DUNE PDELab Tutorial 05

Adaptivity in PDELab

Ole Klein

Interdisziplinäres Zentrum für Wissenschaftliches Rechnen
Im Neuenheimer Feld 205, D-69120 Heidelberg

August 31, 2021

Motivation

- Provide a comparatively simple example of adaptive mesh refinement in PDELab
- Build upon problem definition that is already familiar (tutorial 01)
- Integrate central steps into framework that was introduced for the solution of PDEs
- Show where the approach could be extended and modified to suit other PDEs, error norms or performance functionals

Discretization Error

- FEM approach replaces solution space V, e.g. $V = H^1(\Omega)$ plus constraints, with *finite-dimensional* space V_h
- FEM solution $u_h \in V_h$ is approximation of solution $u \in V$
- Finite approximation leads to discretization error, which should be small:

$$\|u - u_h\| \leq \mathrm{TOL}$$

- $\|\cdot\|$ is suitable norm, e.g. L^2 or H^1 norm, TOL is user-supplied tolerance

Central Aspects of Mesh Generation

- Number of degrees of freedom (dofs) important for applicability of method:
  - Directly translates to memory requirements
  - Determines computation time (together with mesh geometry)
- Keep number of dofs as small as possible while fulfilling requirements for error norm $\|u - u_h\|$
- Discretization error $u - u_h$ is generally not known (else there would be no need for FEM!)
- A-priori error estimates are for worst case, i.e. may be overly pessimistic, don't provide spatially resolved information and contain unknown constant

$\Rightarrow$ *A-posteriori error estimates* and iterative procedure required

Derivation of Local Error Indicators

PDE Problem

We consider the problem

$$\begin{aligned} -\Delta u + q(u) &= f && \text{in } \Omega, \\ u &= g && \text{on } \Gamma_D \subseteq \partial\Omega, \\ -\nabla u \cdot \nu &= j && \text{on } \Gamma_N = \partial\Omega \setminus \Gamma_D. \end{aligned}$$

- $q : \mathbb{R} \to \mathbb{R}$ is possibly nonlinear function
- $f : \Omega \to \mathbb{R}$ the source term
- ν unit outer normal to the domain

Weak Formulation

$$\text{Find } u \in U \text{ s.t.:} \quad r^{\mathrm{NLP}}(u,v) = 0 \quad \forall v \in V,$$

with the continuous residual form

$$r^{\mathrm{NLP}}(u,v) = \int_\Omega \nabla u \cdot \nabla v + (q(u) - f)v \, dx + \int_{\Gamma_N} jv \, ds$$

and the function spaces

- $U = \{v \in H^1(\Omega) \,:\, \text{“}v = g\text{” on } \Gamma_D\}$ (affine space)
- $V = \{v \in H^1(\Omega) \,:\, \text{“}v = 0\text{” on } \Gamma_D\}$

We assume that a unique solution exists.

For Derivation: Linear PDE Problem

The presented derivation of local error estimates requires that the PDE is linear. We therefore consider

$$\text{Find } u \in U \text{ s.t.:} \quad r^{\mathrm{LP}}(u, v) = 0 \quad \forall v \in V,$$

with the continuous residual form

$$r^{\mathrm{LP}}(u, v) = \int_\Omega \nabla u \cdot \nabla v + (cu - \tilde{f})v \, dx + \int_{\Gamma_N} jv \, ds$$

i.e. $q(u) = cu$ with a constant $c \in \mathbb{R}$ and a different right hand side $\tilde{f}$, and later return to the original nonlinear PDE.

Discretization Error Identity

Define *discretization error* $e = u - u_h \in V$ and bilinear form

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v + cuv \, dx$$

Then we have, due to linearity of the PDE,

$$\begin{aligned} a(e, v) &= a(u, v) - a(u_h, v) \\ &= r^{LP}(u, v) - r^{LP}(u_h, v) \\ &= -r^{LP}(u_h, v) \end{aligned}$$

This provides an expression that does not depend on u and therefore can be evaluated using the finite element solution u_h!

Element Residuals

$$
\begin{aligned}
a(e,v) &= -r^{LP}(u_h, v) \\
&= -\int_\Omega \nabla u_h \cdot \nabla v + (cu_h - \tilde{f})\,dx - \int_{\Gamma_N} jv\,ds \\
&= -\sum_{T\in\mathcal{T}_h} \left\{ \int_T \nabla u_h \cdot \nabla v + (cu_h - \tilde{f})\,dx - \int_{\partial T\cap\Gamma_N} jv\,ds \right\} \\
&= \sum_{T\in\mathcal{T}_h} \left\{ \int_T R_T v\,dx + \int_{\partial T} R_{\partial T} v\,ds \right\}
\end{aligned}
$$

with *element residuals* R_T and *element boundary residuals* $R_{\partial T}$ given by

$$
\begin{aligned}
R_T &= \Delta u_h + \tilde{f} - cu_h \\
R_{\partial T} &= \begin{cases} -(\nabla u_h)\cdot\nu & \text{on } \partial T \setminus \Gamma_N \\ -(\nabla u_h)\cdot\nu - j & \text{on } \partial T \cap \Gamma_N \end{cases}
\end{aligned}
$$

Face Residuals

There are three types of faces $F \in \mathcal{F}_h$ that contribute to ∂T:

- Interior faces $F \in \mathcal{F}_h^i$, these appear twice in the summation with changing orientation
- Neumann boundary faces $F \in \mathcal{F}_h^N$, these appear once
- Dirichlet boundary faces $F \in \mathcal{F}_h^D$, here v is zero

Define the *face residuals* R_F for faces $F \in \mathcal{F}$ by setting

$$R_F = \begin{cases} R_{\partial T}(T^-) + R_{\partial T}(T^+) = [-(\nabla u_h) \cdot \nu_F] & F \in \mathcal{F}_h^i \\ R_{\partial T}(T^-) = -(\nabla u_h) \cdot \nu_F - j & F \in \mathcal{F}_h^N \end{cases}$$

where T^- and T^+ are the elements next to F, ν_F points from T^- to T^+ and $[\cdot]$ is the jump operator for tw-valued functions on F, i.e.

$$[v] = v(T^-) - v(T^+)$$

Discretization Error Identity (cont.)

Using the element residuals R_T and face residuals R_F, we have

$$a(e, v) = \sum_{T \in \mathcal{T}_h} \int_T R_T v \, dx + \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^N} \int_F R_F v \, ds$$

For any interpolation operator $\mathcal{I} \colon V \to V_h$ we also have

$$a(e, \mathcal{I}v) = \sum_{T \in \mathcal{T}_h} \int_T R_T \mathcal{I}v \, dx + \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^N} \int_F R_F \mathcal{I}v \, ds = 0$$

(u_h is discrete solution!), and therefore

$$a(e, v) = \sum_{T \in \mathcal{T}_h} \int_T R_T (v - \mathcal{I}v) \, dx + \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^N} \int_F R_F (v - \mathcal{I}v) \, ds$$

Discretization Error Identity (cont.)

Using

- A specific choice of interpolation operator
- Matching interpolation error estimates (independent of problem definition!)
- Shape regularity of the finite element mesh

one can show that

$$
\begin{aligned}
a(e, v) &= \sum_{T \in \mathcal{T}_h} \int_T R_T (v - \mathcal{I} v)\, dx + \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^N} R_F (v - \mathcal{I} v)\, ds \\
&\leq C \|v\|_{1,\Omega} \left\{ \sum_{T \in \mathcal{T}_h} h_T^2 \|R_T\|_{0,T}^2 + \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^N} h_F \|R_F\|_{0,F}^2 \right\}^{1/2}
\end{aligned}
$$

Error Estimate

Set $v = e \in V$ and exploit coercivity $\|e\|_{1,\Omega}^2 \leq Ca(e,e)$, then

$$\begin{aligned} \|e\|_{1,\Omega} &\leq C \left\{ \sum_{T \in \mathcal{T}_h} h_T^2 \|R_T\|_{0,T}^2 + \sum_{F \in \mathcal{F}_h^i \cup \mathcal{F}_h^N} h_F \|R_F\|_{0,F}^2 \right\}^{1/2} \\ &\leq C \left\{ \sum_{T \in \mathcal{T}_h} \gamma_T^2 \right\}^{1/2} \end{aligned}$$

with the *local error indicators*

$$\gamma_T^2 = h_T^2 \|R_T\|_{0,T}^2 + \sum_{F \in \partial T \cap \mathcal{F}_h^N} h_T \|R_F\|_{0,F}^2 + \sum_{F \in \partial T \cap \mathcal{F}_h^i} \frac{h_T}{2} \|R_F\|_{0,F}^2$$

Return to nonlinear PDE problem

For the original nonlinear PDE, linearize residual form around $\xi \in V_h$ and set

$$c = \frac{\partial q}{\partial u}|_{\xi}, \quad \tilde{f} = f - q(\xi) + \frac{\partial q}{\partial u}|_{\xi}\xi$$

The choice $\xi = u_h$ provides face residuals as before and element residuals

$$R_T = \Delta u_h + f - q(u_h)$$

This can be used to compute local error indicators, but the error inequality only holds if u_h is sufficiently close to u!

Local Mesh Adaptation

Basic Adaptation Algorithm

The basic algorithm works as follows:

1. Choose sufficiently fine starting mesh $\mathcal{T}_0$
2. Compute finite element solution u_h on current mesh $\mathcal{T}_h$
3. Compute error estimate $\gamma(u_h)$, stop if $\gamma(u_h) \leq \mathsf{TOL}$
4. Else refine mesh according to the local error indicators γ_T
5. Transfer current solution u_h and use as initial guess
6. Go to step 2)

Bulk Fraction Strategy

- Step 4) requires picking elements for refinement
- Assumption: spatial distribution of error is similar to that of assembled residuals R_T and R_F (reasonable for diffusion-type problems)
- Sort elements according to increasing error contribution:

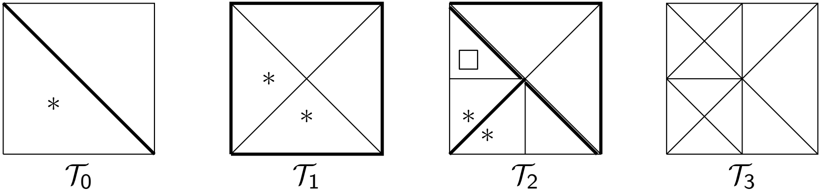
$$\gamma_{T_1}^2 \leq \gamma_{T_2}^2 \leq \cdots \leq \gamma_{T_N}^2$$

- For given $\rho \in (0, 1]$, determine

$$J = \max \left\{ j \colon \sum_{k=j}^{N} \gamma_{T_k}^2 \geq \rho \sum_{T \in \mathcal{T}_h} \gamma_T^2 \right\}$$

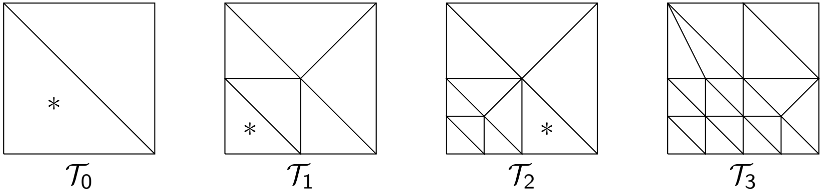
and refine elements $T_J, \ldots, T_N$

Bisection Refinement



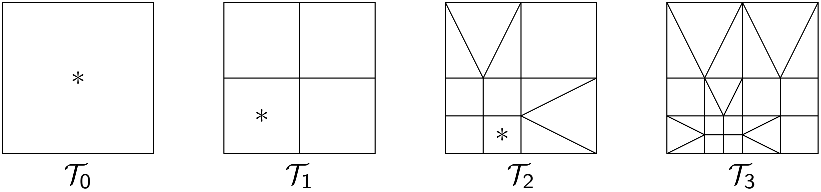
- Refine by cutting element in two (use newest edge)
- Is simple ($*$), but may lead to substantial non-local changes of the mesh ($\mathcal{T}_2 \to \mathcal{T}_3$,$\square$)

Regular Refinement



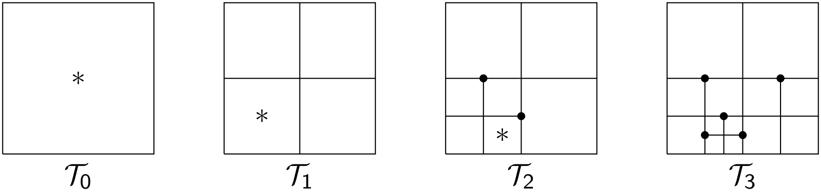
- Refine by dividing local mesh width h_T by two, produces smaller copies of original element
- Requires bisection on the fringe to keep mesh conforming
- Shape regularity requires removal of bisection refinement in subsequent iterations ($\mathcal{T}_2 \to \mathcal{T}_3$)

Refinement of Quadrilaterals



- Regular refinement with conforming closure can be used with quadrilaterals
- Requires using triangular elements for the closure
- Hybrid mesh, no longer one universal reference element

Hanging Nodes



- Omitting closure keeps refinement local
- Straightforward and can also be used with triangles
- Resulting hanging nodes add constraints to the finite element space, i.e. algorithmic complexity is shifted from mesh generation to assembly procedure

Implementation in DUNE/PDELab

Overview DUNE/PDELab Implementation

Files involved are:

1) File `tutorial05.cc`
   - Includes C++, DUNE and PDELab header files
   - Contains the `main` function
   - Creates a finite element mesh and calls the `driver`
2) File `tutorial05.ini`
   - Contains parameters controlling the execution
3) File `driver.hh`
   - Function `driver` iteratively solving the finite element problem and refining the mesh based on the calculated error estimate
4) File `nonlinearpoissonfem.hh`
   - Class `NonlinearPoissonFEM` realizing the necessary element-local computations for the PDE
5) File `nonlinearpoissonfemestimator.hh`
   - Class `NonlinearPoissonFEMEstimator` realizing the necessary element-local computations for the error estimate

Now lets go to the code ...