Friedrich-Alexander-Universität
Erlangen-Nürnberg

What’s new in dune-functions?

Carsten Gräser

FAU Erlangen–Nürnberg, Department Mathematik

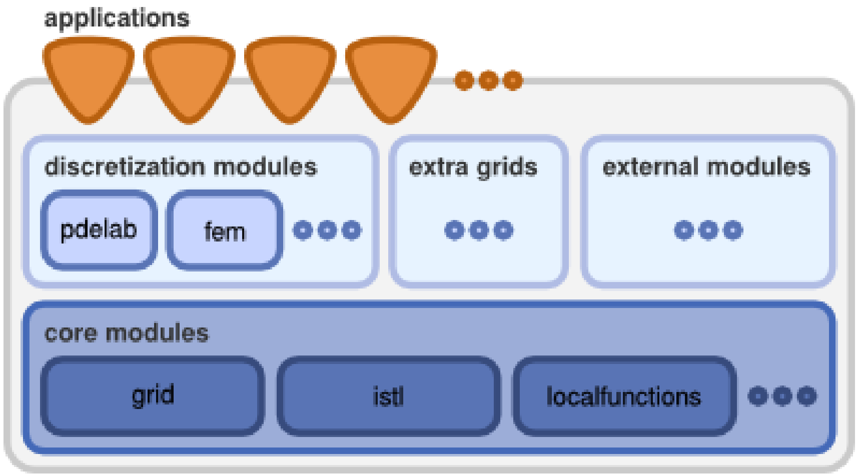
Dune user meeting 2023

2023-09-19

What is dune-function?

Core modules

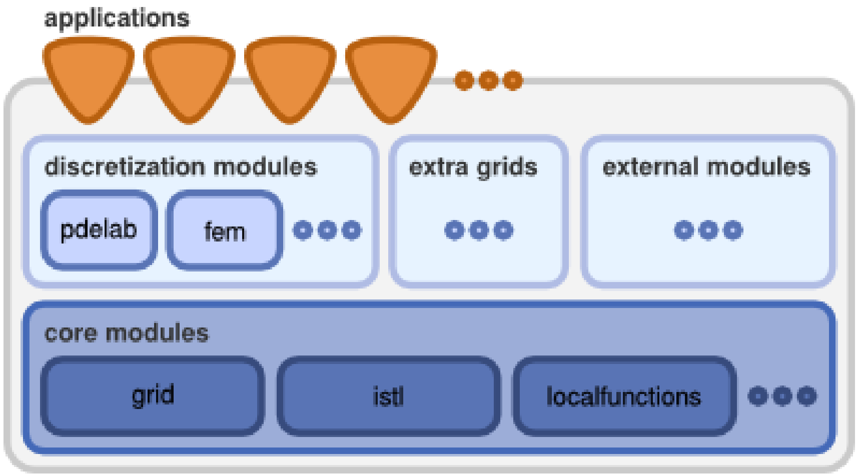
- dune-common
- dune-geometry
- dune-grid
- dune-localfunctions
- dune-istl



What is dune-function?

Core modules

- dune-common
- dune-geometry
- dune-grid
- dune-localfunctions
- dune-istl



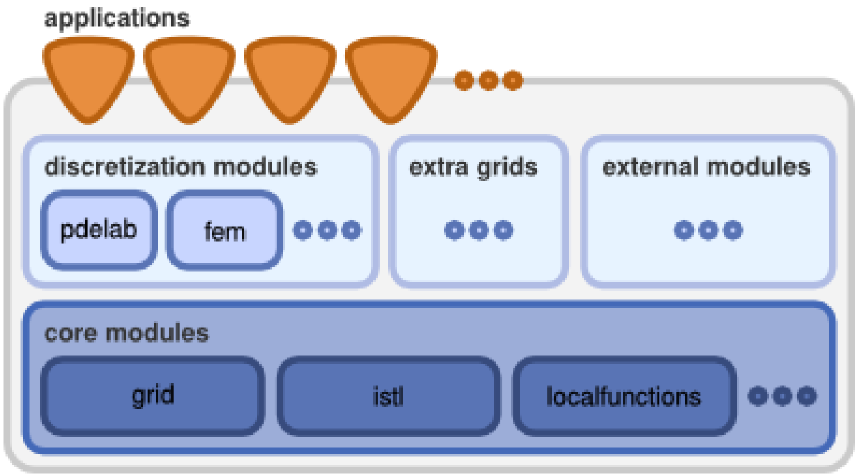
What's missing?

- Interfaces for (differentiable, grid, ...) functions
- Global function space bases
- Local and global assembler framework
- Definition of variational problems
- ...

What is dune-function?

Core modules

- dune-common
- dune-geometry
- dune-grid
- dune-localfunctions
- dune-istl



What's missing?

- Interfaces for (differentiable, grid, ...) functions
- Global function space bases
- Local and global assembler framework
- Definition of variational problems
- ...

This is what dune-functions deals with.

Detour: Some history

Dune-functions history

- Initiated by Engwer, Müthing, Sander, G.
- First commit ’13
- First release ’16

Detour: Some history

Dune-functions history

- Initiated by Engwer, Müthing, Sander, G.
- First commit '13
- First release '16

Dune core history

- First commit '03
- First release '07

Detour: Some history

Dune-functions history

- Initiated by Engwer, Müthing, Sander, G.
- First commit '13
- First release '16
- 10 years of dune-functions!

Dune core history

- First commit '03
- First release '07
- 20 years of dune!

Birthday Cake by Will Clayton, licensed under CC BY 2.0

Aims and design principles

Aims of dune-functions

- Unified interfaces of functions and function space bases
- Shared efforts between discretization modules
- Discretization modules using dune-functions
  - Dune-pdelab, dune-fufem, Amdis, ...

Aims and design principles

Aims of dune-functions

- Unified interfaces of functions and function space bases
- Shared efforts between discretization modules
- Discretization modules using dune-functions
  - Dune-pdelab, dune-fufem, Amdis, ...

Design principles and techniques

- Flexible and efficient interfaces
- Modern and lightweight design
- Duck typing, type deduction, concepts, ...

Function interface

Global function interface

```
auto y = f(x);
std::function<Range(Domain)> g = f;
```

Function interface

Global function interface

```
auto y = f(x);
std::function<Range(Domain)> g = f;
```

Differentiable function interface

```
auto df = derivative(f);
auto y = df(x);

DifferentiableFunction<Range(Domain)> g = f;

auto dg = derivative(g);
auto y = df(x);
```

Function interface

Global function interface

```
auto y = f(x);
std::function<Range(Domain)> g = f;
```

Differentiable function interface

```
auto df = derivative(f);
auto y = df(x);

DifferentiableFunction<Range(Domain)> g = f;

auto dg = derivative(g);
auto y = df(x);
```

Grid function interface

```
auto f_local = localFunction(f);
f_local.bind(element);
auto y = f_local(x_local);

GridViewFunction<Range(Domain), GridView> g = f;

auto g_local = localFunction(g);
f_local.bind(element);
auto z = g_local(x_local);
```

The interface for functions in the dune-functions module. (Engwer/G./Müthing/Sander ’17)

Interface for function space bases

Nested basis interface

Create a nested basis:

```
using namespace Dune::Functions::BasisFactory;
auto basis = makeBasis(gridView,
          composite(
            power<dim>(
              lagrange<2>()),
            lagrange<1>()));
```

Using the basis:

```
auto localView = basis.localView();
for(const auto& element: Dune::elements(gridView))
{
  // Bind to a grid element
  localView.bind(element);

  // Obtain finite element from ansatz tree
  auto&& velocityFE = localView.tree().child(_0, 0).finiteElement();
  auto&& pressureFE = localView.tree().child(_1).finiteElement();

  // Local element-wise index of basis function
  auto&& localIndex = localView.tree().child(_0, 0).localIndex(k);

  // Global index of basis function
  auto&& globalIndex = localView.index(localIndex);
}
```

Function space bases in the dune-functions module. (Engwer/G./Müthing/Sander ’18)

Interface for function space bases

Nested bases

- Bases can be nested using two constructions:
  - Composite product spaces $V_0 \times V_1 \times \cdots \times V_m$
  - Power spaces $V^k = V \times \cdots \times V$

Interface for function space bases

Nested bases

- Bases can be nested using two constructions:
  - Composite product spaces $V_0 \times V_1 \times \cdots \times V_m$
  - Power spaces $V^k = V \times \cdots \times V$
  - Like dune-pdelab

Interface for function space bases

Nested bases

- Bases can be nested using two constructions:
  - Composite product spaces $V_0 \times V_1 \times \cdots \times V_m$
  - Power spaces $V^k = V \times \cdots \times V$
  - Like dune-pdelab ... but less painful

Interface for function space bases

Nested bases

- Bases can be nested using two constructions:
  - Composite product spaces $V_0 \times V_1 \times \cdots \times V_m$
  - Power spaces $V^k = V \times \cdots \times V$
  - Like dune-pdelab . . . but less painful
- Order and blocking of global indices can be influenced
  - Interleaved or lexicographic order
  - Blocked or flat indices
  - Custom reordering/blocking

Interface for function space bases

Nested bases

- Bases can be nested using two constructions:
  - Composite product spaces $V_0 \times V_1 \times \cdots \times V_m$
  - Power spaces $V^k = V \times \cdots \times V$
  - Like dune-pdelab . . . but less painful
- Order and blocking of global indices can be influenced
  - Interleaved or lexicographic order
  - Blocked or flat indices
  - Custom reordering/blocking

Currently implemented bases

- Lagrange, Langrange-DG, hierarchical P2, Rannacher-Turek
- BDM, Raviart-Thomas, B-splines

Current development

Report from 2003 meeting

- Hosted in Münster
- Oliver Sander, Christian Engwer, Simon Praetorius, Santiago Ospina, Maik Porrmann, C.G.

Current development

Report from 2003 meeting

- Hosted in Münster
- Oliver Sander, Christian Engwer, Simon Praetorius, Santiago Ospina, Maik Porrmann, C.G.

Main decisions on the meeting

- Export information on index blocking structure
- Implementation of Hermite bases
- Dynamic power spaces

Technical detour: Indexing basis function

Example: Various multi-index schemes for Taylor-Hood

- Each column represents an indexing scheme
- Different indexing schemes for different containers and algorithms

	BL(BL)	BL(BI)	BL(FL)	BL(FI)	FL(BL)	FL(BI)	FL(FL)	FL(FI)
$v_{x_0,0}$	$(0,0,0)$	$(0,0,0)$	$(0,0)$	$(0,0+0)$	$(0,0)$	$(0,0)$	(0)	$(0+0)$
$v_{x_0,1}$	$(0,0,1)$	$(0,1,0)$	$(0,1)$	$(0,3+0)$	$(0,1)$	$(1,0)$	(1)	$(3+0)$
$v_{x_0,2}$	$(0,0,2)$	$(0,2,0)$	$(0,2)$	$(0,6+0)$	$(0,2)$	$(2,0)$	(2)	$(6+0)$
$v_{x_0,3}$	$(0,0,3)$	$(0,3,0)$	$(0,3)$	$(0,9+0)$	$(0,3)$	$(3,0)$	(3)	$(9+0)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$v_{x_1,0}$	$(0,1,0)$	$(0,0,1)$	$(0,n_2+0)$	$(0,0+1)$	$(1,0)$	$(0,1)$	(n_2+0)	$(0+1)$
$v_{x_1,1}$	$(0,1,1)$	$(0,1,1)$	$(0,n_2+1)$	$(0,3+1)$	$(1,1)$	$(1,1)$	(n_2+1)	$(3+1)$
$v_{x_1,2}$	$(0,1,2)$	$(0,2,1)$	$(0,n_2+2)$	$(0,6+1)$	$(1,2)$	$(2,1)$	(n_2+2)	$(6+1)$
$v_{x_1,3}$	$(0,1,3)$	$(0,3,1)$	$(0,n_2+3)$	$(0,9+1)$	$(1,3)$	$(3,1)$	(n_2+3)	$(9+1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$v_{x_2,0}$	$(0,2,0)$	$(0,0,2)$	$(0,2n_2+0)$	$(0,0+2)$	$(2,0)$	$(0,2)$	$(2n_2+0)$	$(0+2)$
$v_{x_2,1}$	$(0,2,1)$	$(0,1,2)$	$(0,2n_2+1)$	$(0,3+2)$	$(2,1)$	$(1,2)$	$(2n_2+1)$	$(3+2)$
$v_{x_2,2}$	$(0,2,2)$	$(0,2,2)$	$(0,2n_2+2)$	$(0,6+2)$	$(2,2)$	$(2,2)$	$(2n_2+2)$	$(6+2)$
$v_{x_2,3}$	$(0,2,3)$	$(0,3,2)$	$(0,2n_2+3)$	$(0,9+2)$	$(2,3)$	$(3,2)$	$(2n_2+3)$	$(9+2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
p_0	$(1,0)$	$(1,0)$	$(1,0)$	$(1,0)$	$(3+0)$	(n_2+0)	$(3n_2+0)$	$(3n_2+0)$
p_1	$(1,1)$	$(1,1)$	$(1,1)$	$(1,1)$	$(3+1)$	(n_2+1)	$(3n_2+1)$	$(3n_2+1)$
p_2	$(1,2)$	$(1,2)$	$(1,2)$	$(1,2)$	$(3+2)$	(n_2+2)	$(3n_2+2)$	$(3n_2+2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Export index tree information

Problem

- Currently the user has to select container types manually
- How to properly export the index information?

Export index tree information

FAU

Problem

- Currently the user has to select container types manually
- How to properly export the index information?

Solution

- The basis exports an object describing the index tree
- The descriptor mimics a suitable container

Export index tree information

Problem

- Currently the user has to select container types manually
- How to properly export the index information?

Solution

- The basis exports an object describing the index tree
- The descriptor mimics a suitable container
- Effect on users:
  - Users will hardly use this directly
  - Library developers can use this to automatically generate container types

Export index tree information

Problem

- Currently the user has to select container types manually
- How to properly export the index information?

Solution

- The basis exports an object describing the index tree
- The descriptor mimics a suitable container
- Effect on users:
  - Users will hardly use this directly
  - Library developers can use this to automatically generate container types

Implementation status

- Prototype exists
- Needs review and merge
- `https://gitlab.dune-project.org/staging/dune-functions/-/merge_requests/350`

Implementation of Hermite bases

Problem

- C^1/Hermite bases cannot use plain affine transformations
- How to generically implement Hermite-type bases?

Implementation of Hermite bases

Problem

- C^1/Hermite bases cannot use plain affine transformations
- How to generically implement Hermite-type bases?

Solution

- Intermediate interface for linearly transformed bases
- Sparse transformation of local reference bases

Implementation of Hermite bases

Problem

- C^1/Hermite bases cannot use plain affine transformations
- How to generically implement Hermite-type bases?

Solution

- Intermediate interface for linearly transformed bases
- Sparse transformation of local reference bases

Implementation status

- Prototype(s) exists
- Included implementations:
  - Hermite triangle, Morley, Agyris, and Arnold-Winther element
- Needs polishing and review
- https://gitlab.dune-project.org/staging/dune-functions/-/merge_requests/421

Dynamic power spaces

Problem

- Power spaces V^k with run-time exponent k
- How to efficiently generate dynamically sized return values?

```
auto x = f(y);
```

Dynamic power spaces

Problem

- Power spaces V^k with run-time exponent k
- How to efficiently generate dynamically sized return values?

```
auto x = f(y);
```

Partial Solution

- Incorporate dynamic power spaces
- Postpone solution of proper return values

Dynamic power spaces

Problem

- Power spaces V^k with run-time exponent k
- How to efficiently generate dynamically sized return values?

```
auto x = f(y);
```

Partial Solution

- Incorporate dynamic power spaces
- Postpone solution of proper return values

Implementation status

- Implementation exists
- Needs review and merge
- https://gitlab.dune-project.org/staging/dune-functions/-/merge_requests/285

Further discussed topics and future development

Multi-threading

- Interface is designed to localize mutable data
- For now no changes required
- Document thread safety guarantees

Caching of non-trivial bases

- How to implements evaluation caching of non-affine bases?
- Proper interface for cacheable information needed

Further discussed topics and future development

Multi-threading

- Interface is designed to localize mutable data
- For now no changes required
- Document thread safety guarantees

Caching of non-trivial bases

- How to implements evaluation caching of non-affine bases?
- Proper interface for cacheable information needed

Future development

- Data model for distributed bases
- ...

Ad: Dune-functions based assembly in dune-fufem

Painless definition of local assemblers

- UFL-like laguage to describe variational forms
- No code generation, plain C++
- Automatic caching and sharing of shape function evaluations
- Interacts nicely with dune-functions interfaces

Ad: Dune-functions based assembly in dune-fufem

Painless definition of local assemblers

- UFL-like laguage to describe variational forms
- No code generation, plain C++
- Automatic caching and sharing of shape function evaluations
- Interacts nicely with dune-functions interfaces

Example: Poisson-problem

```
auto u = trialFunction(basis);
auto v = testFunction(basis);
auto A = integrate(dot(grad(u), grad(v)));
```

Ad: Dune-functions based assembly in dune-fufem

Painless definition of local assemblers

- UFL-like laguage to describe variational forms
- No code generation, plain C++
- Automatic caching and sharing of shape function evaluations
- Interacts nicely with dune-functions interfaces

Example: Poisson-problem

```
auto u = trialFunction(basis);
auto v = testFunction(basis);
auto A = integrate(dot(grad(u), grad(v)));
```

Example: Mixed Poisson-problem

```
auto sigma = trialFunction(basis, _0, NonAffineFamiliy{});
auto u     = trialFunction(basis, _1);
auto tau   = testFunction(basis, _0, NonAffineFamiliy{});
auto v     = testFunction(basis, _1);

auto A = integrate(dot(sigma, tau) - div(tau)*u - div(sigma)*v);
```

Example: Navier–Stokes

```
// Explicitly denote subspaces of the mixes finite element space
auto velocityBasis = subspaceBasis(basis, _0);
auto pressureBasis = subspaceBasis(basis, _1);

// Define trial and test function spaces
auto u = trialFunction(velocityBasis);
auto p = trialFunction(pressureBasis);
auto v = testFunction(velocityBasis);
auto q = testFunction(pressureBasis);

auto f = Coefficient(rhsGridFunction);

// Initialize coefficient vector coeff
[...]

// Fixed point iteration
while ([...])
{
  // Previous iterate
  auto coeff_old = coeff;
  auto u_old = bindToCoefficients(u, coeff_old);

  // Define local assembler for Oseen problem with given advection
  auto A = integrate(nu*dot(grad(u),grad(v)) + dot(dot(u_old,grad(u)),v) + div(u)*q+div(v)*p);
  auto b = integrate(dot(f,v));

  // Call global assembler and linear solver
  [...]
}
```

Example: Primal plasticity with kinematic and isotropic hardening

```cpp
// Basis with deformation, plastic tensor, and hardening variable
auto basis = makeBasis(grid.leafGridView(), composite(
    power<dim>(lagrange<1>()),
    power<k>(lagrange<0>()),
    lagrange<0>()));

// B is an isometry between R^k and the trace free symmetric matrices
auto B = RangeOperator([](const auto& p) {
  return 1.0/std::sqrt(2.0) * FieldMatrix<double,2,2>{{p[0], p[1]},{p[1], -p[0]}};
});

// Symmetric gradient
auto E = [&](const auto& v) { return symmetrize(grad(v)); };

// Elasticity tensor for isotropic material
auto C = RangeOperator([&](const auto& e) {
  return 2*mu*e + lambda*Id*trace(e);
});

// Trial and test function spaces
auto u   = trialFunction(basis, _0);
auto p   = trialFunction(basis, _1);
auto eta = trialFunction(basis, _2);
auto v   = testFunction(basis, _0);
auto q   = testFunction(basis, _1);
auto nu  = testFunction(basis, _2);

// Apply isometry to get matrix valued trial and test function spaces
auto P = B(p);
auto Q = B(q);

auto A = integrate(dot(C(E(u)-P),E(v)-Q) + k1*dot(P,Q) + k2*dot(eta,nu));
```

Summary

Features to come soon

- Export information on index blocking structure
- Implementation of Hermite bases
- Dynamic power spaces

Summary

Features to come soon

- Export information on index blocking structure
- Implementation of Hermite bases
- Dynamic power spaces

Getting started

- https://gitlab.dune-project.org/staging/dune-functions
- Manuals on function and basis interfaces
- Examples with raw dune-functions

Summary

Features to come soon

- Export information on index blocking structure
- Implementation of Hermite bases
- Dynamic power spaces

Getting started

- `https://gitlab.dune-project.org/staging/dune-functions`
- Manuals on function and basis interfaces
- Examples with raw dune-functions

Dune-functions based assemblers in dune-fufem

- If you're interested: Contact me.

Thank you for your attention.