What’s new in dune-functions?

Carsten Gräser

FAU Erlangen–Nürnberg, Department Mathematik

Dune user meeting 2023
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.
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

Dune core history
- First commit ‘03
- First release ‘07

10 years of dune-functions!
20 years of dune!
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!
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

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

Global function interface

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

Differentiable function interface

```cpp
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

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

Differentiable function interface

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

DifferentiableFunction<Range(Domain)> g = f;

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

Grid function interface

```cpp
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)
Nested basis interface

Create a nested basis:

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

Using the basis:

```cpp
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)
Nested bases

- Bases can be nested using two constructions:
  - Composite product spaces $V_0 \times V_1 \times \cdots \times V_m$
  - Power spaces $V^k = V \times \cdots \times V$
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
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
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
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
**Example: Various multi-index schemes for Taylor-Hood**

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

<table>
<thead>
<tr>
<th></th>
<th>BL(BL)</th>
<th>BL(BI)</th>
<th>BL(FL)</th>
<th>BL(FI)</th>
<th>FL(BL)</th>
<th>FL(BI)</th>
<th>FL(FL)</th>
<th>FL(FI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_{x_0,0}$</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0)</td>
<td>(0, 0 + 0)</td>
<td>(0, 0)</td>
<td>(0, 0)</td>
<td>(0)</td>
<td>(0 + 0)</td>
</tr>
<tr>
<td>$v_{x_0,1}$</td>
<td>(0, 0, 1)</td>
<td>(0, 1, 0)</td>
<td>(0, 1)</td>
<td>(0, 3 + 0)</td>
<td>(0, 1)</td>
<td>(1, 0)</td>
<td>(1)</td>
<td>(3 + 0)</td>
</tr>
<tr>
<td>$v_{x_0,2}$</td>
<td>(0, 0, 2)</td>
<td>(0, 2, 0)</td>
<td>(0, 2)</td>
<td>(0, 6 + 0)</td>
<td>(0, 2)</td>
<td>(2, 0)</td>
<td>(2)</td>
<td>(6 + 0)</td>
</tr>
<tr>
<td>$v_{x_0,3}$</td>
<td>(0, 0, 3)</td>
<td>(0, 3, 0)</td>
<td>(0, 3)</td>
<td>(0, 9 + 0)</td>
<td>(0, 3)</td>
<td>(3, 0)</td>
<td>(3)</td>
<td>(9 + 0)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$v_{x_1,0}$</td>
<td>(0, 1, 0)</td>
<td>(0, 0, 1)</td>
<td>(0, $n_2 + 0$)</td>
<td>(0, 0 + 1)</td>
<td>(1, 0)</td>
<td>(0, 1)</td>
<td>($n_2 + 0$)</td>
<td>(0 + 1)</td>
</tr>
<tr>
<td>$v_{x_1,1}$</td>
<td>(0, 1, 1)</td>
<td>(0, 1, 1)</td>
<td>(0, $n_2 + 1$)</td>
<td>(0, 3 + 1)</td>
<td>(1, 1)</td>
<td>(1, 1)</td>
<td>($n_2 + 1$)</td>
<td>(3 + 1)</td>
</tr>
<tr>
<td>$v_{x_1,2}$</td>
<td>(0, 1, 2)</td>
<td>(0, 2, 1)</td>
<td>(0, $n_2 + 2$)</td>
<td>(0, 6 + 1)</td>
<td>(1, 2)</td>
<td>(2, 1)</td>
<td>($n_2 + 2$)</td>
<td>(6 + 1)</td>
</tr>
<tr>
<td>$v_{x_1,3}$</td>
<td>(0, 1, 3)</td>
<td>(0, 3, 1)</td>
<td>(0, $n_2 + 3$)</td>
<td>(0, 9 + 1)</td>
<td>(1, 3)</td>
<td>(3, 1)</td>
<td>($n_2 + 3$)</td>
<td>(9 + 1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$v_{x_2,0}$</td>
<td>(0, 2, 0)</td>
<td>(0, 0, 2)</td>
<td>(0, $2n_2 + 0$)</td>
<td>(0, 0 + 2)</td>
<td>(2, 0)</td>
<td>(0, 2)</td>
<td>($2n_2 + 0$)</td>
<td>(0 + 2)</td>
</tr>
<tr>
<td>$v_{x_2,1}$</td>
<td>(0, 2, 1)</td>
<td>(0, 1, 2)</td>
<td>(0, $2n_2 + 1$)</td>
<td>(0, 3 + 2)</td>
<td>(2, 1)</td>
<td>(1, 2)</td>
<td>($2n_2 + 1$)</td>
<td>(3 + 2)</td>
</tr>
<tr>
<td>$v_{x_2,2}$</td>
<td>(0, 2, 2)</td>
<td>(0, 2, 2)</td>
<td>(0, $2n_2 + 2$)</td>
<td>(0, 6 + 2)</td>
<td>(2, 2)</td>
<td>(2, 2)</td>
<td>($2n_2 + 2$)</td>
<td>(6 + 2)</td>
</tr>
<tr>
<td>$v_{x_2,3}$</td>
<td>(0, 2, 3)</td>
<td>(0, 3, 2)</td>
<td>(0, $2n_2 + 3$)</td>
<td>(0, 9 + 2)</td>
<td>(2, 3)</td>
<td>(3, 2)</td>
<td>($2n_2 + 3$)</td>
<td>(9 + 2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$p_0$</td>
<td>(1, 0)</td>
<td>(1, 0)</td>
<td>(1, 0)</td>
<td>(1, 0)</td>
<td>(3 + 0)</td>
<td>($n_2 + 0$)</td>
<td>($3n_2 + 0$)</td>
<td>($3n_2 + 0$)</td>
</tr>
<tr>
<td>$p_1$</td>
<td>(1, 1)</td>
<td>(1, 1)</td>
<td>(1, 1)</td>
<td>(1, 1)</td>
<td>(3 + 1)</td>
<td>($n_2 + 1$)</td>
<td>($3n_2 + 1$)</td>
<td>($3n_2 + 1$)</td>
</tr>
<tr>
<td>$p_2$</td>
<td>(1, 2)</td>
<td>(1, 2)</td>
<td>(1, 2)</td>
<td>(1, 2)</td>
<td>(3 + 2)</td>
<td>($n_2 + 2$)</td>
<td>($3n_2 + 2$)</td>
<td>($3n_2 + 2$)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Problem

- Currently the user has to select container types manually
- How to properly export the index information?
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
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](https://gitlab.dune-project.org/staging/dune-functions/-/merge_requests/350)
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?

```cpp
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?

```cpp
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
- ...

Painless definition of local assemblers

- UFL-like language 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

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

Example:

Mixed Poisson-problem

```cpp
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);```
Painless definition of local assemblers

- UFL-like language 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

```cpp
auto u = trialFunction(basis);
auto v = testFunction(basis);
auto A = integrate(dot(grad(u), grad(v)));`
Painless definition of local assemblers

- UFL-like language 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

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

**Example:** Mixed Poisson-problem

```cpp
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

```cpp
// 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

// 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 $\mathbb{R}^k$ 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

Getting started

• ![GitHub repository](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.
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
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.