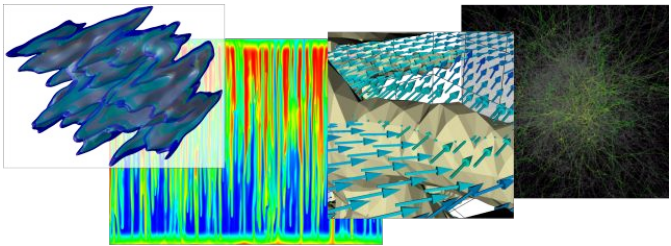
DUNE PDELab Tutorial 00

An Introduction to the Finite Element Method



Speaker:

Peter Bastian
IWR
Heidelberg University

Motivation

- Start with an introduction to the finite element method (FEM) for solving Poisson's equation with piecewise linear "P_1" finite elements
- In particular the weak formulation of PDEs is the key abstraction
- "Hello World!" for any numerical partial differential equation (PDE) solver framework!
- Gives necessary background for `dune-grid` and `dune-pdelab` module
- We will implement the P_1 FEM in PDELab on Wednesday ...

Challenges for PDE Software

- **Many different PDE applications**
  - Multi-physics
  - Multi-scale
  - Inverse modeling: parameter estimation, optimal control
  - Uncertainty quantification, data assimilation, ...
- **Many different numerical solution methods**
  - No single method to solve all PDEs!
  - Different mesh types, mesh generation, mesh refinement
  - Higher-order approximations (polynomial degree)
  - Error control and adaptive mesh/degree refinement
  - Iterative solution of (non-)linear algebraic equations
- **High-performance Computing**
  - Single core performance: bandwidth vs. compute-bound
  - Parallelization through domain decomposition
  - Robustness w.r.t. to mesh size, model parameters, processors
  - Dynamic load balancing

→ **One flexible software to do it all!**

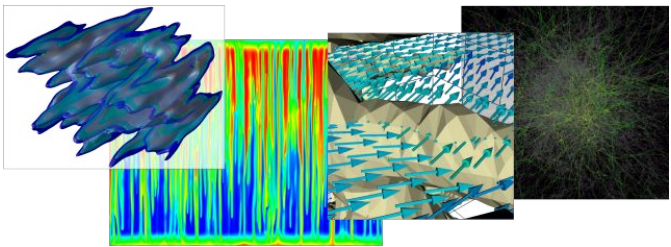
Flexibility Requires Abstraction!

- DUNE/PDELab is based on formulating numerical schemes as **residual forms**
- In order to implement a scheme it requires to put it to that form!
- Although you might be familiar with the FEM, you might not be familiar to the notation used here
- When you have mastered the abstraction you can solve complex problems with reasonable effort
- Still, there will be some learning curve ...
- Important feature: Orthogonality of concepts:
  - Dimension $d = 1, 2, 3, \ldots$
  - Linear and nonlinear
  - Stationary and Instationary
  - Scalar PDE and systems of PDEs
  - Uniform and adaptive mesh refinement of different types
  - Sequential and parallel

→ Introduce one feature at a time!

DUNE PDELab Tutorial 00

(Part 1: Freshup of the Finite Element Method)



Speaker:

Peter Bastian
IWR
Heidelberg University

Strong Formulation of the PDE Problem

We solve Poisson's equation with inhomogeneous Dirichlet boundary conditions:

$$-\Delta u = f \quad \text{in } \Omega, \tag{1a}$$

$$u = g \quad \text{on } \partial\Omega, \tag{1b}$$

- $\Omega \subset \mathbb{R}^d$ is a polygonal domain in d-dimensional space
- Domains are open and connected sets of points
- A function $u \in C^2(\Omega) \cap C^0(\overline{\Omega})$ solving (1a), (1b) is called *strong solution*
- Inhomogeneous Dirichlet boundary conditions could be reduced to *homogeneous* ones: we will not do this!
- Proving existence and uniqueness of solutions of strong solutions requires quite restrictive conditions on f and g

Weak Formulation of the PDE Problem

Let u be a strong solution, take *a test function* $v \in C^1(\Omega) \cap C^0(\overline{\Omega})$, $v = 0$ on $\partial\Omega$, then:

$$\int_\Omega (-\Delta u) v \, dx = \int_\Omega (-\nabla \cdot \nabla u) v \, dx = \underbrace{\int_\Omega \nabla u \cdot \nabla v \, dx}_{=:a(u,v)} + \underbrace{\int_{\partial\Omega} (-\nabla u) \cdot \nu \, v \, ds}_{=0 (v|_{\partial\Omega}=0)} = \underbrace{\int_\Omega f v \, dx}_{=:l(v)}.$$

bilinear form

linear form

Question: Is there a vector space of functions V with $V_g = \{v \in V : v = g \text{ on } \partial\Omega\}$ and $V_0 = \{v \in V : v = 0 \text{ on } \partial\Omega\}$ such that the problem

$$u \in V_g : \qquad a(u, v) = l(v) \qquad \forall v \in V_0 \tag{2}$$

V, V_g, V_0

has a unique solution?

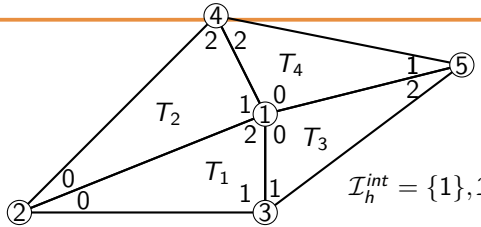
Answer: Yes, $V = H^1(\Omega)$. This u is called *weak solution*.

Advantage: Weak solutions do exist under less restrictive conditions on the data.

The Finite Element Method

- The finite element method (FEM) is one class of methods for the numerical solution of PDEs
- Others are the finite volume method (FVM) or the finite difference method (FDM)
- FEMs are based on a weak formulation of the PDE
- In this lecture we will focus on the *conforming* finite element method
- Its basic idea is to replace the space (infinite dimensional) space V by a *finite-dimensional space* $V_h \subset V$
- The construction of these finite-dimensional spaces needs some preparations ...

Finite Element Mesh



$g_{T_1}(0) = 2, \quad g_{T_1}(1) = 3, \quad g_{T_1}(2) = 1,$
$g_{T_2}(0) = 2, \quad g_{T_2}(1) = 1, \quad g_{T_2}(2) = 4,$
$g_{T_3}(0) = 1, \quad g_{T_3}(1) = 3, \quad g_{T_3}(2) = 5,$
$g_{T_4}(0) = 1, \quad g_{T_4}(1) = 5, \quad g_{T_4}(2) = 4.$

$\mathcal{I}_h^{int} = \{1\}, \mathcal{I}_h^{\partial\Omega} = \{2,3,4,5\}$

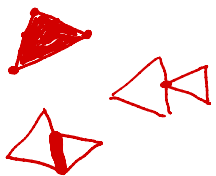
- A mesh consists of *ordered* sets of vertices and elements:

$$\mathcal{X}_h = \{x_1, \ldots, x_N\} \subset \mathbb{R}^d, \quad \mathcal{T}_h = \{T_1, \ldots, T_M\}$$

N=5
M=4

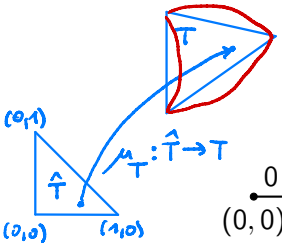
- *Simplicial element*: $T = \text{interior}(\text{convex\_hull}(x_{T,0}, \ldots, x_{T,d}))$
- *Conforming*: Intersection of boundary is subentity
- *Local to global map* : $g_T : \{0, \ldots, d\} \to \mathcal{I}_h$

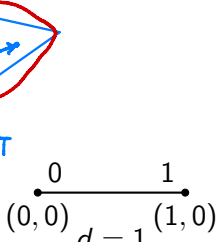
$$\forall T \in \mathcal{T}_h, 0 \le i \le d : g_T(i) = j \Leftrightarrow x_{T,i} = x_j.$$

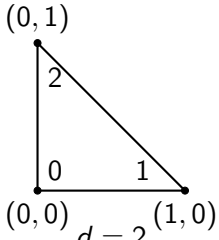


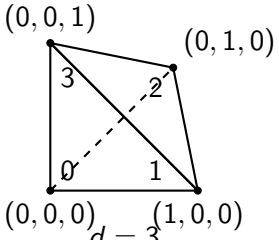
- *Interior and boundary vertex index sets*: $\mathcal{I}_h = \mathcal{I}_h^{int} \cup \mathcal{I}_h^{\partial\Omega}$,
$\mathcal{I}_h^{int} = \{i \in \mathcal{I}_h : x_i \in \Omega\}, \mathcal{I}_h^{\partial\Omega} = \{i \in \mathcal{I}_h : x_i \in \partial\Omega\}$

Reference Element and Element Transformation









- $\hat{T}^d$ is the reference simplex in d space dimensions
- The mesh $\mathcal{T}_h$ is called *affine* if for every $T \in \mathcal{T}_h$ there is an affine linear map $\mu_T : \hat{T} \to T$,

$$\mu_T(\hat{x}) = B_T \hat{x} + b_T$$

with

$$\forall i \in \{0, \ldots, d\} : \mu_T(\hat{x}_i) = x_{T,i}$$

Piecewise Linear Finite Element Space

- The idea of the *conforming* FEM is to solve the weak problem in a *finite-dimensional* function space $V_h \subset V$ used in the weak form:

$$u_h \in V_{h,g}: \quad a(u_h, v) = l(v) \quad \forall v \in V_{h,0}.$$

- A particular choice is the space of *piecewise linear* functions

$$V_h(\mathcal{T}_h) = \{v \in C^0(\overline{\Omega}) : \forall T \in \mathcal{T}_h : v|_T \in \mathbb{P}_1^d\}$$

where $\mathbb{P}_1^d = \{p : \mathbb{R}^d \to \mathbb{R} : p(x) = q^T x + r, q \in \mathbb{R}^d, r \in \mathbb{R}\}$ $d=2$ $p(x_1,x_2) = q_1^T x_1 + q_2^T x_2 + r^T$

- One can show $\dim V_h = N = \dim \mathcal{X}_h$ and $V_h \subset H^1(\Omega)$
- *Lagrange* basis functions: $v \in V_h : v(x) = \sum_{j=1}^{N} (z)_j \phi_j(x)$

$$\Phi_h = \{\phi_1, \ldots, \phi_N\}, \quad \forall i,j \in \mathcal{I}_h : \phi_i(x_j) = \delta_{i,j} = \begin{cases} 1 & i=j \\ 0 & \text{else} \end{cases}$$

- *Test and Ansatz spaces*:

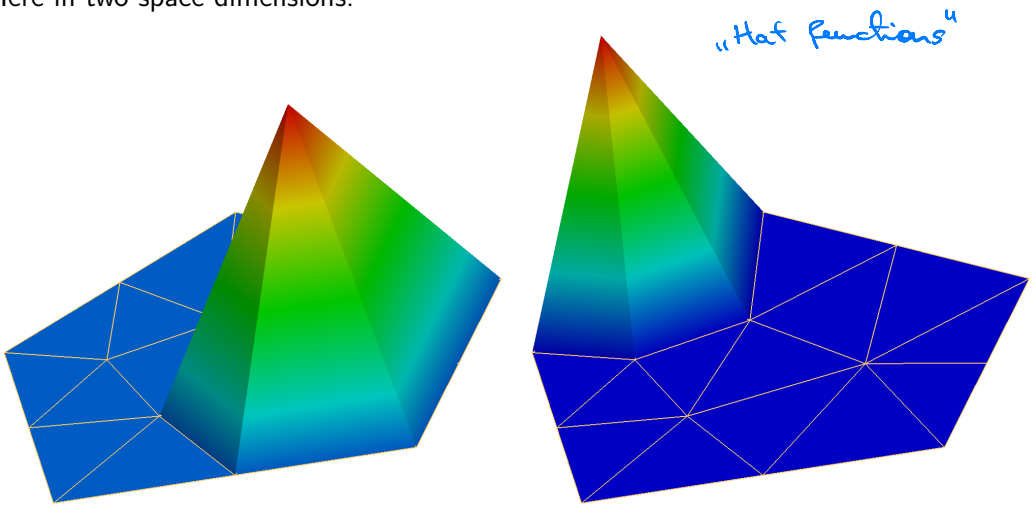
V_h → $V_{h,g}$, $V_{h,0}$

$$V_{h,0} = \{v \in V_h : \forall i \in \mathcal{I}_h^{\partial\Omega} : v(x_i) = 0\}, \qquad \text{(vector space)}$$

$$V_{h,g} = \{v \in V_h : \forall i \in \mathcal{I}_h^{\partial\Omega} : v(x_i) = g(x_i)\} = v_{h,g} + V_{h,0} \qquad \text{(affine shifted space)}$$

Examples of Finite Element Functions

Here in two space dimensions:



Computing the Finite Element Solution

Inserting a *basis representation* $u_h = \sum_{j=1}^{N} (z)_j \phi_j$ results in

$$a(u_h, v) = l(v) \quad \forall v \in V_{h,0} \qquad \text{(discrete weak problem)},$$

$$\Leftrightarrow a\left(\sum_{j=1}^{N} (z)_j \phi_j, \phi_i\right) = l(\phi_i) \quad \forall i \in \mathcal{I}_h^{int} \qquad \text{(insert basis, linearity)},$$

$$\Leftrightarrow \sum_{j=1}^{N} (z)_j a(\phi_j, \phi_i) = l(\phi_i) \quad \forall i \in \mathcal{I}_h^{int} \qquad \text{(linearity)}.$$

$(Az)_i$

Together with the condition $u_h \in V_{h,g}$ expressed as

$$u_h(x_i) = z_i = g(x_i) \quad \forall i \in \mathcal{I}_h^{\partial\Omega}$$

this forms a system of linear equations

$$Az = b, \qquad (A)_{i,j} = \begin{cases} a(\phi_j, \phi_i) & i \in \mathcal{I}_h^{int} \\ \delta_{i,j} & i \in \mathcal{I}_h^{\partial\Omega} \end{cases}, \quad (b)_i = \begin{cases} l(\phi_i) & i \in \mathcal{I}_h^{int} \\ g(x_i) & i \in \mathcal{I}_h^{\partial\Omega} \end{cases}.$$

$N \times N$ matrix

Solution of Linear Systems

- *Exact* solvers based on Gaussian elimination
- This may become inefficent for *sparse* linear systems
- *Iterative* methods (hopefully) produce a convergent sequence

$$\lim_{k\to\infty} z^k = z$$

- A very simple example is *Richardson's* iteration:

$$z^{k+1} = z^k + \omega(b - Az^k), \qquad \mathbb{R} \ni \omega > 0,$$

  requiring only *matrix-vector products*
- Krylov methods require also only matrix-vector products
- Larger and/or difficult problems require efficient *preconditioners*

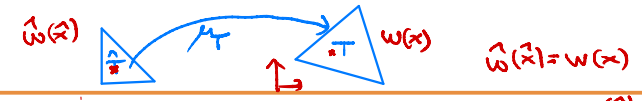
Three Steps to Solve the FE Problem

1. Assembling the matrix A. This means the computation of the matrix entries $a(\phi_j, \phi_i)$ and storing them in an appropriate data structure
2. Assembling the right hand side vector b. This means evaluating the right hand side functional $l(\phi_i)$
3. Solve the linear system $Az = b$ using direct or iterative methods
4. *Variant:* Perform a matrix free operator evaluation $y = Az$ within an iterative solver. This involves evaluations of $a(u_h, \phi_i)$ for all test functions ϕ_i:

$$(Ax)_i = \sum_{j=1}^{N} (A)_{i,j}(x)_j = \sum_{j=1}^{N} a(\phi_j, \phi_i)(x)_j = a\left(\sum_{j=1}^{N} (x)_j \phi_j, \phi_i\right) = a(v_h, \phi_i)$$

We now discuss *how* these steps are implemented

Four Important Tools



1. Transformation formula for integrals. For $T \in \mathcal{T}_h$:

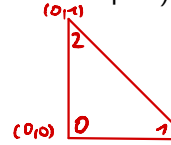
$$\int_T y(x)\,dx = \int_{\hat{T}} y(\mu_T(\hat{x}))|\det B_T|\,d\hat{x}, \quad \text{where } x = \mu_T(\hat{x}) = B_T\hat{x} + b_T$$

2. Midpoint rule on the reference element:

$$\int_{\hat{T}} q(\hat{x})\,dx \approx q(\hat{S}_d)w_d, \qquad (\hat{S}_d\text{: center of } d\text{-simplex})$$

(More accurate formulas are used later)

3. Basis functions via shape function transformation:



$$\hat{\phi}_0(\hat{x}) = 1 - \sum_{i=1}^{d}(\hat{x})_i, \quad \hat{\phi}_i(\hat{x}) = (\hat{x})_i, i > 0, \quad \phi_{T,i}(\mu_T(\hat{x})) = \hat{\phi}_i(\hat{x})$$

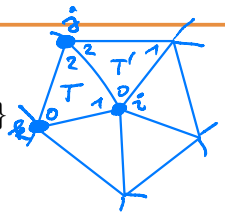
4. Computation of gradients. For any $w(\mu_T(\hat{x})) = \hat{w}(\hat{x})$:

$$B_T^T \nabla w(\mu_T(\hat{x})) = \hat{\nabla}\hat{w}(\hat{x}) \quad \Leftrightarrow \quad \nabla w(\mu_T(\hat{x})) = B_T^{-T}\hat{\nabla}\hat{w}(\hat{x}).$$

Assembly of Right Hand Side I

In computing $(b)_i$ only the following elements are involved:

$$C(i) = \{(T, m) \in \mathcal{T}_h \times \{0, \ldots, d\} \,:\, g_T(m) = i\}$$



Then

$$\begin{aligned}
(b)_i = l(\phi_i) &= \int_\Omega f\phi_i \, dx && \text{(definition)} \\
&= \sum_{T \in \mathcal{T}_h} \int_T f\phi_i \, dx && \text{(use mesh)} \\
&= \sum_{(T,m) \in C(i)} \int_{\hat{T}} f(\mu_T(\hat{x}))\hat{\phi}_m(\hat{x}) |\det B_T| \, d\hat{x} && \text{(localize)} \\
&= \sum_{(T,m) \in C(i)} f(\mu_T(\hat{S}_d))\hat{\phi}_m(\hat{S}_d) |\det B_T| w_d + \text{err.} && \text{(quadrature)}
\end{aligned}$$

Assembly of Right Hand Side II

- Now we need to perform these computations *for all* $i \in \mathcal{I}_h^{int}$!
- Collect *element-local* computations:

$$(b_T)_m = f(\mu_T(\hat{S}_d))\hat{\phi}_m(\hat{S}_d)|\det B_T|w_d \quad \forall m = 0, \dots, d$$

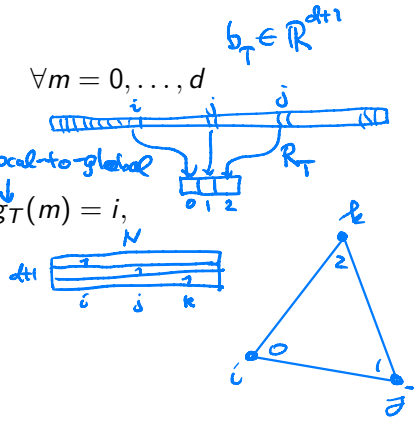
- Define restriction matrix $R_T : \mathbb{R}^N \to \mathbb{R}^{d+1}$ with

"picking out matrix"

$$(R_T x)_m = (x)_i \quad \forall\, 0 \le m \le d,\ g_T(m) = i,$$

- Then

$$b = \sum_{T \in \mathcal{T}_h} R_T^T b_T.$$



Assembly of Global Stiffness Matrix I

In computing $(A)_{i,j}$ only the following elements are involved:

$$C(i,j) = \{(T,m,n) \in \mathcal{T}_h \times \{0,\dots,d\} \,:\, g_T(m) = i \wedge g_T(n) = j\}$$

Then

$$\begin{aligned}
(A)_{i,j} = a(\phi_j,\phi_i) &= \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, dx && \text{(definition)}\\
&= \sum_{T\in\mathcal{T}_h} \int_T \nabla\phi_j \cdot \nabla\phi_i \, dx && \text{(use mesh)}\\
&= \sum_{(T,m,n)\in C(i,j)} \int_{\hat{T}} (B_T^{-T}\hat{\nabla}\hat{\phi}_n(\hat{x})) \cdot (B_T^{-T}\hat{\nabla}\hat{\phi}_m(\hat{x}))|\det B_T| \, d\hat{x} && \text{(localize)}\\
&= \sum_{(T,m,n)\in C(i,j)} (B_T^{-T}\hat{\nabla}\hat{\phi}_n(\hat{S}_d)) \cdot (B_T^{-T}\hat{\nabla}\hat{\phi}_m(\hat{S}_d))|\det B_T| w_d. && \text{(quadrature)}
\end{aligned}$$

no error involved!

Assembly of Global Stiffness Matrix II

- Now we need to perform these computations for *all* matrix entries!
- Define the $d \times d+1$ matrix of shape function gradients

$$\hat{G} = \left[\hat{\nabla}\hat{\phi}_0(\hat{S}_d)), \ldots, \hat{\nabla}\hat{\phi}_d(\hat{S}_d))\right].$$

and the matrix of transformed gradients

$$G = B_T^{-T}\hat{G}$$

- Define the *local stiffness matrix*

$$\mathbb{R}^{d+1 \times d+1} \ni A_T = G^T G |\det B_T| w_d.$$

- Then

$$A = \sum_{T \in \mathcal{T}_h} R_T^T A_T R_T.$$

never formed explicitly!

Matrix-free Operator Evaluation

- Similar considerations apply for the operation $y = Az$
- Pick out the coefficients on the element T:

$$z_T = R_T z$$

- Perform the *element-local computation*:

$$y_T = |\det B_T| w_d G^T G z_T$$

- Accumulate the results:

$$Az = \sum_{T \in \mathcal{T}_h} R_T^T y_T.$$

Implementation Summary

- All necessary steps in the solution procedure have the following general form:

```
1: for T ∈ T_h do                                   ▷ loop over mesh elements
2:     z_T = R_T z                                  ▷ load element data
3:     q_T = compute(T, z_T)                        ▷ element local computations
4:     Accumulate(q_T)                              ▷ store result in global data structure
5: end for
```

- PDELab provides a generic *assembler* that performs all these steps, except (3) which needs to be supplied by the implementor of a FEM
- All these concepts carry over to
  - Nonlinear problems
  - Time-dependent problems
  - Systems of PDEs
  - High-order methods
  - Other schemes such as FVM, nonconforming FEM
  - Parallel computations

Residual Forms

$a(u_h, v) = l(v) \quad \forall v \in V_h$

$\Leftrightarrow \quad a(u_h, v) - l(v) = 0 \quad \forall v \in V_h$

$r(u_h, v) = 0$

- The FEM based on the weak formulation may equivalently be written as

$$\text{Find } u_h \in U_h \text{ s.t.:} \quad r_h^{\text{Poisson}}(u_h, v) = 0 \quad \forall v \in V_h.$$

  where $r^{\text{Poisson}}(u_h, v) = a(u_h, v) - l(v)$ is the **residual form**
- This residual form is *affine linear* in u_h and *linear* in v
- A *nonlinear* PDE results in a residual form $r(u, v)$ that is *nonlinear* in its first argument
- In that sense a linear problem is only a special case
- Residual forms are always linear in the second argument due to linearity of the integral
- **PDELab uses the concept of a residual form as its main abstraction!**

Generalization

- More complicated discretization schemes:

$$r(u,v) = \sum_{T\in\mathcal{T}_h} \alpha_T^V(R_T u, R_T v) + \sum_{T\in\mathcal{T}_h} \lambda_T^V(R_T v) + \sum_{F\in\mathcal{F}_h^i} \alpha_F^S(R_{T_F^-}u, R_{T_F^+}u, R_{T_F^-}v, R_{T_F^+}v) + \sum_{F\in\mathcal{F}_h^{\partial\Omega}} \alpha_F^B(R_{T_F^-}u, R_{T_F^-}v) + \sum_{F\in\mathcal{F}_h^{\partial\Omega}} \lambda_F^B(R_{T_F^-}v).$$

- Instationary problems: Find $u_h(t) \in U_h$ s.t.:

$$d_t m_h(u_h(t), v; t) + r_h(u_h(t), v; t) = 0 \quad \forall v \in V_h$$

- Systems of PDEs: Find $u_h \in U_h = U_h^1 \times \ldots \times U_h^s$ s.t.:

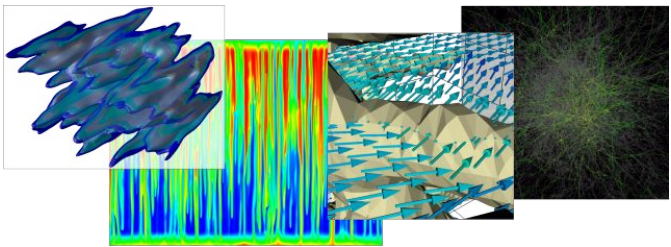
$$r_h(u_h, v) = 0 \quad \forall v \in V_h = V_h^1 \times \ldots \times V_h^s$$

Literature

1. Bastian, P., Blatt, M., Dedner, A. et al. *A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework.* Computing 82, 103–119 (2008). https://doi.org/10.1007/s00607-008-0003-x
2. Bastian, P., Blatt, M., Dedner, A. et al. *A generic grid interface for parallel and adaptive scientific computing. Part II: implementation and tests in DUNE.* Computing 82, 121–138 (2008). https://doi.org/10.1007/s00607-008-0004-9
3. Peter Bastian, Markus Blatt, Andreas Dedner, Nils-Arne Dreier, Christian Engwer, René Fritze, Carsten Gräser, Christoph Grüninger, Dominic Kempf, Robert Klöfkorn, Mario Ohlberger, Oliver Sander, *The Dune framework: Basic concepts and recent developments*, Computers & Mathematics with Applications, Volume 81, 2021, Pages 75-112, https://doi.org/10.1016/j.camwa.2020.06.007.
4. Oliver Sander, *DUNE — The Distributed and Unified Numerics Environment*, Lecture Notes in Computational Science and Engineering, 140, Springer-Verlag, 2020, https://doi.org/10.1007/978-3-030-59702-3

DUNE PDELab Tutorial 00

(Part 2: Implementation in DUNE/PDELab)



Speaker:

Peter Bastian
IWR
Heidelberg University

The PDE Problem Revisited

We solve Poisson's equation with inhomogeneous Dirichlet boundary conditions:

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega \\ u &= g \qquad \text{on } \partial\Omega \end{aligned}$$

The weak formulation is

$$u \in V_g : \qquad a(u, v) = l(v) \qquad \forall v \in V_0$$

with

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, dx \quad \text{and} \quad l(v) = \int_\Omega fv \, dx$$

and

$$V_0 = H_0^1(\Omega)$$
$$V_g = \{v \in H^1(\Omega) \, : \, v = u_g + w \wedge u_g | \Gamma_D = g \wedge w \in V_0\}$$

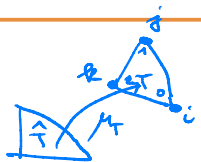
$H^1(\Omega)$

Generic Assembly Loop

1: **for** $T \in \mathcal{T}_h$ **do** ▷ loop over mesh elements
2: $z_T = R_T z$ ▷ load element data
3: $q_T = \text{compute}(T, z_T)$ ▷ element local computations
4: Accumulate(q_T) ▷ store result in global data structure
5: **end for**

Only the computational kernels compute(T, z_T) need to be implemented by the user to implement the finite element method

Assembly of Right Hand Side



- Now we need to perform these computations *for all* $i \in \mathcal{I}_h^{int}$!
- Collect *element-local* computations:

$$(b_T)_m = f(\mu_T(\hat{S}_d))\hat{\phi}_m(\hat{S}_d)|\det B_T|w_d \quad \forall m = 0, \dots, d$$

- Define destriction matrix $R_T : \mathbb{R}^N \to \mathbb{R}^{d+1}$ with

$$\mu_T(\hat{x}) = B_T\hat{x} + b_T$$

$$(R_T x)_m = (x)_i \quad \forall\, 0 \le m \le d,\, g_T(m) = i,$$

- Then

$$b = \sum_{T \in \mathcal{T}_h} R_T^T b_T.$$

Assembly of Global Stiffness Matrix

- Define the $d \times d+1$ matrix of shape function gradients

$$\hat{G} = \left[\hat{\nabla}\hat{\phi}_0(\hat{S}_d)), \ldots, \hat{\nabla}\hat{\phi}_d(\hat{S}_d))\right].$$

  and the matrix of transformed gradients

$$G = B_T^{-T}\hat{G}$$

- Define the *local stiffness matrix*

$$A_T = G^T G |\det B_T| w_d.$$

- Then

$$A = \sum_{T \in \mathcal{T}_h} R_T^T A_T R_T.$$

Matrix-free Operator Evaluation

- Similar considerations apply for the operation $y = Az$
- Pick out the coefficients on the element T:

$$z_T = R_T z$$

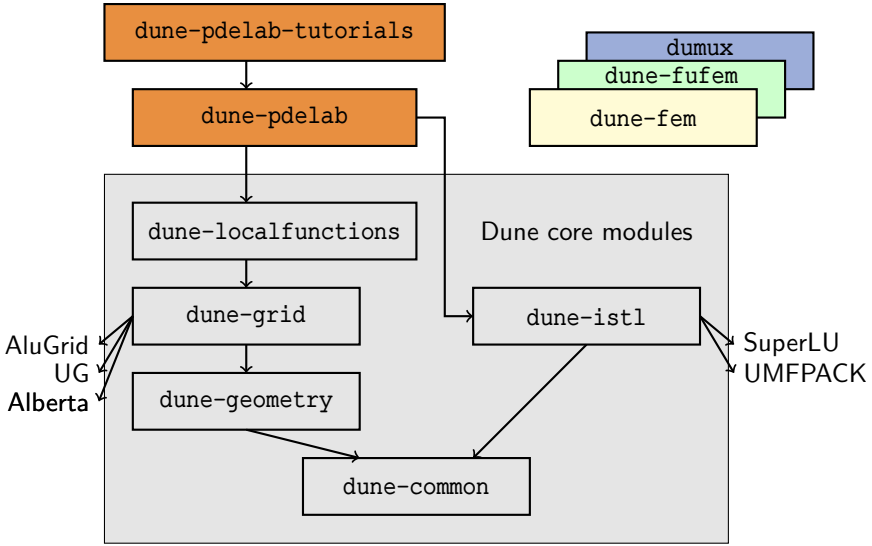
- Perform the *element-local computation*:

$$y_T = |\det B_T| w_d G^T G z_T$$

- Accumulate the results:

$$Az = \sum_{T \in \mathcal{T}_h} R_T^T y_T.$$

The Duniverse



Overview DUNE/PDELab Implementation

pdelab-tutorials/tutorial00/src

Files involved are:

1) File `tutorial00.cc`
   - Includes C++, DUNE and PDELab header files
   - Includes all the other files
   - Contains the `main` function
   - Creates a finite element mesh and calls the `driver`
2) File `tutorial00.ini`
   - Contains parameters controlling the execution
3) File `driver.hh`
   - Function `driver` setting up and solving the finite element problem
4) File `poissonp1.hh`
   - Class `PoissonP1` realizing the necessary element-local computations

Now lets go to the code ...

Methods in PoissonP1

FEM: $u_h \in V_{h,g}$: $a(u_h, v) = \ell(v) \quad \forall v \in V_{h,0}$

$\Leftrightarrow \underbrace{a(u_h, v) - \ell(v)}_{r(u_h, v)} = 0 \quad \forall v \in V_h$

lambda_volume

Evaluate $-\int_T f \phi_i \, dx$ for all testfunctions nonzero on T

$\rightsquigarrow$ the b_T

jacobian_volume

Evaluate $\int_T \nabla \phi_j \cdot \nabla \phi_i \, dx$ for all ansatz/test fcts on T

$\rightsquigarrow$ the A_T

alpha_volume

Evaluate $\int_T \nabla u_h \cdot \nabla \phi_i \, dx$ for all test fcts nonzero on T

jacobian_apply_volume $\rightsquigarrow$ matrix-free eval $A_T z_T$

in two variants. Coincide with alpha_volume in linear case.

GridFunctionSpace

represents all information about a finite element space, in part. local-to-global

consists of:

1) GridView — i.e. a finite element mesh
2) FiniteElementMap — information about local basis functions for each mesh element
3) Constraints — provides information where constraints are
4) VectorBackend — how to represent coefficient vectors

ConstraintsContainer — stores constraints information
is filled by the function constraints

Vector — coefficient vector in chosen backend

DiscreteGridFunction — a finite element function $u_h = \sum_j x_j \phi_j$

interpolate — fill coefficient vector from a function

Local Operator — element-local computations in the finite element method

GridOperator — a generic assembler

consists of:

1) two grid function spaces — ansatz and test space
2) local operator
3) MatrixBackend — knows how to create/store sparse matrices
4) two constraints containers — corresp. to the two grid function spaces

LinearSolverBackend — iterative linear solver

StationaryLinearProblemSolver — assembles and solves linear problem

VTKWriter — for writing Paraview files

Residual Formulation

Put your scheme in the form

$$u_h \in V_{h,g}: \quad \underbrace{r(u_h, v) = 0}_{\text{residual form}} \qquad \forall v \in V_h$$

In case of a linear PDE:

$$u_h \in V_{h,g}: \quad \underbrace{a(u_h, v) - \ell(v) = 0}_{= r(u_h, v)} \qquad \forall v \in V_h$$

Practice: Insert basis representation: $u_h(x) = \sum_{j=1}^{N} (z)_j \phi_j(x)$

$$a(u_h, \phi_i) = \sum_{T \in \mathcal{T}_h} \underbrace{\int_T \nabla u_h \cdot \nabla \phi_i \, dx}_{\text{alpha\_volume}} = \sum_{T \in \mathcal{T}_h} \int_T \nabla\left(\sum_{j=1}^{N} (z)_j \phi_j\right) \cdot \nabla \phi_i \, dx$$

$$= \sum_{j=1}^{N} (z_j) \sum_{T \in \mathcal{T}_h} \underbrace{\int_T \nabla \phi_j \cdot \nabla \phi_i \, dx}_{\text{jacobian\_volume}}$$

$$\ell(\phi_i) = \sum_{T \in \mathcal{T}_h} \underbrace{\int_T f \phi_i \, dx}_{\text{lambda\_volume}}$$

Generic Assembly Loop

GridOperator

1: **for** $T \in \mathcal{T}_h$ **do** ▷ loop over mesh elements

LocalOperator

2: $z_T = R_T z$ ▷ load element data

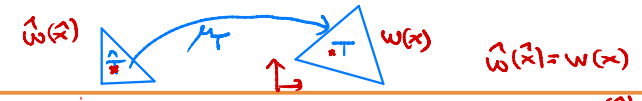
3: $q_T = \text{compute}(T, z_T)$ ▷ element local computations

4: Accumulate(q_T) ▷ store result in global data structure

5: **end for**

Only the computational kernels compute(T, z_T) need to be implemented by the user to implement the finite element method

Four Important Tools



1. Transformation formula for integrals. For $T \in \mathcal{T}_h$:

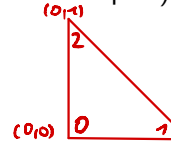
$$\int_T y(x)\,dx = \int_{\hat{T}} y(\mu_T(\hat{x}))|\det B_T|\,d\hat{x}, \quad \text{where } x = \mu_T(\hat{x}) = B_T\hat{x} + b_T$$

2. Midpoint rule on the reference element:

$$\int_{\hat{T}} q(\hat{x})\,dx \approx q(\hat{S}_d)w_d, \qquad (\hat{S}_d\text{: center of } d\text{-simplex})$$

(More accurate formulas are used later)

3. Basis functions via shape function transformation:



$$\hat{\phi}_0(\hat{x}) = 1 - \sum_{i=1}^{d}(\hat{x})_i, \quad \hat{\phi}_i(\hat{x}) = (\hat{x})_i, i > 0, \quad \phi_{T,i}(\mu_T(\hat{x})) = \hat{\phi}_i(\hat{x})$$

4. Computation of gradients. For any $w(\mu_T(\hat{x})) = \hat{w}(\hat{x})$:

$$B_T^T \nabla w(\mu_T(\hat{x})) = \hat{\nabla}\hat{w}(\hat{x}) \quad \Leftrightarrow \quad \nabla w(\mu_T(\hat{x})) = B_T^{-T}\hat{\nabla}\hat{w}(\hat{x}).$$