Shape functions based on the generic reference elements

ANDREAS DEDNER

A.S.Dedner@warwick.ac.uk,
www2.warwick.ac.uk/fac/sci/maths/people/staff/andreas_dedner
Mathematics Institute,

THE UNIVERSITY OF WARWICK

Co-worker:
Martin Nolte

Stuttgart, 7th of October 2010

My road to DUNE

WARWICK

2003: Asked to join DUNE

It is not possible to implement an efficient generic grid interface
(A. Dedner, writting up his phd thesis)

2004: Asked to join DUNE again

Ok, I will give it a try
(A. Dedner, finished his phd thesis)

I missed some of the initial fun (introduction of *const Entities* in 2004) but added some fun of my own (*GenericReferenceElements* in 2008).

My road to DUNE

WARWICK

2003: Asked to join DUNE

It is not possible to implement an efficient generic grid interface
(A. Dedner, writting up his phd thesis)

2004: Asked to join DUNE again

Ok, I will give it a try
(A. Dedner, finished his phd thesis)

I missed some of the initial fun (introduction of *const Entities* in 2004) but added some fun of my own (*GenericReferenceElements* in 2008).

My road to DUNE

WARWICK

2003: Asked to join DUNE

It is not possible to implement an efficient generic grid interface
(A. Dedner, writting up his phd thesis)

2004: Asked to join DUNE again

Ok, I will give it a try
(A. Dedner, finished his phd thesis)

I missed some of the initial fun (introduction of *const Entities* in 2004) but added some fun of my own (*GenericReferenceElements* in 2008).

My road to DUNE

WARWICK

2003: Asked to join DUNE

It is not possible to implement an efficient generic grid interface
(A. Dedner, writting up his phd thesis)

2004: Asked to join DUNE again

Ok, I will give it a try
(A. Dedner, finished his phd thesis)

I missed some of the initial fun (introduction of *const Entities* in 2004) but added some fun of my own (*GenericReferenceElements* in 2008).

Given: domain Ω, ansatz space V, test space W,

quadratic form $a\colon V \times W \to \mathbb{R}$ (linear in second term), data $u_0 \in V$

Problem: find $u\colon (0,\infty) \times \Omega$ with $u(t,\cdot) \in V$, $u(0,x) = u_0(x)$, and

$$\int_\Omega \partial_t u(t,\cdot)\varphi = a(u(t,\cdot),\varphi) \qquad \forall \varphi \in V$$

Example: Non-linear heat equation ($\partial_t u - \nabla \cdot A(u)\nabla u = 0$)

$\Omega = [0,1]^2$, $V = H^1(\Omega)$ and

$$a(u,\varphi) = -\int_\Omega A(u)\nabla u \cdot \nabla \varphi$$

Discrete Formulation of Evolution Equations

WARWICK

$$u(0,x) = u_0(x), \qquad \int_\Omega \partial_t u(t,\cdot)\varphi = a(u(t,\cdot),\varphi) \quad \forall \varphi \in V$$

Construct: tesselation $\mathcal{G}$ of Ω, discrete function space $V_\mathcal{G} \subset V$, projection $u_{\mathcal{G},0} = \Pi_\mathcal{G}[u_0] \in V_\mathcal{G}$

Problem (semi-discrete): find $u_\mathcal{G} : (0,\infty) \to V_\mathcal{G}$ with $u_\mathcal{G}(0) \equiv u_{\mathcal{G},0}$, and

$$\sum_{E\in\mathcal{G}} \int_E \partial_t u_\mathcal{G}(t)\varphi_\mathcal{G} = a(u_\mathcal{G}(t),\varphi_\mathcal{G}) \quad \forall \varphi_\mathcal{G} \in V_\mathcal{G}$$

Using a basis $\mathcal{B}$ of $V_\mathcal{G}$ we can write $u(t) = \sum_{\psi\in\mathcal{B}} u_\psi(t)\psi$ where $u_\psi(t) : [0,\infty) \to \mathbb{R}$ are the degrees of freedom (DoF):

$$\sum_{\psi\in\mathcal{B}} \frac{d}{dt} u_\psi(t) \sum_{E\in\mathcal{G}} \int_E \psi\varphi = a(u_\mathcal{G}(t),\varphi) \quad \forall \varphi \in \mathcal{B}$$

Problem (fully discrete): The semi-discrete forumulation leads to a system of ordinary differential equations for the DoFs $(u_\psi)_{\psi\in\mathcal{B}}$. Use of an ODE solver to obtain fully discrete scheme (*method of lines*)

Requires

1. construct tessilation $\mathcal{G}$ of Ω
2. construct basis $\mathcal{B}$ of $V_{\mathcal{G}}$
3. implement of quadratic forms $a(\cdot,\cdot)$ with implementation of efficient operations $+, \circ$ etc.
4. implement ODE solver (e.g. explicit, implicit, semi-implicit)

Need to:

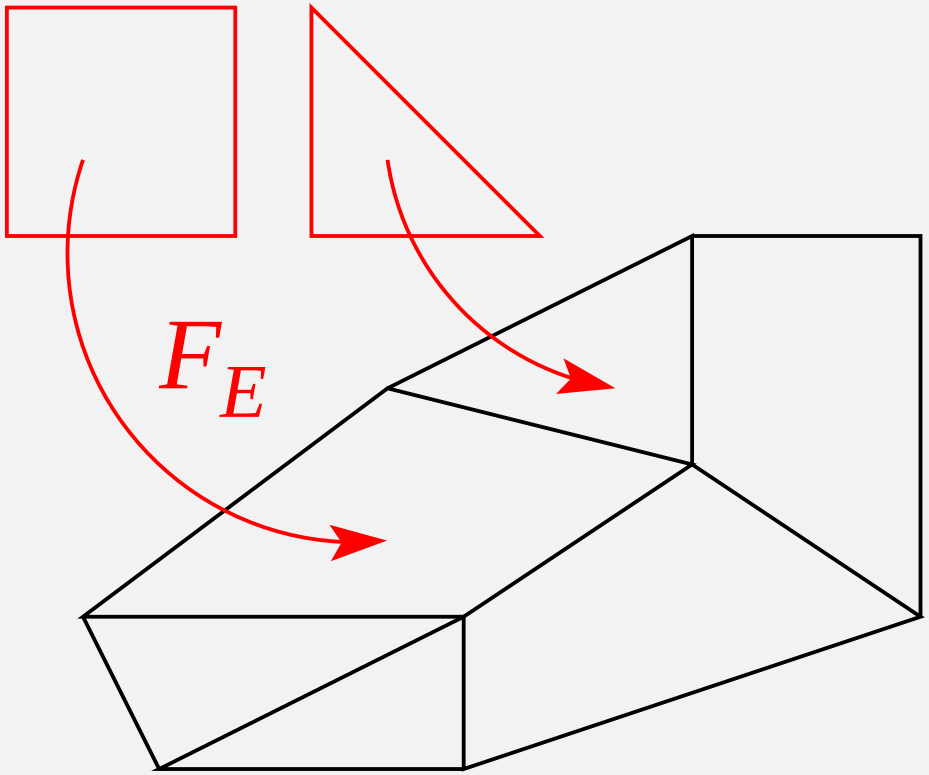
1. evaluate basis function and derivatives
2. efficient evaluation in given set of points (quadrature)
3. interpolation and projection into discrete function space
4. associate basis functions with subentities

Construction of Discrete Function Space

WARWICK

Given: grid $\mathcal{G}$ with entities E and a small set of reference elements $\mathcal{R}$

Assumption: for each entity E there is a reference element $\hat{E} \in \mathcal{R}$
and a bijective smooth mapping $F_E : \hat{E} \to E$



Localized Discrete Function Space

WARWICK

Shape functions: for each $\hat{R} \in \mathcal{R}$ there is a shape function set $\hat{\mathcal{B}}_{\hat{R}}$

Task 1: construct basis function set $\mathcal{B}$ of $V_{\mathcal{G}}$ with for all

$$\psi \in \mathcal{B} \text{ and } E \in \mathcal{G}\colon \psi_{|E} = \hat{\psi} \circ F_E^{-1} \text{ for some } \psi \in \hat{\mathcal{B}}_{\hat{R}}$$

Task 2: construct a mapper $\mu_E : \hat{\mathcal{B}}_{\hat{E}} \to \mathcal{B}$ so that

$$\mu_E(\hat{\psi})_{|E} = \hat{\psi} \circ F_E^{-1}$$

Then given $u_{\mathcal{G}} = \sum_{\psi \in \mathcal{B}} u_\psi \psi \in V_{\mathcal{G}}$ we have for $x \in E$

$$u_E(x) = \sum_{\hat{\psi} \in \hat{\mathcal{B}}_{\hat{E}}} u_{\mu_E(\hat{\psi})} (\hat{\psi} \circ F_E^{-1})(x)$$

or using local coordinates $\hat{x} \in \hat{E}$:

$$\hat{u}_E(\hat{x}) = \sum_{\hat{\psi} \in \hat{\mathcal{B}}_{\hat{E}}} u_{\mu_E(\hat{\psi})} \hat{\psi}(\hat{x})$$

Advantage:

simple construction of space $V_{\mathcal{G}}$ and speedup through caching of $\hat{\psi}$.

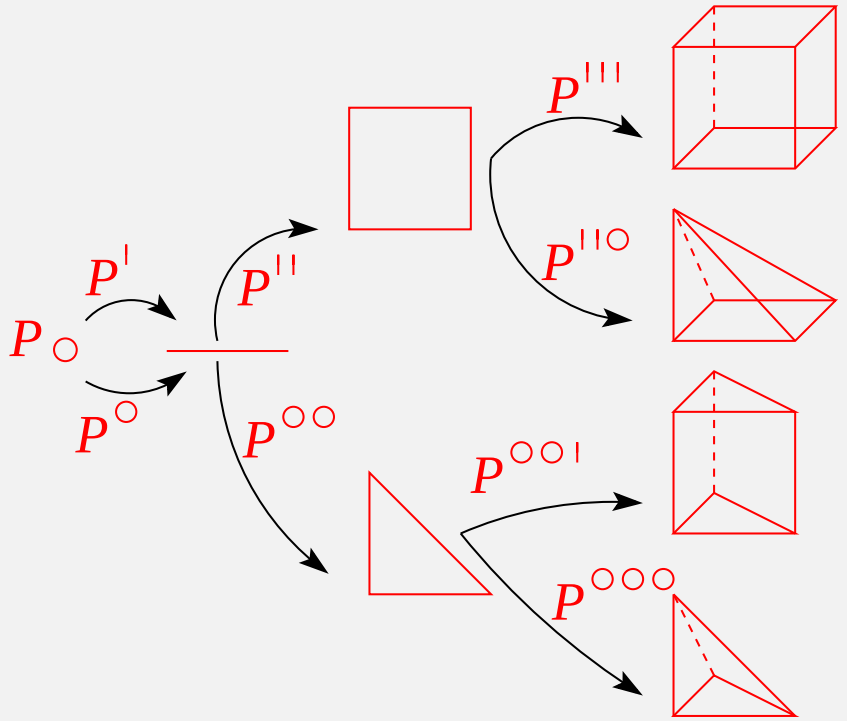
Generic Reference Elements

Given set of reference elements $\mathcal{R}^d$ with $R \subset \mathbb{R}^d, R \in \mathcal{R}^d$ we define

$$\mathcal{R}^{d+1} = \{R^{|}, R^{\circ} : R \in \mathcal{R}^{d+1}\}$$

where for $R \in \mathcal{R}^d$:

$$\begin{aligned} R^{|} &= \{(x,z) : z \in [0,1], x \in R\} \\ R^{\circ} &= \{(x(1-z),z) : z \in [0,1], x \in R\} \end{aligned}$$



For $d = 0$ we set
$\mathcal{R}^0 = \{P\}$ with $P = \{0\} \in \mathbb{R}^0$.
Note: R° is the Duffy transform of R.
Note: $R^{||} = R^{\circ|}, R^{\circ\circ} = R^{|\circ}, \ldots$.

Advantage:

Use of *template meta program* to construct numbering of subentities, mappings, and much more.

Static identification

```
struct Point;
template <class Base> struct Prism;
template <class Base> struct Pyramid;
```

Dynamic identification (Topologyid)

Identify R by pair of positive numbers $(\text{topologyId}(R), \text{dimension}(R))$:

$$\text{topologyId}(R^{\circ}) = \text{topologyId}(R),$$
$$\text{topologyId}(R^{|}) = 2^{dim-1} + \text{topologyId}().$$

IfTopology

Convert dynamic $\text{topologyId}(R)$ into static type

```
template< class Topology > struct Operation {
  static void apply ( ... ) {
  };
};

IfToplogy<Operation ,dimension >::apply(topologyId ,...);
```

Using a *dynamic* topologyId, calls Operation<Topology>::apply(...); with the correct *static* topology class.

Storage

```
template <class Traits> struct TopologyFactory {
  static Traits::Object*
      create(unsigned int topologyId, const Traits::Key &key) {
    Object *object;
    IfTopology< Factory, Traits::dimension >::
      apply( topologyId, key, object );
    return object;
  }
};
```

calls Factory<Topolgy> to generate an instance of *Object* for a given *dynamic* topologyId, the construction of which depends staticly on the Topology.

Example: quadrature point sets

Wrapper for singleton storage

```
template <class Factory>
struct TopologySingletonFactory;
```

Template Meta Programs

WARWICK

```
template <class Topology>
struct Factory;

template <>
struct Factory<Point> {
  Object *create(const Key &key) { ... };
};
template <class Base>
struct Factory< Pyramid<Base> > {
  Object *create(const Key &key) { ... };
};
template <class Base>
struct Factory< Prism<Base> > {
  Object *create(const Key &key) { ... };
};
```

Local Interfaces (static)

WARWICK

Basis

```
unsigned int size () const;
unsigned int order () const;
inline void evaluateFunction (
    const typename Traits::DomainType& in,
    std::vector<typename Traits::RangeType>& out ) const;
inline void evaluateJacobian (
    const typename Traits::DomainType& in,
    std::vector<typename Traits::JacobianType>& out ) const;
inline void evaluate (
    const typename Dune::template array<int,Traits::diffOrder>& direc
    const typename Traits::DomainType& in,
    std::vector<typename Traits::RangeType>& out ) const;
```

Interpolation

```
template<class F, class C>
void interpolate (const F& f, std::vector<C>& out ) const;
```

Local Interfaces

WARWICK

Key

```
std::size_t size () const ;
const LocalKey& localKey (std::size_t i) const ;

struct LocalKey {
unsigned int subEntity () const ;
unsigned int codim () const ;
unsigned int index () const ;
};
```

Note

All interface exist in as virtual interface for use with different reference elements...
First approache: static interface -> construct virtual interface using wrapper (e.g. with TopologyFactory)
Second approach: use virtual interface directly

Construction of General Shape Function Set

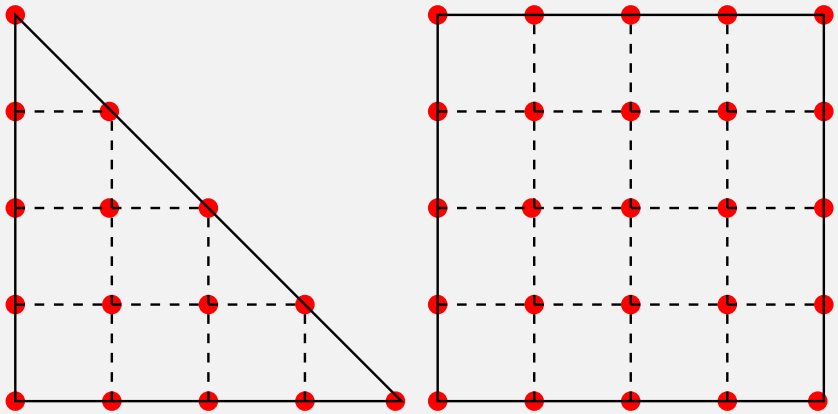
WARWICK

1. Generic implementation of monomial basis of order p:
   Given M_R^p for all $R \in \mathcal{R}^d$, define $M_{R^|}^p, M_{R^\circ}^p$
   Note: M_R^p is set needed for Lagrange interpolation on R of order p
2. Use M_R^p to define a set of (vectorial) polynomial functions $\tilde{\mathcal{B}} = \{\tilde{b}_j\}$
3. Construct matrix A and define final set: $\mathcal{B} = A\,\tilde{\mathcal{B}}$

Example:

Lagrange space is defined through Lagrange interpolation $\lambda_i^p(u) = u(\mathbf{x}_i^p)$ where $\mathbf{x}^p$ are the Lagrange points of order p.

Use $\tilde{B} = M_R^p$ and define $B = (\lambda_i^p(\tilde{b}_j))_{ij}$ then with $A = B^{-T}$, we have $\lambda_i^p(b_j) = \delta_{ij}$ for $\mathcal{B}_R^p = \{b_j\}$



template meta program over reference elements to construct lagrange points

Generic Monomials

WARWICK

Ansatz: we assume each base function is a polynomial in d variables.

Example on simplex topology $S^0 = P$, $S^{d+1} = (S^d)^\circ$

We construct Ψ_k^d of all monomials in d variables of exactly order k:

dim.	monomials	recursion relation
0	$\Psi_0^0 = 1$	$\Psi_0^0 = 1$
1	$\Psi_0^1 = 1$	$\Psi_0^0 = 1$
	$\Psi_1^1 = x$	$\Psi_1^0 = \emptyset \quad x\Psi_0^0 = x$
	$\Psi_2^1 = x^2$	$\Psi_2^0 = \emptyset \quad x\Psi_1^0 = \emptyset \quad x(x\Psi_0^0) = x^2$
2	$\Psi_0^2 = 1$	$\Psi_0^1 = 1$
	$\Psi_1^2 = \{x, y\}$	$\Psi_1^1 = x \quad = y$
	$\Psi_2^2 = \{x^2, xy, y^2\}$	$\Psi_2^1 = x^2 \quad y\Psi_1^1 = xy \quad y \quad = y^2$
d+1	$\Psi_0^{d+1} = \{\dots\}$	Ψ_0^d
	$\Psi_1^{d+1} = \{\dots\}$	$\Psi_1^d \quad z\Psi_0^d$
	$\Psi_2^{d+1} = \{\dots\}$	$\Psi_2^d \quad z\Psi_1^d \quad z(z\Psi_0^d)$

Slight modification required for general reference element R°.

Note: recursion also works for arbitrary derivatives

Generic Monomials

WARWICK

Ansatz: we assume each base function is a polynomial in d variables.

Example on simplex topology $S^0 = P$, $S^{d+1} = (S^d)^\circ$

We construct Ψ_k^d of all monomials in d variables of exactly order k:

dim.	monomials	recursion relation		
0	$\Psi_0^0 = 1$	$\Psi_0^0 = 1$		
1	$\Psi_0^1 = 1$	$\Psi_0^0 = 1$		
	$\Psi_1^1 = x$	$\Psi_1^0 = \emptyset$	$x\Psi_0^0 = x$	
	$\Psi_2^1 = x^2$	$\Psi_2^0 = \emptyset$	$x\Psi_1^0 = \emptyset$	$x(x\Psi_0^0) = x^2$
2	$\Psi_0^2 = 1$	$\Psi_0^1 = 1$		
	$\Psi_1^2 = \{x, y\}$	$\Psi_1^1 = x$	$y\Psi_0^1 = y$	
	$\Psi_2^2 = \{x^2, xy, y^2\}$	$\Psi_2^1 = x^2$	$y\Psi_1^1 = xy$	$y(y\Psi_0^1) = y^2$
d+1	$\Psi_0^{d+1} = \{\dots\}$	Ψ_0^d		
	$\Psi_1^{d+1} = \{\dots\}$	Ψ_1^d	$z\Psi_0^d$	
	$\Psi_2^{d+1} = \{\dots\}$	Ψ_2^d	$z\Psi_1^d$	$z(z\Psi_0^d)$

Slight modification required for general reference element R°.

Note: recursion also works for arbitrary derivatives

Generic Monomials

WARWICK

Ansatz: we assume each base function is a polynomial in d variables.

Example on simplex topology $S^0 = P$, $S^{d+1} = (S^d)^\circ$

We construct Ψ_k^d of all monomials in d variables of exactly order k:

dim.	monomials	recursion relation
0	$\Psi_0^0 = 1$	$\Psi_0^0 = 1$
1	$\Psi_0^1 = 1$	$\Psi_0^0 = 1$
	$\Psi_1^1 = x$	$\Psi_1^0 = \emptyset \quad x\Psi_0^0 = x$
	$\Psi_2^1 = x^2$	$\Psi_2^0 = \emptyset \quad x\Psi_1^0 = \emptyset \quad x(x\Psi_0^0) = x^2$
2	$\Psi_0^2 = 1$	$\Psi_0^1 = 1$
	$\Psi_1^2 = \{x, y\}$	$\Psi_1^1 = x \quad y\Psi_0^1 = y$
	$\Psi_2^2 = \{x^2, xy, y^2\}$	$\Psi_2^1 = x^2 \quad y\Psi_1^1 = xy \quad y(y\Psi_0^1) = y^2$
d+1	$\Psi_0^{d+1} = \{\dots\}$	Ψ_0^d
	$\Psi_1^{d+1} = \{\dots\}$	$\Psi_1^d \quad z\Psi_0^d$
	$\Psi_2^{d+1} = \{\dots\}$	$\Psi_2^d \quad z\Psi_1^d \quad z(z\Psi_0^d)$

Slight modification required for general reference element R°.

Note: recursion also works for arbitrary derivatives

Generic Monomials

WARWICK

Example on cube topology $Q^0 = P$, $Q^{d+1} = (Q^d)^|$

We construct Ψ_k^d of all bi-monomials in d variables of exactly order k:

dim.	bi-monomials	recursion relation
0	$\Psi_0^0 = 1$	$\Psi_0^0 = 1$
1	$\Psi_0^1 = 1$ $\Psi_1^1 = x$	$\Psi_0^0 = 1$ $\Psi_1^0 = \emptyset \quad x\Psi_1^0 = \emptyset \quad x\Psi_0^0 = x$
2	$\Psi_0^2 = 1$ $\Psi_1^2 = \{x, xy, y\}$	$\Psi_0^1 = 1$ $\Psi_1^1 = x \quad y\Psi_1^1 = yx \quad y\Psi_0^1 = y$
d+1	$\Psi_0^{d+1} = \{\ldots\}$ $\Psi_1^{d+1} = \{\ldots\}$ $\Psi_2^{d+1} = \{\ldots\}$	Ψ_0^d $\Psi_1^d \quad z\Psi_1^d \quad z\Psi_0^d$ $\Psi_2^d \quad z\Psi_2^d \quad z(z\Psi_2^d) \quad z(z\Psi_1^d) \quad z(z\Psi_0^d)$

Recursion correct for any reference element $R^|$.

Use *template meta program* over the definition of the generic reference elements.

Example Implementation

WARWICK

```
template <class Topology> class Monomials;

template <class BaseTopology>
class Monomials < Pyramid < BaseTopology > > {
  static const int d = BaseTopology :: dimension;
  static void evaluate(Domain x, Range m) {
    Monomials< BaseTology >::evaluate(x,m);
    // fill right column, multiplying by z = x[ d ]
  }
};
template <class BaseTopology>
class Monomials < Prism < BaseTopology > > {
  static const int d = BaseTopology :: dimension;
  static void evaluate(Domain x, Range m) {
    Monomials< BaseTology >::evaluate(x,m);
    // fill right column, multiplying by z = x[ d ]
  }
};
template <>
class Monomials < Point > {
  static void evaluate(Domain x, Range m) {
    m[0] = 1.;
  }
};
```

Simular for quadratures, lagrange point sets...

Construction of General Basis Function Set

WARWICK

Setting 1:

Given functionals $\Lambda = (\lambda_i)_i$ and polynomial function set $\tilde{\mathcal{B}} = (\tilde{b}_j)_j$:

1. define matrix $B = (\lambda_i(\tilde{b}_j))_{ij}$
2. construct $A = B^{-T}$ (using AlgLib with multiprecision arithmetic)
3. basis $\mathcal{B} = B^{-T}\tilde{\mathcal{B}}$ satisfies $\Lambda(\mathcal{B}) = I$

Example: Raviart-Thomas space (arbitrary dimension and order)

vector valued functions on simplex reference element R

$$\tilde{\mathcal{B}} = [\Psi^d]^d + x\Psi_p^d$$

$$\lambda_i(u) = \int_{\partial R} u \cdot n\varphi_i, \quad \lambda_{N+i}(u) = \int_R u\psi_i$$

where $\varphi_1, \ldots, \varphi_N$ is basis of $P_k(\partial R)$ and $\psi_1, \ldots, \psi_L$ is basis of $P_{k-1}(R)$

Setting 2 :

given a bilinear form a (e.g. $a(u,v) = \int_R uv$) construct orthonormal basis starting from $\tilde{B}$. Requires QR factorization of $(a(\tilde{b}_i, \tilde{b}_j))_{ij}$.

Usage of high precision field type

WARWICK

1. Construction phase

Evaluate *prebasis* $\tilde{\mathcal{B}}$ for $B = (\lambda_i(\tilde{b}_j))_{ij}$, compute B^{-T} (QR factorization).

2. Evaluation phase

Evaluation of *prebasis* $\tilde{\mathcal{B}}$ to compute basis functions $\mathcal{B} = B^{-T}\tilde{\mathcal{B}}$.

Note: For any basis the main step during evaluation is always the same matrix-vector multiplication (even for derivatives using derivatives of $\tilde{\mathcal{B}}$).

Setting up the matrix is only done once.

Usage of different field types:

We use *high precision floating point* arithmetics (alglib based on mpfr, gmp).

ComputeField: used to setup matrix and during inversion/QR.

StorageField: used for storing the matrix B^{-T}.

Evaluation is possible in any field (high for caching, double for on-the-fly...)

Converison between different field types is transparent for the user using the field_cast mechanism:

```
field_cast( in, out );   out = field_cast< OutField > ( in );
```

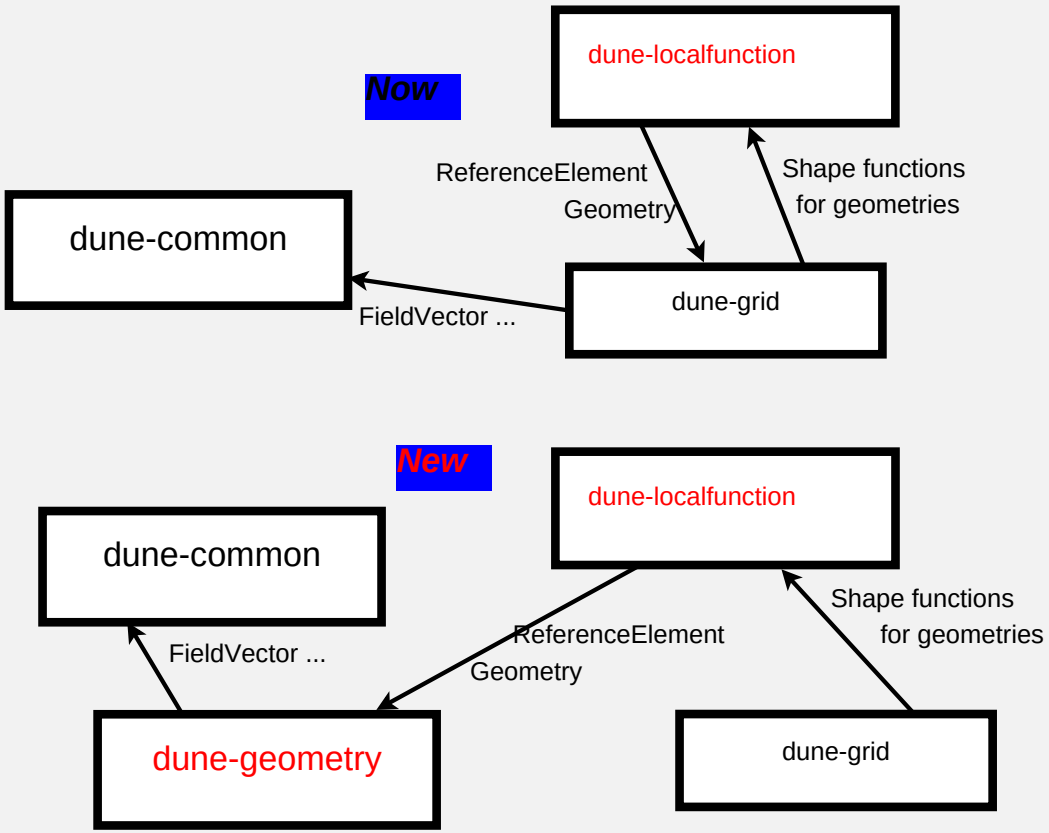
Further examples

WARWICK

- spectral Lagrange elements
- Raviart-Thomas using Lagrange points
  (still need implementation on general Prism type topologies)
- Nedelec edge elements
  (need to be implemented)

To Do (general)

dune module dependency



global interface

1. Take reference mapping into account
2. Piola transofrom, orientation of normals...
3. Global orientation of subentities (twists)

To Do (generic basis construction)

WARWICK

replace monomials with more stable basis

code generation

Many possibilities: write Maple code and Maple generates optimized code.

Example:

TEX output for RT on 2D Simplex order 3 (vector valued base functions)

$$\varphi_1(a,b) = (-0.24248711305964282109384248781082213137 25E1a + 0.41569219381653055044658712196140936806 64E1a^2 + 0.35333836474405096787959905366719796285 57E2ab - 0.58196907134314277062522197074597311529 20E2a^2 b - 0.94569974093260700226598570246220631235 19E2ab^2 + 0.11639381426862855412504439414919462305 84E3a^2 b^2 + 0.58196907134314277062522197074597311529 20E2ab^3, 0.17320508075688772935274463415058723669 45E1 - 0.34641016151377545870548926830117447338 91E1a - 0.28752043405643363072555609268997481291 26E2b + 0.54039985196148971558056325854983217848 68E2ab + 0.10704073990775661673999618390506291227 71E3b^2 - 0.16004149461936426192193604195514260670 56E3ab^2 - 0.13821765444399640802349021805216861488 24E3b^3 + 0.11639381426862855412504439414919462305 84E3ab^3 + 0.58196907134314277062522197074597311529 20E2b^4)$$

$$\varphi_2(a,b) = (0.49193495504995373321001820712088077179 73E1a - 0.21466252583997981085528067219820251860 26E2a^2 - 0.45615786740995709806747142842118035202 99E2ab + 0.18782971010998233449837058817342720377 71E2a^3 + 0.18782971010998233449837058817342720377 69E3a^2 b + 0.65740398538493817074429705860699521321 83E2ab^2 - 0.15026376808798586759869647053874176302 17E3a^3 b - 0.15026376808798586759869647053874176302 17E3a^2 b^2 - 0.25043961347997644599782745089790293836 91E2ab^3, ...)$$