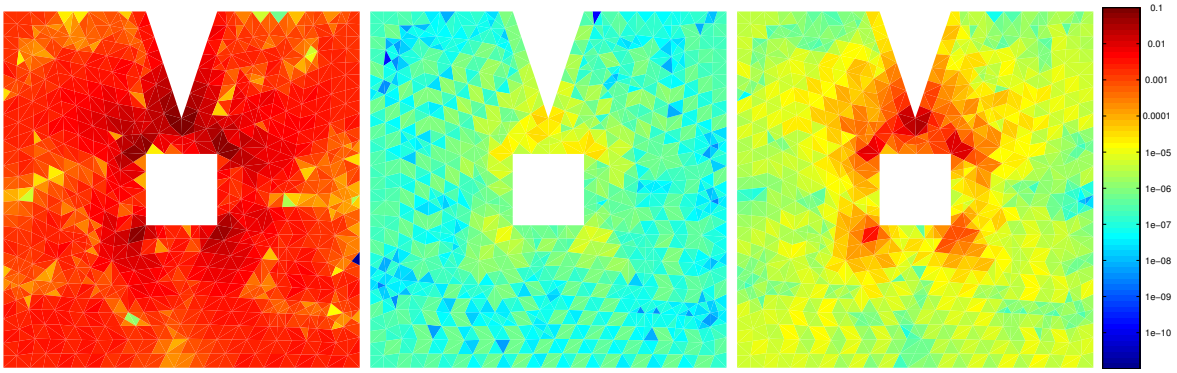
(b) Estimator 2

(c) Estimator 3

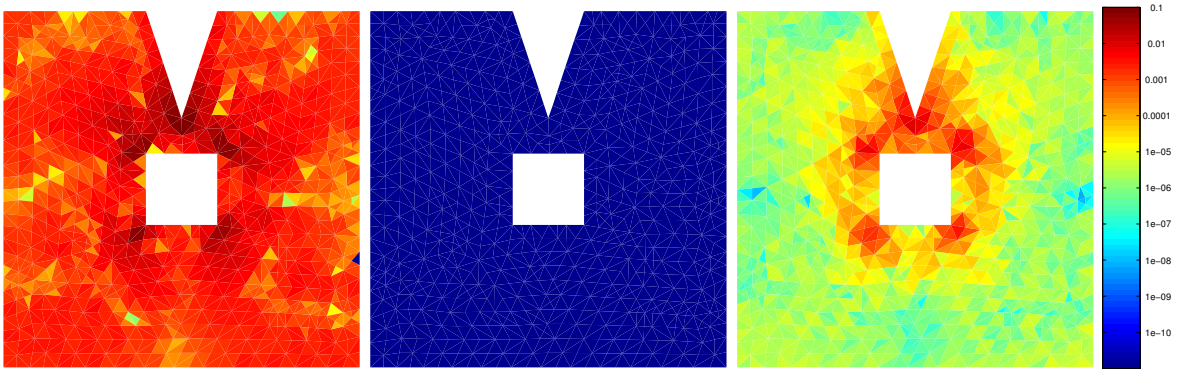
(d) Estimator 4

Figure 3.12: Refined meshes, (M2, force computed on entire domain)

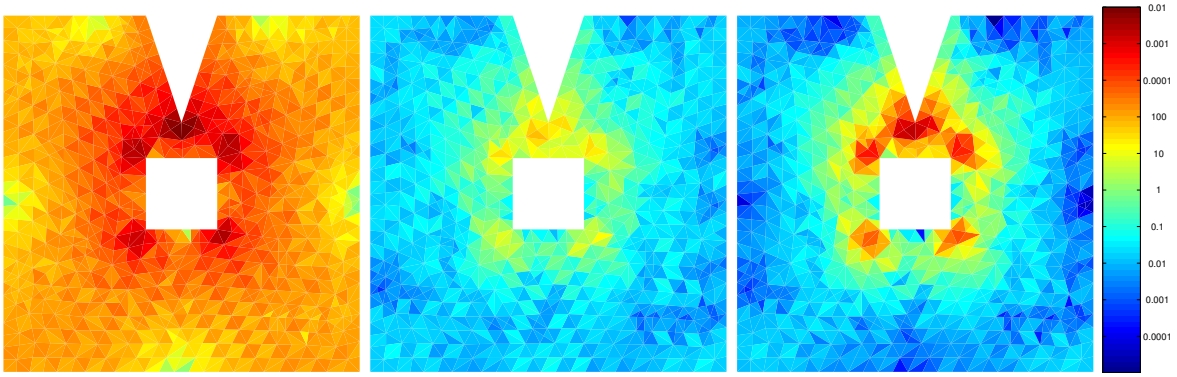
Error distribution



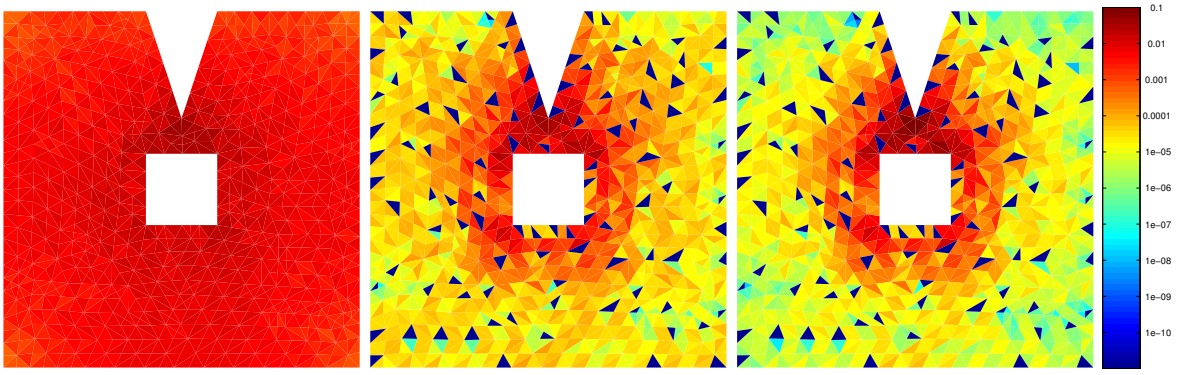
(a) Est1, ρ_K (b) Est1, ω_K (c) Est1, η_K



(e) Est2, ρ_K (f) Est2, ω_K (g) Est2, η_K



(i) Est3, ρ_K (j) Est3, ω_K (k) Est3, η_K



(m) Est4, ρ_K (n) Est4, ω_K (o) Est4, η_K

Est1	ndofs	78	155	284	504	872	1522	2591	4407	7472	12549	20726	33834
	I_{eff}	0.759	0.875	0.873	0.938	0.959	1.045	1.117	1.266	1.482	1.899	3.075	10.164
Est2	ndofs	78	157	310	581	1080	1985	3582	6482	11584	20721	36465	
	I_{eff}	0.256	0.370	0.125	0.114	0.038	0.647	0.811	1.896	3.016	5.922	13.681	

Table 3.3: M2, shell entire domain

Effectivity indices

3.2.3 Model problem M3

Compact eggshell

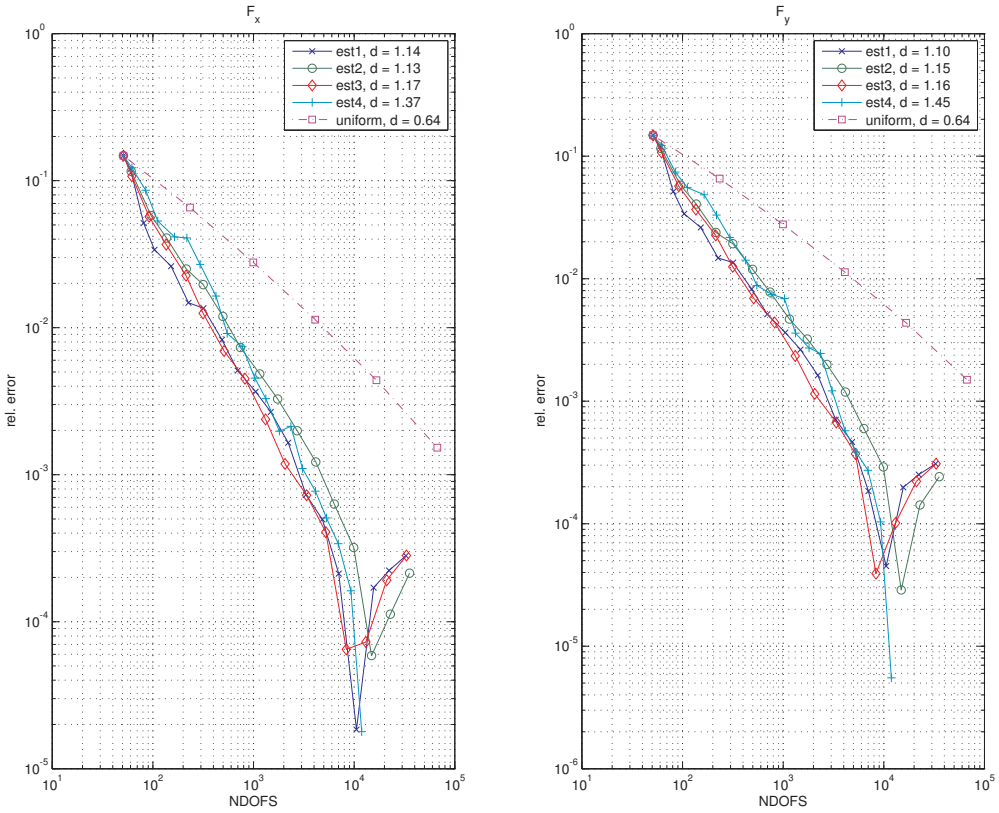
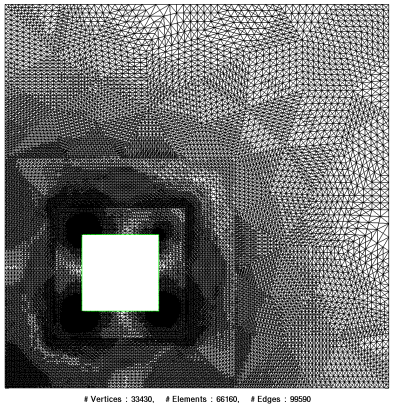
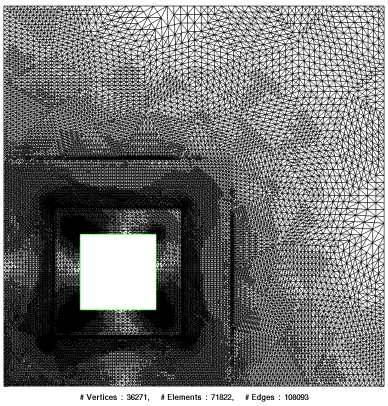


Figure 3.13: Convergence rates (M3, compact eggshell)

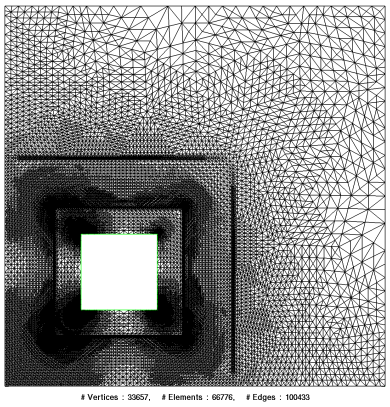
Effectivity indices



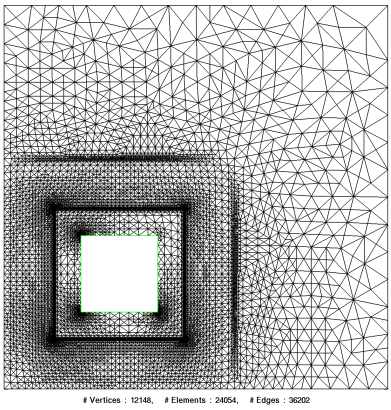
(a) Estimator 1



(b) Estimator 2



(c) Estimator 3



(d) Estimator 4

Figure 3.14: Refined meshes, (M3, compact eggshell)

Est1	ndofs	51	103	191	331	591	1040	1781	3082	5334	9026	15151	25256
	I_{eff}	0.721	0.874	0.825	0.842	0.824	0.849	0.859	0.884	0.925	0.994	1.131	1.375
Est2	ndofs	51	102	191	352	640	1198	2242	4126	7617	13958	25591	
	I_{eff}	0.402	2.457	1.547	2.081	1.969	1.679	1.737	2.116	1.740	2.478	2.398	

Table 3.4: Effectivity indices (M3, compact eggshell)

Force computation on entire domain

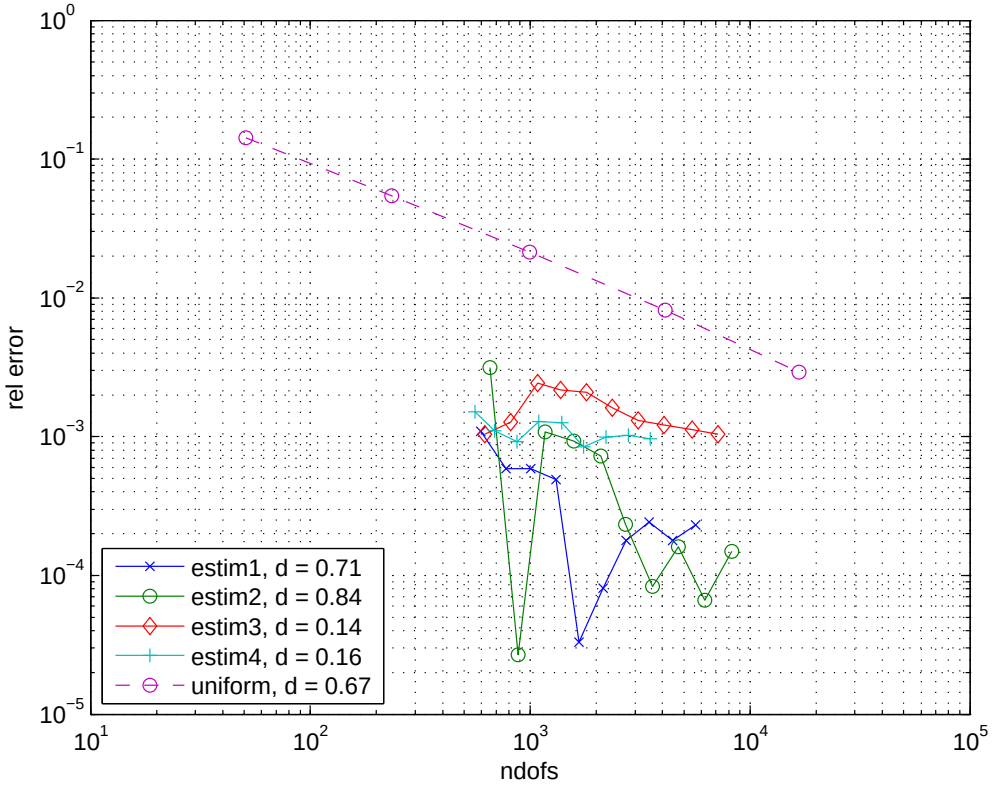


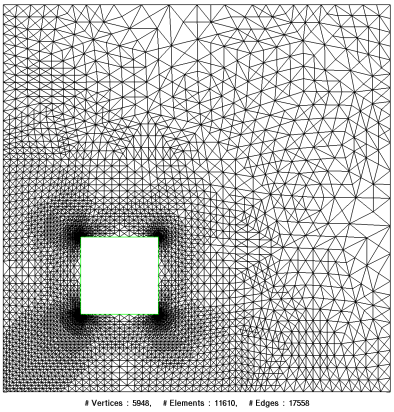
Figure 3.15: Convergence rates (M3, force comp. on entire domain)

The convergence in x-direction breaks down after 1000 degrees of freedom, this is most likely because the accuracy of the reference solution was reached on that point. In general we observe again better convergence rates when the eggshell is extended to the entire domain.

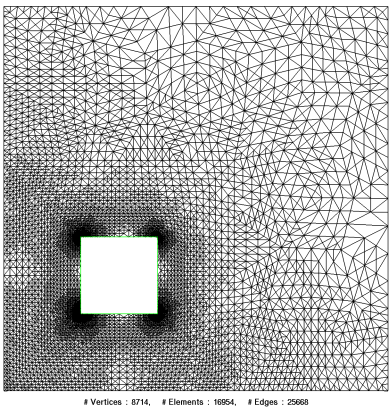
Effectivity indices

Est1	ndofs	83	165	293	525	914	1587	2704	4625	7815	13190	21809
	I_{eff}	0.890	1.068	1.010	1.121	1.091	1.259	1.228	1.414	1.376	1.393	1.066
Est2	ndofs	83	167	316	602	1089	2004	3653	6658	11965	21657	
	I_{eff}	0.850	1.204	0.667	0.962	1.440	1.504	2.718	2.538	5.106	3.604	

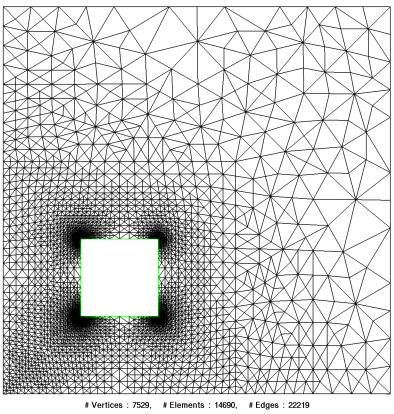
Table 3.5: Effectivity indices (M4, force computation on entire domain)



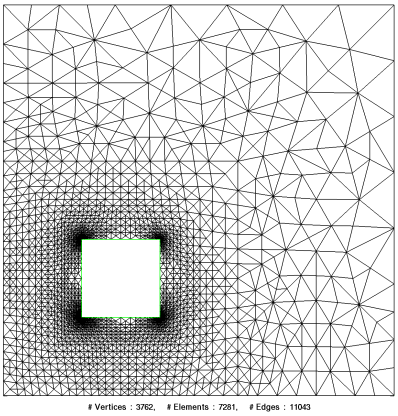
(a) Estimator 1



(b) Estimator 2



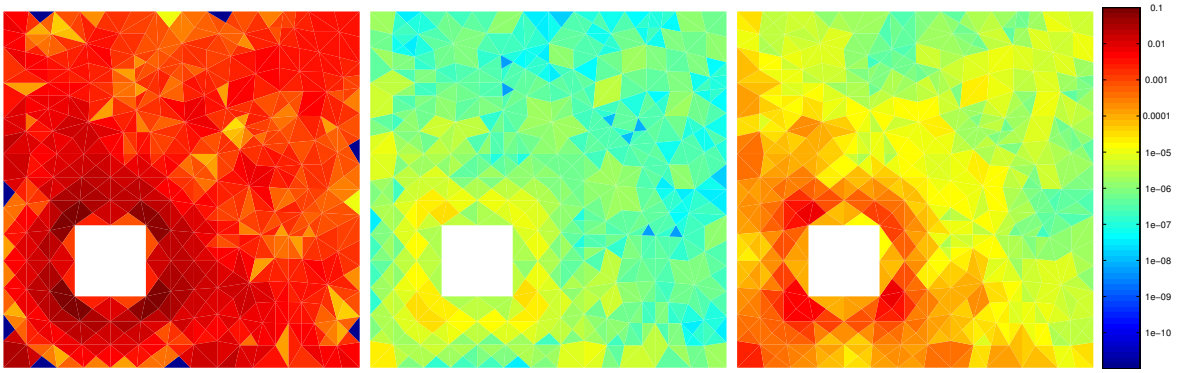
(c) Estimator 3



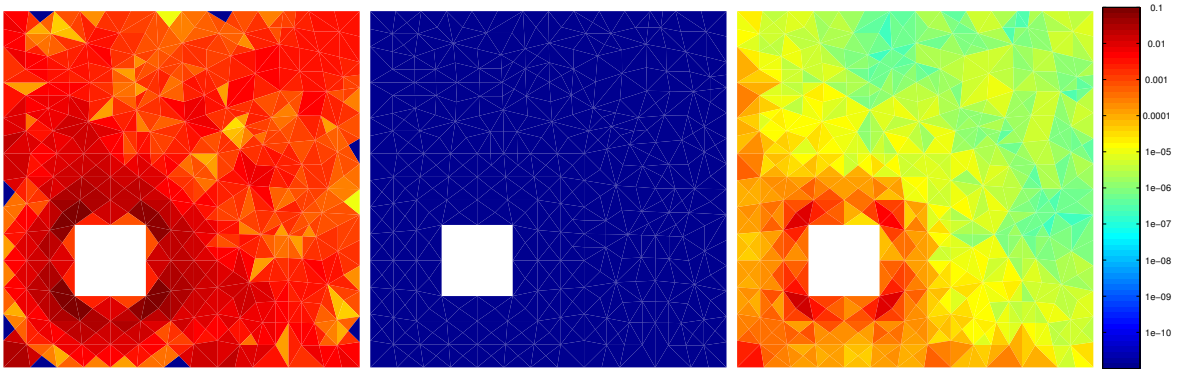
(d) Estimator 4

Figure 3.16: refined meshes, (M4, force computed on entire domain)

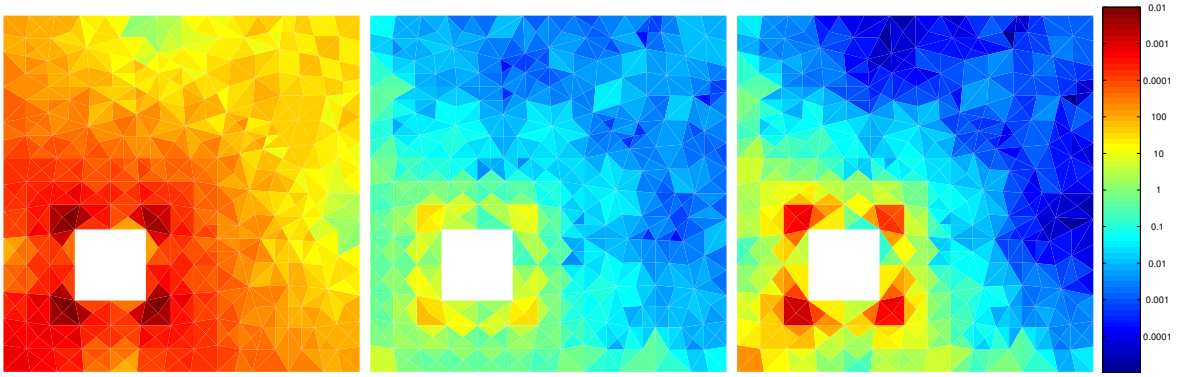
Error distribution



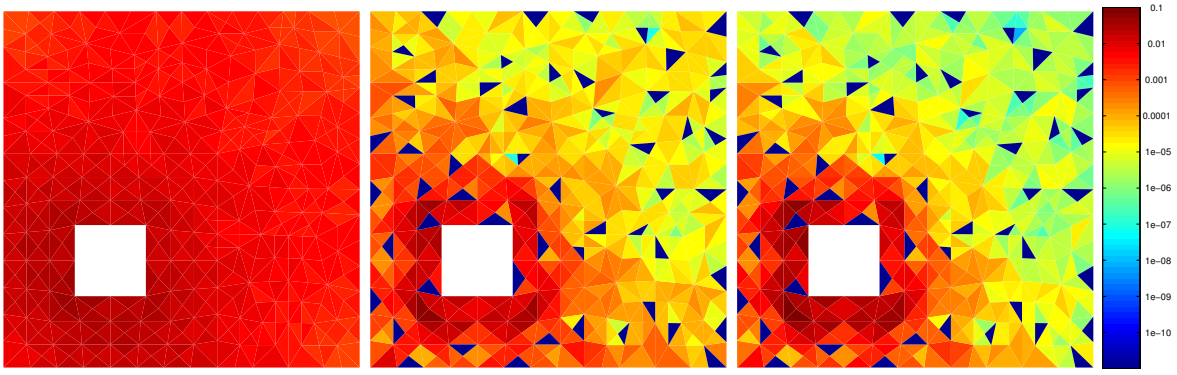
(a) Est1, ρ_K (b) Est1, ω_K (c) Est1, η_K



(e) Est2, ρ_K (f) Est2, ω_K (g) Est2, η_K



(i) Est3, ρ_K (j) Est3, ω_K (k) Est3, η_K



(m) Est4, ρ_K (n) Est4, ω_K (o) Est4, η_K

3.2.4 Model problem M4

The performance in this example is poor. There a huge oscillations in the convergence rates. It seems that the problem originates in the curvilinear object.

Compact eggshell

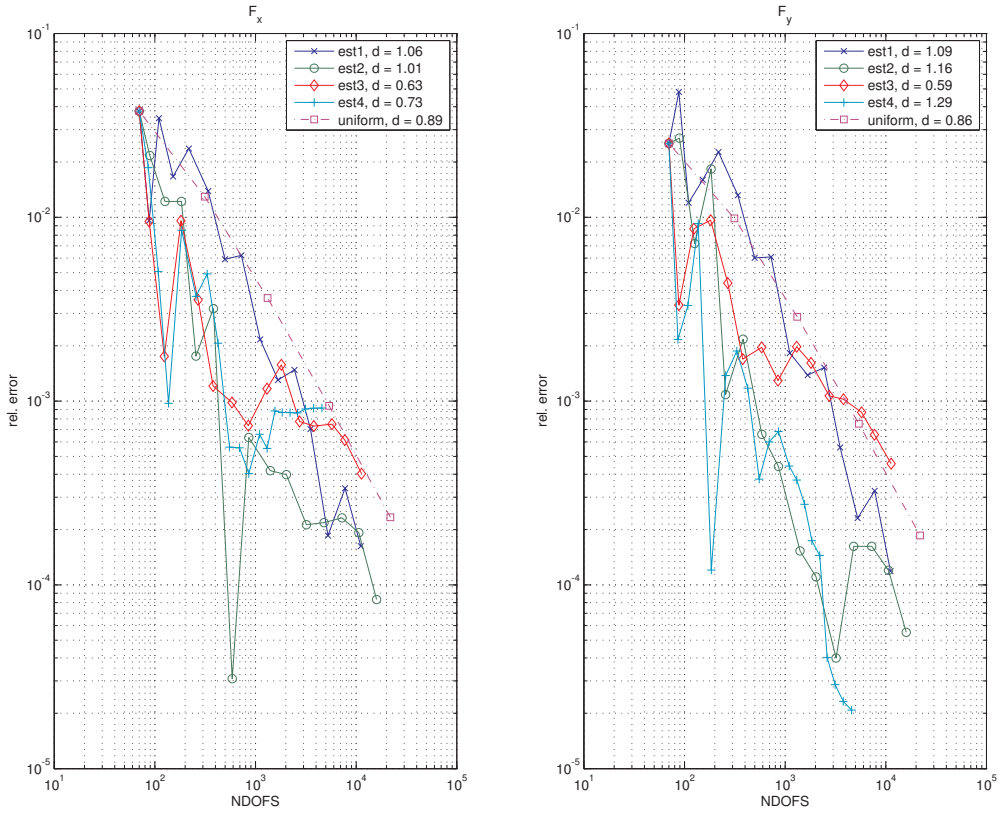
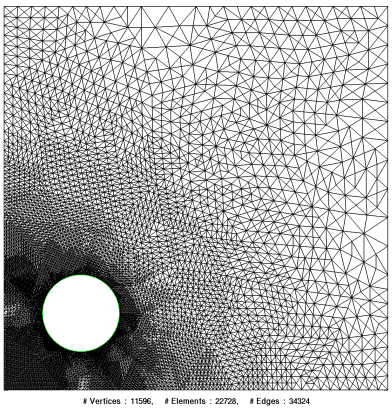
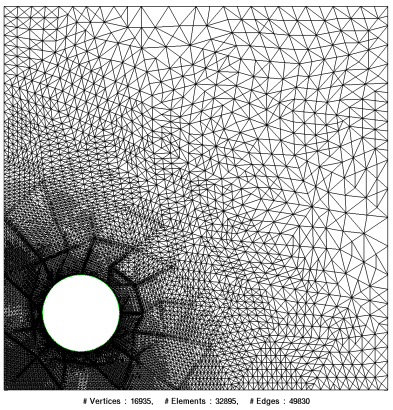


Figure 3.17: Convergence rates (M4, compact eggshell)

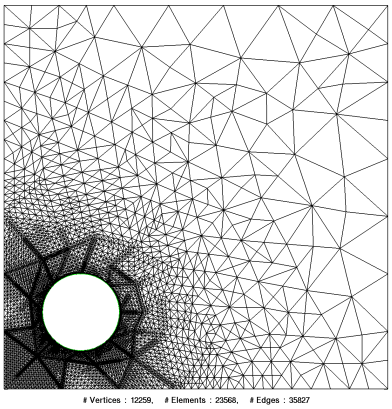
Force computation on entire domain



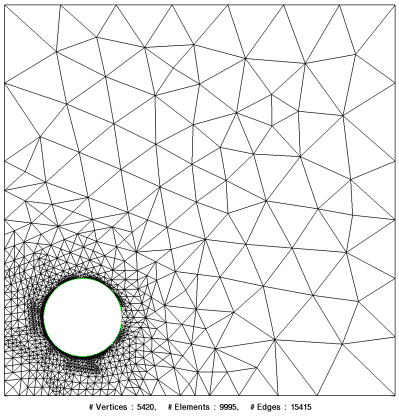
(a) Estimator 1



(b) Estimator 2



(c) Estimator 3



(d) Estimator 4

Figure 3.18: refined meshes, (M04, compact eggshell)

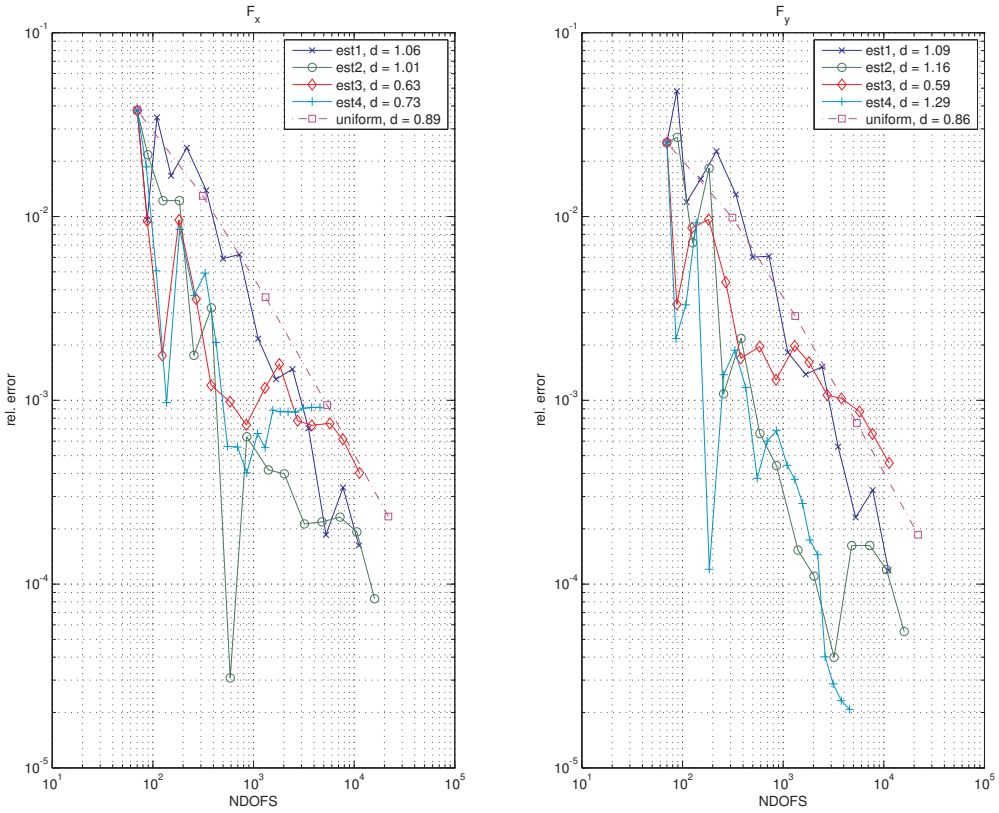
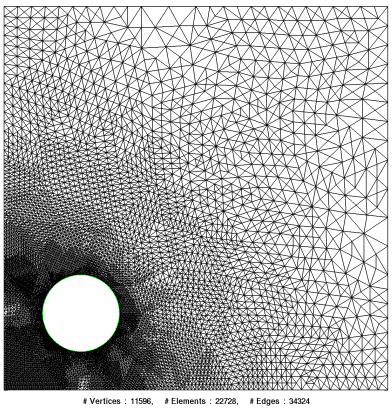
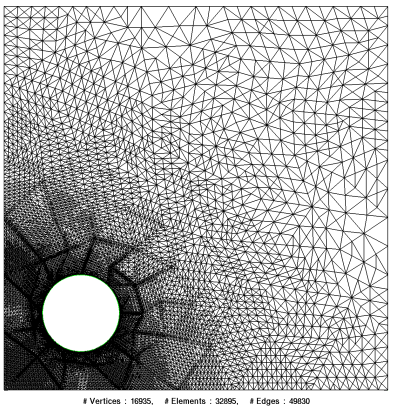


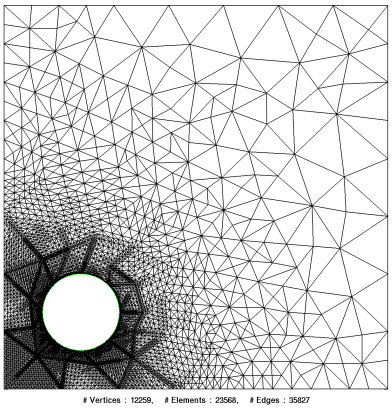
Figure 3.19: Convergence rates (M6, force computation on entire domain)



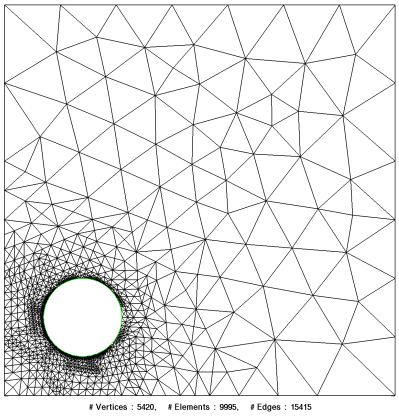
(a) Estimator 1



(b) Estimator 2



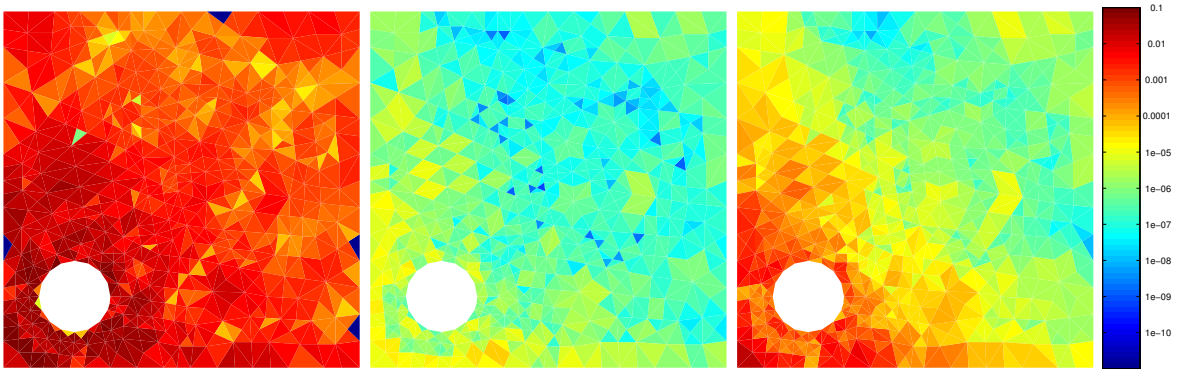
(c) Estimator 3



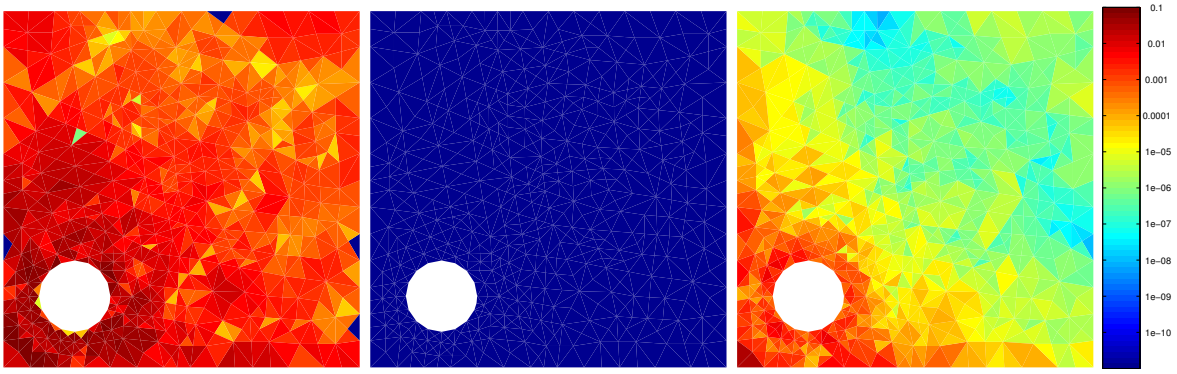
(d) Estimator 4

Figure 3.20: Refined meshes, (M04, force computed on entire domain)

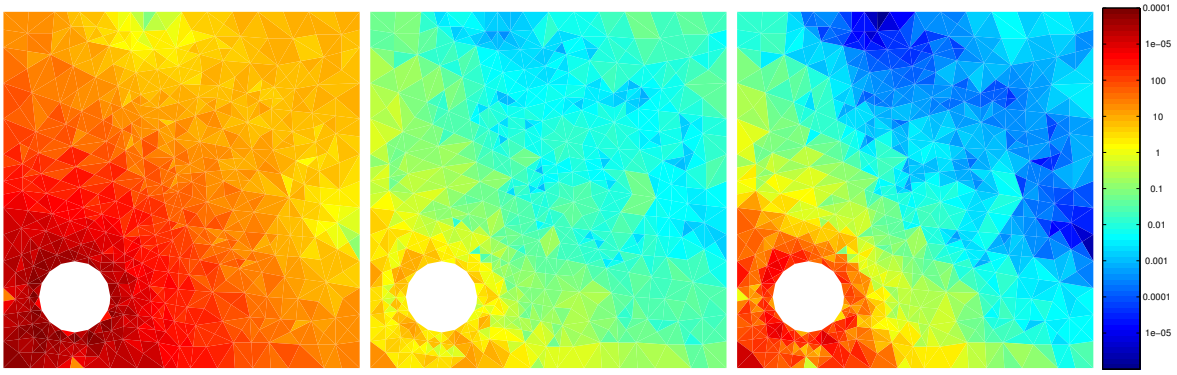
Error distribution



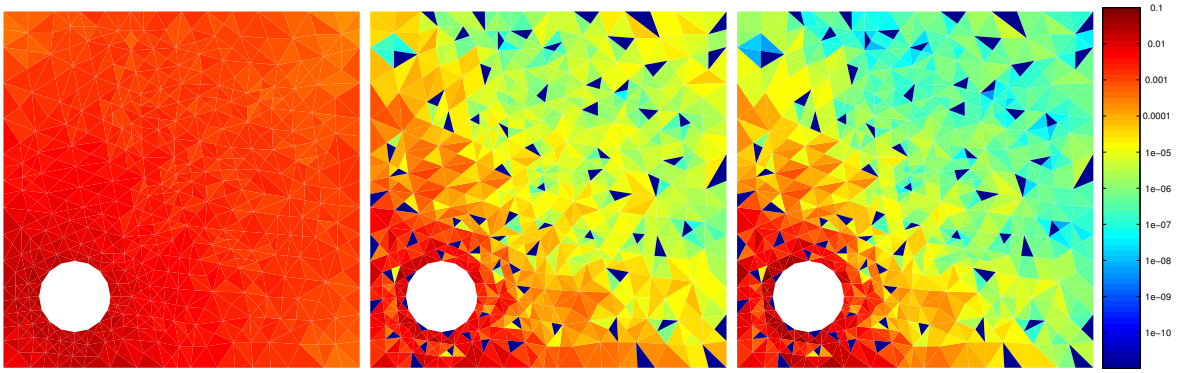
(a) Est1, ρ_K (b) Est1, ω_K (c) Est1, η_K



(e) Est2, ρ_K (f) Est2, ω_K (g) Est2, η_K



(i) Est3, ρ_K (j) Est3, ω_K (k) Est3, η_K



(m) Est4, ρ_K (n) Est4, ω_K (o) Est4, η_K

3.2.5 Model problem M5

Force computed on entire domain

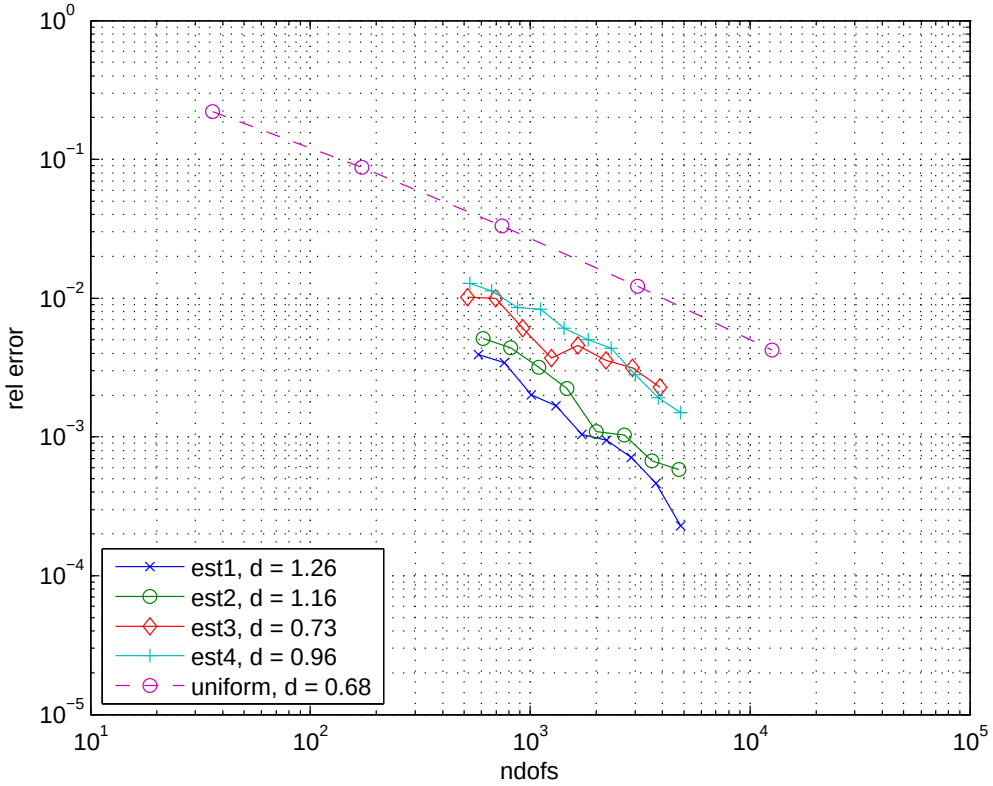


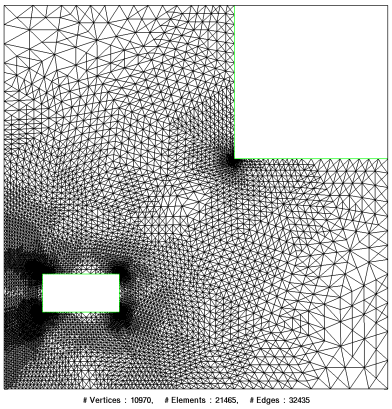
Figure 3.21: Convergence rates (M5, force computed on entire domain)

The convergence rates of all estimators in y-direction is poor. If the error estimation in x-direction is turned off, i.e. only the errors in y-direction are considered, the convergence rate can be recovered at the expense of a higher computational effort.

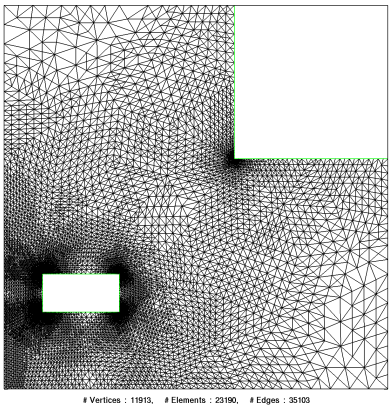
Effectivity indices

Est1	ndofs	36	74	141	255	462	813	1419	2458	4224	7199	12212
	I_{eff}	0.904	0.996	0.920	0.989	0.905	0.982	0.905	0.990	0.921	1.018	0.964
Est2	ndofs	36	75	153	290	550	1027	1902	3495	6408	11731	
	I_{eff}	0.379	0.460	0.530	0.288	0.892	0.534	1.093	0.881	1.208	1.048	

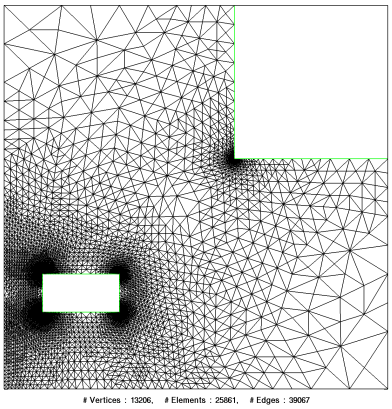
Table 3.6: Effectivity indices (M5, shell entire domain)



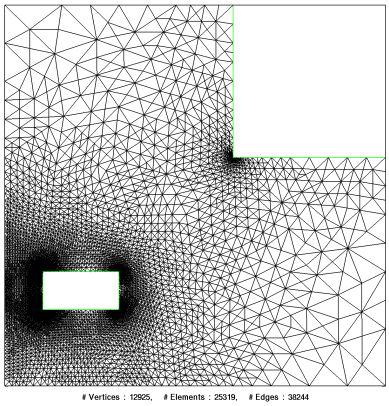
(a) Estimator 1



(b) Estimator 2



(c) Estimator 3



(d) Estimator 4

Figure 3.22: Refined meshes, (M5, force computed on entire domain)

Compact eggshell

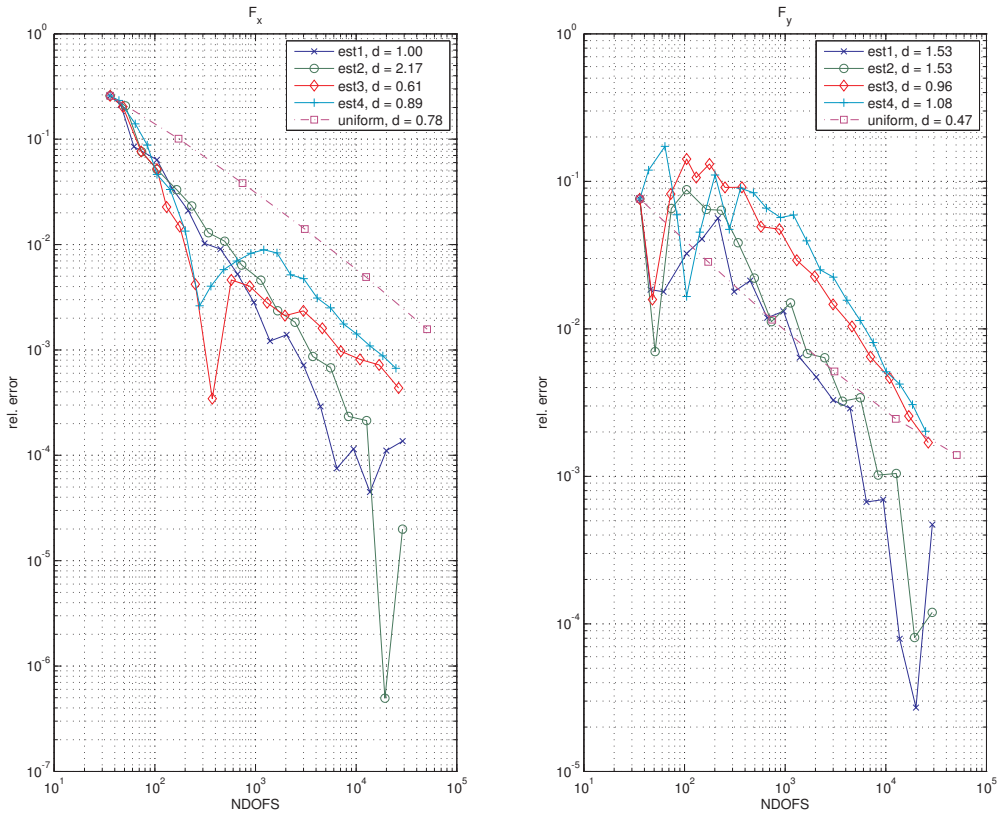
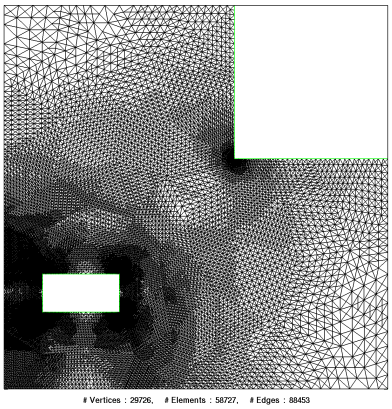


Figure 3.23: Convergence rates (M8, compact shell)

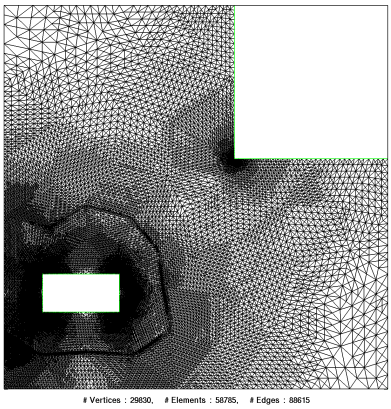
Effectivity indices

Est1	ndofs	125	249	455	810	1416	2448	4187	7112	12045	20293	34011
	I_{eff}	0.849	0.847	0.915	0.884	0.971	0.922	1.090	1.065	1.491	1.653	6.424
Est2	ndofs	125	261	491	925	1703	3128	5684	10282	18510	33424	
	I_{eff}	1.032	0.531	0.314	0.743	0.519	1.747	0.729	3.965	1.714	12.679	

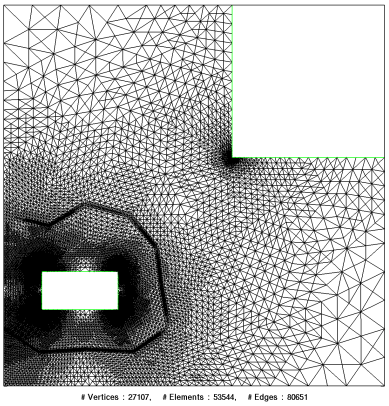
Table 3.7: Effectivity indices (M5, compact shell)



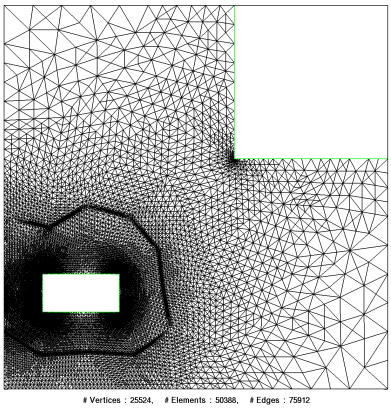
(a) Estimator 1



(b) Estimator 2



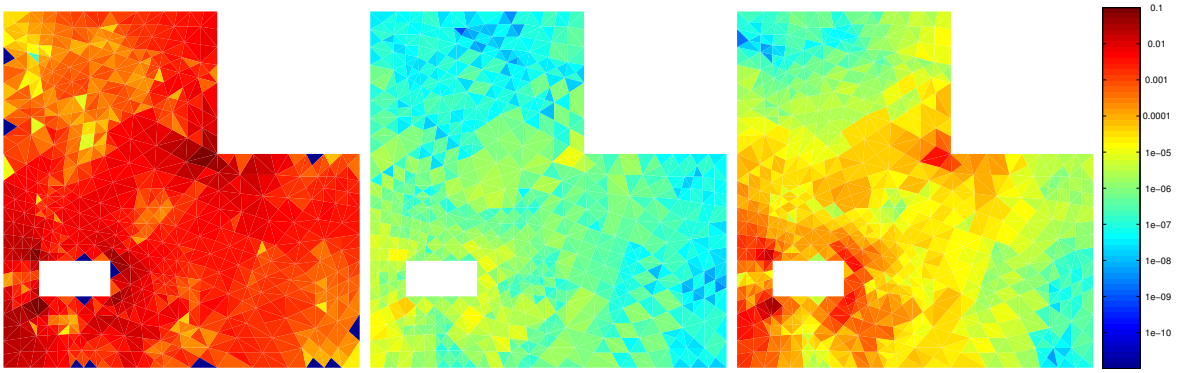
(c) Estimator 3



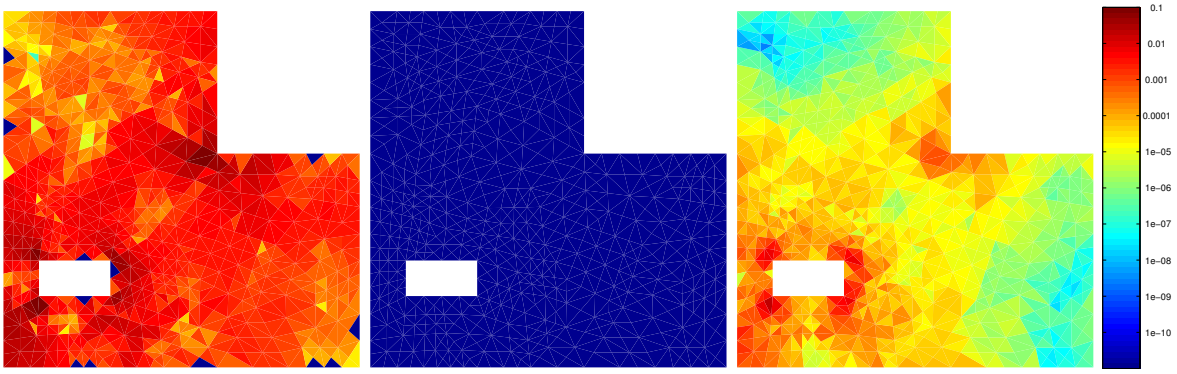
(d) Estimator 4

Figure 3.24: Refined meshes, (M5, compact shell)

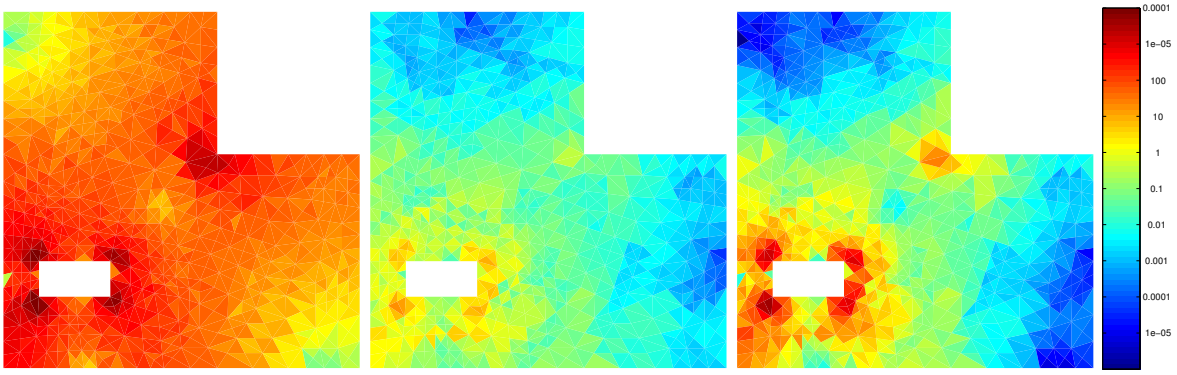
Error distribution



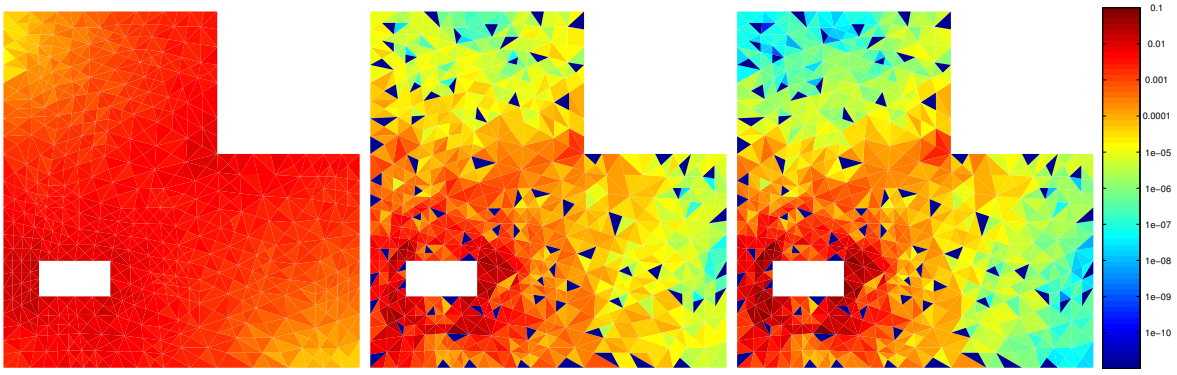
(a) Est1, ρ_K (b) Est1, ω_K (c) Est1, η_K



(e) Est2, ρ_K (f) Est2, ω_K (g) Est2, η_K



(i) Est3, ρ_K (j) Est3, ω_K (k) Est3, η_K



(m) Est4, ρ_K (n) Est4, ω_K (o) Est4, η_K

Chapter 4

Conclusion

It has been shown that with duality based error estimation high convergence rates can be obtained in electrostatic force computation. The estimators based on a higher-order dual problem and higher-order interpolation turned out to give accurate bounds for the error in the output functional, although under-estimation can occur. Compared to the explicit residual estimator, which is very cheap to compute, they showed no improvement in the convergence rate. The estimators 3,4 (approximate differences, gradient recovery) give no efficient bounds for the error, but are well suited for mesh refinement. Thus it seems to be the best strategy to use one of the first two estimators to get a bound for the error and to refine the mesh based on estimator 3 or 4. Since the evalution of an estimator has only a small additional cost once the dual solution is available. It was observed that there are problems with the convergence rates if the boundary cannot be represented exactly by triangles. It is possible that this can be mitigated if curvilinear elements are used, but that was not done in this work.

Bibliography

[1] M. Ainsworth and John Tinsley Oden. *A posteriori error estimation in finite element analysis*. Wiley-IEEE, 2000.

[2] Wolfgang Bangerth and Rolf Rannacher. *Adaptive finite element methods for differential equations*. Birkhäuser, 2003.

[3] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 2008.

[4] Donald Estep, Michael Holst, and Mats Larson. Generalized green's functions and the effective domain of influence. *SIAM J. Sci. Comput.*, 26(4):1314–1339, 2005.

[5] François Henrotte, Geoffrey Deliége, and Kay Hameyer. The eggshell approach for the computation of electromagnetic forces in 2D and 3D. *COMPEL: The International Journal for Computation and Mathematics in Electrical and Electronic Engineering*, 23(4):996–1005, 2004.

[6] Erdoğan S. Suhubi. *Functional analysis*. Springer, 2003.

Chapter 5

CODE

5.1 Mesh

This routine is called only on the first mesh in a simulation. The routines for uniform and adaptive mesh refinement where also adapted such that the boundary vertices and the elements marked as eggshell elements were updated after every refinement.

```
function [Mesh] = add_eggshell_dist(Mesh,dist_inner,dist_outer)
% [Mesh] = add_eggshell_dist( Mesh, dist_inner, dist_outer)
%  Author: Simon Pintarelli, simonpi@student.ethz.ch
%
%   ARGUMENTS
%                Mesh       = ...
%                dist_inner = distance handle for inner boundary
%                dist_outer = distance handle for outer boundary
%
%   DESCRIPTION
%                - add the fields 'EggShellCoordsInnerBd' and '
    EggShellCoordsInnerBd' to the struct Mesh
%                - mark all eggshell elements with ElemFlag = -1

  dist_region = @(x) dist_diff(dist_outer(x),dist_inner(x));
  eggshell_elem_flag = -1;

  nElements = size(Mesh.Elements,1);

  % mark elements , reset elem_flags
  Mesh.ElemFlag = ones(nElements,1);

  vidx = Mesh.Elements;
  C    = (Mesh.Coordinates(vidx(:,1),:) + Mesh.Coordinates(vidx(:,2),:) +
       Mesh.Coordinates(vidx(:,3),:))/3;
  l    = dist_region(C) <= 0;
  Mesh.ElemFlag(l) = eggshell_elem_flag;

  Mesh = extract_boundary(Mesh,dist_inner,dist_outer);
end
```

```
function [Mesh] = extract_boundary(Mesh,dist_inner,dist_outer)
  % all vertex id's belonging in the eggshell

  el_id_eggshell = find(Mesh.ElemFlag == -1);
  all_vid = unique(Mesh.Elements(el_id_eggshell,:));
  all_eid = [];

  for i = 1:length(el_id_eggshell)
    vid = Mesh.Elements(el_id_eggshell(i),:);
    e1 = Mesh.Vert2Edge(vid(1),vid(2));
    e2 = Mesh.Vert2Edge(vid(2),vid(3));
    e3 = Mesh.Vert2Edge(vid(3),vid(1));
    all_eid = [all_eid; e1; e2; e3];
  end

  % n x 2, matrix, with left and right element id
  all_left_and_right_elem = Mesh.Edge2Elem(all_eid,:);

  % remove edges that are not inside the eggshell region

  % find all boundary edges, of the domain!
  [tmp, ~] = find(all_left_and_right_elem == 0);
  edg_id_boundary_dom = all_eid(tmp);
  vid_bd_edges = unique(Mesh.Edges(edg_id_boundary_dom,:));

  % contains all edges belonging to the eggshell that are not part of the
      boundary
  edg_id_interior_eggshell = setdiff(all_eid,edg_id_boundary_dom);

  % [LHS, RHS] ELEM ID of the "interior eggshell edges"
  ele_id_lar_e = Mesh.Edge2Elem(edg_id_interior_eggshell,:);

  interior_elem_flags = Mesh.ElemFlag(ele_id_lar_e);

  % edge ids that are not part of the eggshell boundary
  edg_id_eg     = edg_id_interior_eggshell(sum(interior_elem_flags,2) ~=
      -2); % -2 means both elems belong to eggshell
  vid_not_bd_eg = unique(Mesh.Edges(edg_id_eg,:));
  vid_eg        = [vid_not_bd_eg;vid_bd_edges];

  % all boundary edges of the eggshell
  edg_bd_edges = [edg_id_eg;edg_id_boundary_dom];
  v = Mesh.Edges(edg_bd_edges,:);
  n_v = size(v,1);
  nCoords = size(Mesh.Coordinates,1);

  M = sparse(v(:,1),v(:,2),ones(n_v,1),nCoords,nCoords);

  % adjacency matrix
  M = M + M';
```

```
  % find the first partition of this bipartite graph M
81 id = vid_eg(1);
  part1 = id;
  while true
    [~,x] = find(M(id,:));
    next = setdiff(x,part1);
86
    if isempty(next)
      break;
    end
    id = next(1);
91  part1 = [part1,id];
  end

  % second partition: remaining vertices
  part2 = setdiff(vid_eg,part1);
96
  if isempty(part2) || isempty(part1)
    error('this␣eggshell␣does␣not␣enclose␣the␣body!')
  end

101 % decide which partition is which boundary (inner,outer) of the
      eggshell
  c1 = Mesh.Coordinates(part1(1),:);
  c2 = Mesh.Coordinates(part1(2),:);
  if abs(dist_inner(c1)) < abs(dist_outer(c2))
    Mesh.EggShellCoordsInnerBd = transpose(part1);
106 Mesh.EggShellCoordsOuterBd = transpose(part2);
  else
    Mesh.EggShellCoordsInnerBd = transpose(part2);
    Mesh.EggShellCoordsOuterBd = transpose(part1);
  end
111
end
```

5.2 Dual problem

```
1 function [Lx,Ly] = assemLoad_Dual_LFE(U,Psi,QuadRule,Mesh)
  % [Lx,Ly] = assemLoad_Dual_LFE( U,Psi,QuadRule,Mesh )
  %  Short description: this assembles the RHS of the variational
  %  formulation in thm 6.5.13
  %
6 %       SYNTAX
  %               [Lx,Ly] = assemLoad_Dual_LFE( U,Psi,QuadRule,Mesh )
  %
  %   ARGUMENTS
  %               U,Psi,Fhandle,QuadRule,Mesh =  ...
11 %
  %
  %   DESCRIPTION
  %       Long description:
```

```
%

  nPts = size(QuadRule.w,1);
  nCoordinates = size(Mesh.Coordinates,1);

  Lx = zeros(nCoordinates,1);
  Ly = zeros(nCoordinates,1);

  gN0 = grad_shap_LFE(QuadRule.x);
  gN  = zeros(nPts,6);

  eggshell_elems  = find(Mesh.ElemFlag == -1);
  neggshell_elems = length(eggshell_elems);

  for i = 1:neggshell_elems
    vidx = Mesh.Elements(eggshell_elems(i),:);
    % Compute element mapping

    bK = Mesh.Coordinates(vidx(1),:);
    BK = [Mesh.Coordinates(vidx(2),:)-bK; Mesh.Coordinates(vidx(3),:)-bK
        ];
    det_BK = abs(det(BK));
    inv_BK_t = transpose(inv(BK));

    % transform the gradients
    gN(:,1:2) = gN0(:,1:2)*inv_BK_t;
    gN(:,3:4) = gN0(:,3:4)*inv_BK_t;
    gN(:,5:6) = gN0(:,5:6)*inv_BK_t;

    gU   = U(vidx(1))*gN(:,1:2) + U(vidx(2))*gN(:,3:4) + U(vidx(3))*gN
        (:,5:6);
    gPsi = Psi(vidx(1))*gN(:,1:2) + Psi(vidx(2))*gN(:,3:4) + Psi(vidx(3))
        *gN(:,5:6);

    % Add contributions to global load vector
    for k = 1:3
      idx = 2*k-1;       % column index corresponding to \partial_x of
                         % node k in grad_shap_LFE
      idy = 2*k;         % the same for \partial_y ...

      Lx(vidx(k)) = Lx(vidx(k))-sum(QuadRule.w.*(gN(:,idx).*(gPsi(:,1).*
          gU(:,1) + gPsi(:,2).*gU(:,2)) + ...
                                                gN(:,idy).*(gPsi(:,2).*
                                                    gU(:,1) - gPsi(:,1).*
                                                    gU(:,2))))*det_BK;

      Ly(vidx(k)) = Ly(vidx(k))-sum(QuadRule.w.*(gN(:,idx).*(gPsi(:,1).*
          gU(:,2) - gPsi(:,2).*gU(:,1)) + ...
                                                gN(:,idy).*(gPsi(:,1).*
                                                    gU(:,1) + gPsi(:,2).*
                                                    gU(:,2))))*det_BK;
    end
  end
```

```
end
```

5.3 Force computation

$$F(u) = -\int_\Omega T(u)\cdot\nabla\Psi \,\mathrm{d}x$$

and

$$\begin{aligned} T(u)\cdot\nabla\Psi &= \left(\nabla u\cdot\nabla u^T - \frac{1}{2}\|\nabla u\|_2^2\,\mathbf{I}\right)\cdot\nabla\Psi \\ &= \begin{bmatrix} \frac{1}{2}(\partial_x u)^2 & (\partial_x u\,\partial_y u) \\ (\partial_x u\,\partial_y u) & \frac{1}{2}(\partial_y u)^2 \end{bmatrix}\begin{bmatrix}\partial_x\Psi \\ \partial_y\Psi\end{bmatrix} - \frac{1}{2}\|\nabla u\|_2^2\mathbf{I}\cdot\nabla\Psi \\ &= \begin{bmatrix} \frac{1}{2}\left((\partial_x u)^2 - (\partial_y u)^2\right)\partial_x\Psi + (\partial_x u\partial_y u)\partial_y\Psi \\ \frac{1}{2}\left((\partial_y u)^2 - (\partial_x u)^2\right)\partial_y\Psi + (\partial_x u\partial_y u)\partial_x\Psi \end{bmatrix} \end{aligned}$$

```
function [F] = force_LFE(Mesh,QuadRule,U,Psi)
% [F] = force_LFE( Mesh,QuadRule,U,Phi,varargin )
%  Short description: eggshell formula for force computation
%
%       SYNTAX
%                [F] = force_LFE( Mesh,QuadRule,U,Psi,varargin )
%
%   ARGUMENTS
%                Mesh,QuadRule,U,Psi,varargin =  ...
%                pass an exact function handle of U as varargin
%
%   DESCRIPTION
%       Long description: instead of the FEM solution U a function
%       handle (exact solution) can be passed via varargin, then
%       instead U this function handle is used.

  nPts = size(QuadRule.w,1);
  %nElements = size(Mesh.Elements,1);

  EggShellElems  = find(Mesh.ElemFlag == -1);
  nEggShellElems = size(EggShellElems,1);

  % initialize the return arguments
  F = [0 0];
  gN0 = grad_shap_LFE(QuadRule.x);
  id3 = kron(1:3,ones(nPts,2));

  for i = 1:nEggShellElems
  % $$$   for i = 1:nElements
    vidx = Mesh.Elements(EggShellElems(i),:);
    %  vidx = Mesh.Elements(i,:);

    % Compute element mapping
```

```
    bK = Mesh.Coordinates(vidx(1),:);
    BK = [Mesh.Coordinates(vidx(2),:)-bK; Mesh.Coordinates(vidx(3),:)-bK
        ];

    inv_BK_t = transpose(inv(BK));
    det_BK = abs(det(BK));

    PsiG = Psi(vidx(id3)) .* gN0; % Gradient of Psi
    PsiG(:,1:2) = PsiG(:,1:2)*inv_BK_t;
    PsiG(:,3:4) = PsiG(:,3:4)*inv_BK_t;
    PsiG(:,5:6) = PsiG(:,5:6)*inv_BK_t;

    PsiGx = sum(PsiG(:,1:2:end),2);
    PsiGy = sum(PsiG(:,2:2:end),2);

    UG = U(vidx(id3)) .* gN0; % Gradient of U
    UG(:,1:2) = UG(:,1:2)*inv_BK_t;
    UG(:,3:4) = UG(:,3:4)*inv_BK_t;
    UG(:,5:6) = UG(:,5:6)*inv_BK_t;

    UGx = sum(UG(:,1:2:end),2);
    UGy = sum(UG(:,2:2:end),2);

    % Add contributions to force vector
    F(1) = F(1) + sum(QuadRule.w .* (0.5*(UGx.^2-UGy.^2).*PsiGx + UGx.*
        UGy.*PsiGy))*det_BK;
    F(2) = F(2) + sum(QuadRule.w .* (0.5*(UGy.^2-UGx.^2).*PsiGy + UGx.*
        UGy.*PsiGx))*det_BK;
  end

  F = -F;

end
```

```
function [Psi] = eggshell_psi_LFE(Mesh,A,varargin)
% [phi] = eggshell_psi(Mesh,A)
%  Author: Simon Pintarelli, simonpi@student.ethz.ch
%  Short description
%
%       SYNTAX
%               [phi] = eggshell_psi(Mesh,A)
%
%   ARGUMENTS
%               Mesh,A =  ...
%
%
%   DESCRIPTION
%       Long description:
%

  if isempty(varargin)
    elem_flag = -1;
  else
```

```
    elem_flag = varargin{1};
  end

  FreeDofs = Mesh.Elements(Mesh.ElemFlag==elem_flag,:);
  FreeDofs = setdiff(FreeDofs(:),[Mesh.EggShellCoordsInnerBd;Mesh.
      EggShellCoordsOuterBd]);

  % remove outer boundary
  outer_bd_flag = -1;
  l_id = (Mesh.BdFlags == outer_bd_flag);
  outer_dom_vid = unique(Mesh.Edges(l_id,:));
  FreeDofs = setdiff(FreeDofs,outer_dom_vid);

  nCoordinates = size(Mesh.Coordinates,1);
  Psi = zeros(nCoordinates,1);
  L = zeros(nCoordinates,1);

  Psi(Mesh.EggShellCoordsInnerBd) = 1;
  L = L - A*Psi;
  Psi(FreeDofs) = A(FreeDofs,FreeDofs)\L(FreeDofs);
end
```

5.4 Error estimation

5.4.1 Higher-order method and higher-order interpolation

```
function varargout = ErrEst_Interp(z,U,FHandle,Mesh,QuadRule2D,QuadRule1D
    )
% [Eta] = ErrEst_Interp( z,U,FHandle,Mesh,QuadRule2D,QuadRule1D )
%  Author: Simon Pintarelli, simonpi@student.ethz.ch
%  Short description
%
%       SYNTAX
%               [Eta] = ErrEst_Interp( z,U,FHandle,Mesh,QuadRule2D,
    QuadRule1D )
%
%   ARGUMENTS
%               z,U,FHandle,Mesh,QuadRule2D,QuadRule1D =  ...
%
%
%   DESCRIPTION
%       Long description: Long Description
%


  nElems       = size(Mesh.Elements,1);
  nCoordinates = size(Mesh.Coordinates,1);

  if length(z)==nCoordinates
      z_is_QFE = false;
  else
      z_is_QFE = true;
```

```
  end

  Eta     = zeros(nElems,1);
  Rho     = zeros(nElems,1);
  Weights = zeros(nElems,1);

  nPts    = size(QuadRule2D.x,1);
  nPts_1D = size(QuadRule1D.x,1);

  xe1 = [QuadRule1D.x 1-QuadRule1D.x];
  xe2 = [zeros(nPts_1D,1) QuadRule1D.x];
  xe3 = [QuadRule1D.x zeros(nPts_1D,1)];

  grad_N = grad_shap_LFE(zeros(nPts_1D,2));

  for i = 1:nElems
    vidx = Mesh.Elements(i,:);
    ve(1) = Mesh.Vert2Edge(vidx(1),vidx(2))+nCoordinates;
    ve(2) = Mesh.Vert2Edge(vidx(2),vidx(3))+nCoordinates;
    ve(3) = Mesh.Vert2Edge(vidx(3),vidx(1))+nCoordinates;
    vidx = [vidx,ve];

    % Compute element mapping
    bK     = Mesh.Coordinates(vidx(1),:);
    BK     = [Mesh.Coordinates(vidx(2),:)-bK; Mesh.Coordinates(vidx(3),:)
        -bK];
    det_BK = abs(det(BK));

    inv_BK_t = transpose(inv(BK));
    x = QuadRule2D.x*BK + ones(nPts,1)*bK;

    if  z_is_QFE
      % the adjoint problem was solved using a higher order method (QFE)
      zc2 = z(vidx);
    else
      zc2 = patchwise_interp_QFE(Mesh,z,i);
    end

    zc1 = z(vidx(1:3)); % coefficients for LFE

    % z interpolated to patchwise biquadratic
    zi2 = shap_QFE(QuadRule2D.x)*zc2;
    zi1 = shap_LFE(QuadRule2D.x)*zc1;

    cell_res = sum(QuadRule2D.w.*(FHandle(x) .* (zi2 - zi1)))*det_BK;
    omega    = sum(QuadRule2D.w.*(zi2 - zi1))*det_BK;

    % ---- compute the edge residuals ---- %
    % n is the outward pointing normal vector, abs(n) = edge length
    % edge1 (v2,v3)
    vopp = Mesh.Opp_Vert(i,:);
    if(vopp(1) ~= 0)
      zi1 = shap_LFE(xe1)*zc1;
```

```
76        zi2 = shap_QFE(xe1)*zc2;

          edge = Mesh.Coordinates(vidx(3),:) - Mesh.Coordinates(vidx(2),:);
          n = [edge(2)*ones(nPts_1D,1) -edge(1)*ones(nPts_1D,1)];
          grad_u_this = (U(vidx(1))*grad_N(:,1:2)+ ...
81                       U(vidx(2))*grad_N(:,3:4)+ ...
                         U(vidx(3))*grad_N(:,5:6))*inv_BK_t;

          neighElem = Mesh.Neigh(i,1);
          vidN = Mesh.Elements(neighElem,:);
86        bKN = Mesh.Coordinates(vidN(1),:);
          BKN = [Mesh.Coordinates(vidN(2),:)-bKN; Mesh.Coordinates(vidN(3),:)
              -bKN];

          grad_u_N = (U(vidN(1))*grad_N(:,1:2)+ ...
                      U(vidN(2))*grad_N(:,3:4)+ ...
91                    U(vidN(3))*grad_N(:,5:6))*transpose(inv(BKN));

          edge1  = 0.5*sum(QuadRule1D.w.*sum((grad_u_this - grad_u_N).*n,2)
              .*(zi2-zi1));
          edge1R = 0.5*sum(QuadRule1D.w.*sum((grad_u_this - grad_u_N).*n,2));
        else
96        edge1  = 0;
          edge1R = 0;
        end

        % edge2 (v3,v1)
101     if(vopp(2) ~= 0)
          zi1 = shap_LFE(xe2)*zc1;
          zi2 = shap_QFE(xe2)*zc2;

          edge = Mesh.Coordinates(vidx(1),:) - Mesh.Coordinates(vidx(3),:);
106       n = [edge(2)*ones(nPts_1D,1) -edge(1)*ones(nPts_1D,1)];

          grad_u_this = (U(vidx(1))*grad_N(:,1:2)+ ...
                         U(vidx(2))*grad_N(:,3:4)+ ...
                         U(vidx(3))*grad_N(:,5:6))*inv_BK_t;
111
          neighElem = Mesh.Neigh(i,2);
          vidN = Mesh.Elements(neighElem,:);
          bKN = Mesh.Coordinates(vidN(1),:);
          BKN = [Mesh.Coordinates(vidN(2),:)-bKN; Mesh.Coordinates(vidN(3),:)
              -bKN];
116
          grad_u_N = (U(vidN(1))*grad_N(:,1:2)+ ...
                      U(vidN(2))*grad_N(:,3:4)+ ...
                      U(vidN(3))*grad_N(:,5:6))*transpose(inv(BKN));

121       edge2  = 0.5*sum(QuadRule1D.w .*sum((grad_u_this - grad_u_N).*n,2)
              .*(zi2-zi1));
          edge2R = 0.5*sum(QuadRule1D.w .*sum((grad_u_this - grad_u_N).*n,2))
              ;
        else
```

```
      edge2  = 0;
      edge2R = 0;
    end

    % edge3 (v1,v2)
    if(vopp(3) ~= 0)
      zi1 = shap_LFE(xe3)*zc1;
      zi2 = shap_QFE(xe3)*zc2;

      edge = Mesh.Coordinates(vidx(2),:) - Mesh.Coordinates(vidx(1),:);
      n = [edge(2)*ones(nPts_1D,1) -edge(1)*ones(nPts_1D,1)];
      grad_u_this = (U(vidx(1))*grad_N(:,1:2)+ ...
                     U(vidx(2))*grad_N(:,3:4)+ ...
                     U(vidx(3))*grad_N(:,5:6))*inv_BK_t;

      neighElem = Mesh.Neigh(i,3);
      vidN = Mesh.Elements(neighElem,:);
      bKN = Mesh.Coordinates(vidN(1),:);
      BKN = [Mesh.Coordinates(vidN(2),:)-bKN; Mesh.Coordinates(vidN(3),:)
          -bKN];
      grad_u_N = (U(vidN(1))*grad_N(:,1:2)+ ...
                  U(vidN(2))*grad_N(:,3:4)+ ...
                  U(vidN(3))*grad_N(:,5:6))*transpose(inv(BKN));
      edge3  = 0.5*sum(QuadRule1D.w .*sum((grad_u_this - grad_u_N).*n,2)
          .*(zi2-zi1));
      edge3R = 0.5*sum(QuadRule1D.w .*sum((grad_u_this - grad_u_N).*n,2))
          ;
    else
      edge3  = 0;
      edge3R = 0;
    end
    flux_res = edge1+edge2+edge3;

    Eta(i)     = cell_res + flux_res;
    Weights(i) = abs(omega);
    Rho(i)     = abs(edge1R + edge2R + edge3R);

  end

  if nargout == 3
    varargout{1} = Eta;
    varargout{2} = Rho;
    varargout{3} = Weights;
  else
    varargout{1} = Eta;
  end
end
```

```
function C = patchwise_interp_QFE(mesh,u,elem_index)
% [C] = patchwise_interp_QFE( mesh,u,elem_index )
%  Author: Simon Pintarelli, simonpi@student.ethz.ch

  quadrule = P7O6;
```

```
npts     = size(quadrule.x,1);

vidx = mesh.Elements(elem_index,:);
bK = mesh.Coordinates(vidx(1),:);
BK = [mesh.Coordinates(vidx(2),:)-bK; mesh.Coordinates(vidx(3),:)-bK];
det_BK = abs(det(BK));
inv_BK_t = transpose(inv(BK));

gN = grad_shap_LFE([0,0]);
gN_rec = (u(vidx(1))*gN(:,1:2) + u(vidx(2))*gN(:,3:4) + u(vidx(3))*gN
    (:,5:6))*inv_BK_t*det_BK;

neigh    = mesh.Neigh(elem_index,:);
neigh    = neigh(neigh>0);
adj_elem = setdiff(unique(mesh.AdjElements(vidx,:)),[neigh,0,elem_index
    ]);

n = length(neigh);
if n < 3
  d = 3-n;
  if length(adj_elem) >= d
    neigh = [neigh,adj_elem(1:d)];
  else
    neigh = [neigh,adj_elem];
  end
end

area = det_BK;
% ---------- recover gradients ---------- %
for i = 1:length(neigh)
  vidN = mesh.Elements(neigh(i),:);
  bN   = mesh.Coordinates(vidN(1),:);
  BN   = [mesh.Coordinates(vidN(2),:)-bN; mesh.Coordinates(vidN(3),:)-
      bN];
  inv_BN_t = transpose(inv(BN));
  det_BN = abs(det(BN));
  gloc = (u(vidN(1))*gN(:,1:2) + u(vidN(2))*gN(:,3:4) + u(vidN(3))*gN
      (:,5:6))*inv_BN_t*det_BN;
  gN_rec = gN_rec + gloc;
  area = area + det_BN;
end

gN_rec = ones(npts,1)*gN_rec/area;

% ---------- assemble system ---------- %
Vert = mesh.Coordinates(vidx,:);

M = MASS_QFE(Vert);
S = STIMA_Lapl_QFE(Vert);

gQFE = grad_shap_QFE(quadrule.x);
nQFE = shap_QFE(quadrule.x);
nLFE = shap_LFE(quadrule.x);
```

```
  nU = nLFE*u(vidx);

  for k = 1:6
    idx = 2*k-1;
    idy = 2*k;
    gQFE(:,[idx,idy]) = gQFE(:,[idx,idy])*inv_BK_t;
  end

  L = zeros(6,1);

  L(1) = sum((sum(gQFE(:,1:2).*gN_rec,2)   + nU.*nQFE(:,1)).*quadrule.w)*
      det_BK;
  L(2) = sum((sum(gQFE(:,3:4).*gN_rec,2)   + nU.*nQFE(:,2)).*quadrule.w)*
      det_BK;
  L(3) = sum((sum(gQFE(:,5:6).*gN_rec,2)   + nU.*nQFE(:,3)).*quadrule.w)*
      det_BK;
  L(4) = sum((sum(gQFE(:,7:8).*gN_rec,2)   + nU.*nQFE(:,4)).*quadrule.w)*
      det_BK;
  L(5) = sum((sum(gQFE(:,9:10).*gN_rec,2)  + nU.*nQFE(:,5)).*quadrule.w)*
      det_BK;
  L(6) = sum((sum(gQFE(:,11:12).*gN_rec,2) + nU.*nQFE(:,6)).*quadrule.w)*
      det_BK;

  C = (S+M)\L;
end
```

5.4.2 Approximation by difference quotients

```
function varargout = ErrEst_ApproxDiff(z,U,FHandle,Mesh,QuadRule2D,
    varargin)
% [eta] = ErrEst_ApproxDiff(z,U,FHandle,Mesh,QuadRule2D,QuadRule1D)
%  Author: Simon Pintarelli, simonpi@student.ethz.ch
%  Short description
%
%       SYNTAX
%               [eta] = ErrEst_ApproxDiff(z,U,FHandle,Mesh,QuadRule2D,
    QuadRule1D)
%
%   ARGUMENTS
%               z,U,FHandle,Mesh,QuadRule2D,QuadRule1D
%
%
%   DESCRIPTION
%       Long description: based on ErrEst_RES

  cI = 1.0;
  Rot = [0 -1; 1 0];
  nElems = size(Mesh.Elements,1);
  nEdges = size(Mesh.Edges,1);
  nPts = size(QuadRule2D.x,1);

  grad_N = grad_shap_LFE([0 0]);
```

```
flux_z2 = zeros(nElems,1);
r_h2 = zeros(nElems,1);
R_h2 = zeros(nElems,1);
h_K = zeros(nElems,1);

for i = 1:nEdges
  if(Mesh.BdFlags(i) >= 0)
    P0 = Mesh.Coordinates(Mesh.Edges(i,1),:);
    P1 = Mesh.Coordinates(Mesh.Edges(i,2),:);

    % Compute unit normal and edge length
    normal = P1-P0;
    h_F = norm(normal.^2);
    normal = normal*Rot/h_F;

    % Compute left and right hand side neighbours
    Elem_l = Mesh.Edge2Elem(i,1);
    vidx_l = Mesh.Elements(Elem_l,:);
    Elem_r = Mesh.Edge2Elem(i,2);
    vidx_r = Mesh.Elements(Elem_r,:);

    % Compute element mappings
    bK_l =  Mesh.Coordinates(vidx_l(1),:);
    BK_l = [Mesh.Coordinates(vidx_l(2),:)-bK_l; ...
            Mesh.Coordinates(vidx_l(3),:)-bK_l];
    bK_r =  Mesh.Coordinates(vidx_r(1),:);
    BK_r = [Mesh.Coordinates(vidx_r(2),:)-bK_r; ...
            Mesh.Coordinates(vidx_r(3),:)-bK_r];

    inv_BK_l = inv(BK_l);
    inv_BK_r = inv(BK_r);

    % Compute left and right hand-side gradients
    grad_u_l = (U(vidx_l(1))*grad_N(1:2) + ...
                U(vidx_l(2))*grad_N(3:4) + ...
                U(vidx_l(3))*grad_N(5:6))*transpose(inv_BK_l);
    grad_z_l = (z(vidx_l(1))*grad_N(1:2) + ...
                z(vidx_l(2))*grad_N(3:4) + ...
                z(vidx_l(3))*grad_N(5:6))*transpose(inv_BK_l);

    grad_u_r = (U(vidx_r(1))*grad_N(1:2) + ...
                U(vidx_r(2))*grad_N(3:4) + ...
                U(vidx_r(3))*grad_N(5:6))*transpose(inv_BK_r);
    grad_z_r = (z(vidx_r(1))*grad_N(1:2) + ...
                z(vidx_r(2))*grad_N(3:4) + ...
                z(vidx_r(3))*grad_N(5:6))*transpose(inv_BK_r);

    % Add edge error contributions to left and right hand-side
       neighbours
    fz = h_F*abs(sum((grad_z_l-grad_z_r).*normal,2))^2;
    flux_z2(Elem_l) = flux_z2(Elem_l) + 1/2*fz;
    flux_z2(Elem_r) = flux_z2(Elem_r) + 1/2*fz;
```

```
      fU = h_F*abs(sum((grad_u_l-grad_u_r).*normal,2))^2;
      r_h2(Elem_l) = r_h2(Elem_l) + 1/2*fU;
      r_h2(Elem_r) = r_h2(Elem_r) + 1/2*fU;
    end
  end

  for i = 1:nElems
    vidx = Mesh.Elements(i,:);
    a1 = Mesh.Coordinates(vidx(1),:);
    a2 = Mesh.Coordinates(vidx(2),:);
    a3 = Mesh.Coordinates(vidx(3),:);

    % Compute element mapping
    bK = Mesh.Coordinates(vidx(1),:);
    BK = [Mesh.Coordinates(vidx(2),:)-bK; Mesh.Coordinates(vidx(3),:)-bK
        ];
    det_BK = abs(det(BK));

    h_K(i) = max(sqrt(sum([a1-a2;a2-a3;a3-a1].^2,2)));

    % compute quadrature points
    x = QuadRule2D.x*BK + ones(nPts,1)*bK;

    % compute cell residuals
    R_h2(i) = sum(QuadRule2D.w.*FHandle(x).^2)*det_BK;
  end
  rho_K = sqrt(R_h2 + r_h2./h_K);

  Eta    = cI*h_K.^(3/2).*rho_K.*sqrt(flux_z2);
  Res    = rho_K;
  Weight = h_K.^(3/2).*sqrt(flux_z2);

  if nargout == 3
    varargout{1} = Eta;
    varargout{2} = abs(Res);
    varargout{3} = abs(Weight);
  else
    varargout{1} = Eta;
  end

end
```

5.5 Main

```
function struct_res = run(str_input)

  FHandle = @(x,varargin) zeros(size(x,1),1);      % RHS
                                                   % source term
  QuadRule_1D = gauleg(0,1,5);
```

```
% parameters
theta       = str_input.theta;                  % fraction of refined
    elems where eta > tol/nElems that are refined
MaxDofs     = str_input.MaxDofs;
tol         = str_input.tol;
maxiter     = str_input.maxiter;
H0          = str_input.H0;
DISP        = str_input.DISP;
type        = str_input.type;
estimator   = str_input.estimator;
poly        = str_input.poly;                   % LFE or QFE
plot_on     = str_input.plot_on;
print_on    = str_input.print_on;
print_info  = str_input.print_info;
problem     = str_input.problem;
HHANDLE     = str_input.HHANDLE;
dist_inner  = str_input.dist_inner;             % distance handles used
    to initialize the eggshell
dist_outer  = str_input.dist_outer;             % ....
interpolated_psi = str_input.interpolated_psi; % if true, compute psi
    on coarse grid and interpolate it to the refined meshes

switch poly
  case 'LFE'
    assemDir     = @(Mesh,BDFLAGS,gD) assemDir_LFE(Mesh,BDFLAGS,gD);
    assemMat     = @(Mesh) assemMat_LFE(Mesh,@STIMA_Lapl_LFE);
    assemLoad    = @(Mesh) assemLoad_LFE(Mesh,P1O2,FHandle);
    eggshell_psi = @(Mesh,A) eggshell_psi_LFE(Mesh,A);
    force        = @(Mesh,QuadRule,U,psiegg) force_LFE(Mesh,QuadRule,U,
        psiegg);
    is_qfe       = false;
  case 'QFE'
    assemDir     = @(Mesh,BDFLAGS,gD) assemDir_QFE(Mesh,BDFLAGS,gD);
    assemMat     = @(Mesh) assemMat_QFE(Mesh,@STIMA_Lapl_QFE);
    assemLoad    = @(Mesh) assemLoad_QFE(Mesh,P3O3,FHandle);
    eggshell_psi = @(Mesh,A) eggshell_psi_QFE(Mesh,A);
    force        = @(Mesh,QuadRule,U,psiegg) force_QFE(Mesh,QuadRule,U,
        psiegg);
    is_qfe       = true;
  otherwise
    error('order␣=␣[LFE/QFE]␣\n')
end

switch lower(type)
  case 'adaptive'
    is_adaptive = true;
  case 'uniform'
    is_adaptive = false;
  otherwise
    error('type␣must␣be␣adadptive␣or␣uniform')
end

% ---------- read problem data ---------- %
```

```
gD        = problem.g_D;
init_mesh = problem.init_mesh;
f_exact   = problem.Force;

switch lower(estimator)
  case 'estim1'
    estimator_name = 'dual problem: QFE';
    dual_qfe       = true;
    if is_qfe
      ErrEst       = @ErrEst_Interp_QFE;
    else
      ErrEst       = @ErrEst_Interp;
    end
  case 'estim2'
    estimator_name = 'higher-order interpolation';
    dual_qfe       = false;
    if is_qfe
      ErrEst       = @ErrEst_Interp_QFE;
    else
      ErrEst       = @ErrEst_Interp;
    end
  case  'estim3'
    estimator_name = 'approximate differences';
    dual_qfe       = false | is_qfe;
    ErrEst         = @ErrEst_ApproxDiff;
  case 'estim4'
    estimator_name = 'ErrEst_GOAL';
    dual_qfe       = false | is_qfe;
    ErrEst         = @ErrEst_GOAL;
  otherwise
    error('estimator not available!')
end

% ---------- initialize the mesh ---------- %
[Mesh,DHANDLE] = init_mesh(H0,DISP,HHANDLE,dist_inner,dist_outer);

% ---------- ========= ---------- %
% ---------- main loop ---------- %
% ---------- ========= ---------- %

fprintf(['  problem: '     problem.Name '\n'...
         '  maxiter: %d'   '\n' ...
         '  tol: %.3f' '\n' ...
         '  type: '     type '\n' ...
        ],maxiter,tol);
if is_adaptive
  fprintf(['estimator: '   estimator_name  '\n'])
end

% ---------- compute eggshell psi and create interpolation handle for
    refined meshes ---------- %
if interpolated_psi
  A   = assemMat(Mesh);
```

```
    psi = eggshell_psi(Mesh,A);

    TShandle = TriScatteredInterp(Mesh.Coordinates(:,1),Mesh.Coordinates
        (:,2),psi);
    psi_handle = @(Mesh) interpolate_psi(TShandle,Mesh);
end

F = []; Fds = []; Etax = []; Etay = []; z_x = []; z_y = []; Eta =
    []; Err = []; NDOFS = [];
Meshes = {}; Un = {}; z_xn = {}; z_yn = {}; Etan = {}; Wn = {}; Rn =
    {}; markedn = {};
psin = {}; Lxn = {}; Lyn = {};

for iter=1:maxiter
    nElems = size(Mesh.Elements,1);
    [U,FreeDofs] = assemDir(Mesh,[-1 -2],gD);

    NDOFS(iter) = length(FreeDofs);
    A = assemMat(Mesh);
    L = assemLoad(Mesh);
    L = L - A*U;
    U(FreeDofs) = A(FreeDofs,FreeDofs)\L(FreeDofs);

    Un{iter}     = U;
    Meshes{iter} = Mesh;

    % interpolate psi to refined mesh
    if interpolated_psi
      psi = psi_handle(Mesh);
    else
      psi = eggshell_psi(Mesh,A);
    end

    psin{iter} = psi;
    F(iter,:)   = force(Mesh,P3O3,U,psi);
    Fds(iter,:) = force_ds_LFE(Mesh,U,-2,gauleg(0,1,2));  % integrate
        over boundary
    Err(iter,:) =  F(iter,:)-f_exact;

    if is_adaptive
      % ---------- =============== ---------- %
      % ---------- adjoint problem ---------- %
      % ---------- =============== ---------- %

      if dual_qfe
        % solve biquadratic adjoint problem
        A = assemMat_QFE(Mesh,@STIMA_Lapl_QFE);
        [Lx,Ly] = assemLoad_Dual_QFE(U,psi,P3O3,Mesh);
        [z_x,FreeDofs2] = assemDir_QFE(Mesh,[-1 -2],FHandle);
        [z_y,] = assemDir_QFE(Mesh,[-1 -2],FHandle);

      else
        % solve bilinear adjoint problem
```

```
    [Lx,Ly] = assemLoad_Dual_LFE(U,psi,P3O3,Mesh);
    [z_x,FreeDofs2] = assemDir_LFE(Mesh,[-1 -2],FHandle);
    [z_y,] = assemDir_LFE(Mesh,[-1 -2],FHandle);
  end

  Lxn{iter} = Lx;
  Lyn{iter} = Ly;
  Lx = Lx - A*z_x;
  Ly = Ly - A*z_y;
  z_x(FreeDofs2) = A(FreeDofs2,FreeDofs2)\Lx(FreeDofs2);
  z_y(FreeDofs2) = A(FreeDofs2,FreeDofs2)\Ly(FreeDofs2);

  z_xn{iter} = z_x;
  z_yn{iter} = z_y;

  % ---------- ================ ---------- %
  % ---------- error estimation ---------- %
  % ---------- ================ ---------- %

  [Etax,Rx,Wx] = ErrEst(z_x,U,FHandle,Mesh,P3O3,QuadRule_1D);
  [Etay,Ry,Wy] = ErrEst(z_y,U,FHandle,Mesh,P3O3,QuadRule_1D);

  etas = sqrt(Etax.^2 + Etay.^2);
  Etan{iter} = etas;

  Eta(iter) = sqrt(sum(Etax)^2+sum(Etay)^2);
  Wn{iter} = sqrt(Wx.^2 + Wy.^2);
  Rn{iter} = sqrt(Rx.^2 + Ry.^2);


  if print_info
    fprintf(['*********␣step:␣␣␣␣%d␣\n' ...
             '␣␣␣␣␣␣␣␣␣␣NDOFS:␣␣␣␣%d␣\n' ...
             '␣␣␣␣␣␣␣␣␣␣␣␣Eta:␣␣␣␣%f␣\n' ...
             '␣␣␣␣␣␣norm(Err):␣␣␣␣%f␣\n' ...
            ],iter,NDOFS(iter),Eta(iter),norm(Err(iter,:)))
  end

  % ---------- ================ ---------- %
  % ---------- mesh refinement ---------- %
  % ---------- ================ ---------- %

  q = quantile(etas,0.9);
  marked_elem = find(etas>q*theta);

  % output
  marked              = zeros(nElems,1);
  marked(marked_elem) = 1;
  markedn{iter}       = marked;

  % abort if tolerance reached

  if sum(etas) < tol
```

```
        fprintf('\n tolerance reached , abort \n')
        break;
      end

      % abort if max deg. of freedom reached

      if length(FreeDofs) > MaxDofs
        fprintf('\n abort , resulting system is too large \n')
        break;
      end
      if plot_on
        plotMesh(Mesh,iter,print_on);
      end
      % refine the mesh
      if iter < maxiter
        Mesh = my_refine_LEB(Mesh,marked_elem,DHANDLE);
      end

    else
      % use uniform refinement / no error estimation
      if print_info
        fprintf(['********* step:    %d \n' ...
                 '          NDOFS:    %d \n' ...
                 '      norm(Err):    %f \n' ...
                ],iter,NDOFS(iter),norm(Err(iter,:)))
      end

      % abort if max deg. of freedom reached

      if length(FreeDofs) > MaxDofs
        fprintf('\n abort , resulting system is too large \n')
        break;
      end
      if plot_on
        plotMesh(Mesh,iter,print_on);
      end
      if iter < maxiter
        Mesh = my_refine_REG(Mesh,DHANDLE);
      end
    end
  end % refinements

  struct_res       = struct;

  struct_res.Mesh  = Mesh;
  struct_res.Eta   = Eta;
  struct_res.NDOFS = NDOFS;
  struct_res.F     = F;
  struct_res.Fds   = Fds;
  struct_res.U     = U;
  struct_res.Etax  = Etax;
  struct_res.Etay  = Etay;
  struct_res.Err   = Err;
```

```
  struct_res.z_x    = z_x;
  struct_res.z_y    = z_y;
  struct_res.input  = str_input;
  struct_res.psi    = psi;

  % additional output
  struct_res.Meshes  = Meshes;
  struct_res.markedn = markedn; % stores the elements marked for
      refinement in each step
  struct_res.Un      = Un;
  struct_res.z_xn    = z_xn;
  struct_res.z_yn    = z_yn;
  struct_res.psin    = psin;
  struct_res.Lxn     = Lxn;
  struct_res.Lyn     = Lyn;
  struct_res.Etan    = Etan;    % cell, element indicators for each ref.
  struct_res.Wn      = Wn;      % cell, weights for each ref.
  struct_res.Rn      = Rn;      % cell, residuals for each ref.
end

function [] = plotMesh(Mesh,iter,print_on)
  figure
  plot_Mesh(Mesh,'as')
  title(sprintf('after␣%d␣refinement␣steps',iter))
  if print_on
    filename = sprintf('ref=%d',iter);
    print(fileformat,filename);
  end
  close;
end
```