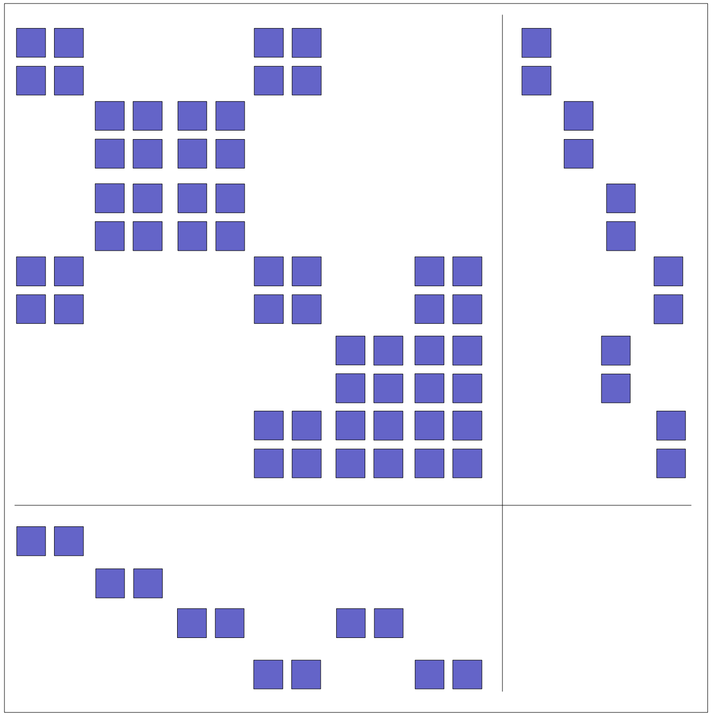
Workshop Kaskade 7 & Applications
A Tour of Kaskade 7

M. Weiser, L. Lubkoll



Zuse Institute Berlin

MATHEON

Ankara, 2013-10-02

Contents

Installation
- download
- installation scripts

Grids
- Dune grid interface
- creating grids
- modifying grids

Finite element spaces
- mathematical concepts
- reference elements
- Lagrangian, hierarchical, constant spaces
- continuous / discontinuous spaces

Variational problems
- problem definition
- Newton perspective
- assembly process and options
- accessing data

Solvers
- Dune ISTL interface
- direct solvers
- iterative solvers, adaptive PCG

IO

Error estimation
- mathematical concepts
- embedded and hierarchical error estimators
- coarsening

Time stepping
- implicit Euler
- extrapolation

Installation

Installation of Kaskade 7 code

MATHEON

Installation in 2 steps:

- Third-party libraries, e.g.,

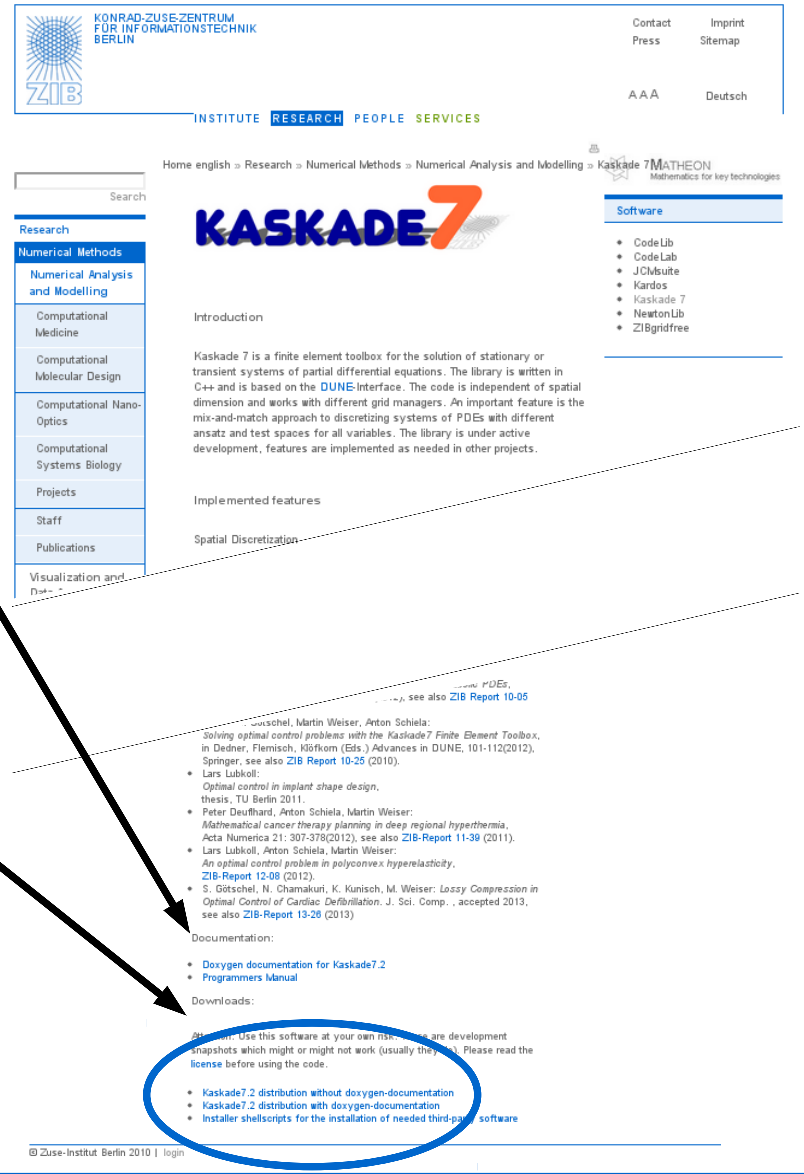
    suitable compiler
    boost
    Dune
    UG
    ACML (BLAS, LAPACK)
    UMFPACK
    MUMPS
    SuperLU
    ITSOL
    HYPRE
    TAUCS
    amiramesh

- Kaskade 7 library and tutorial examples

Installation of Kaskade 7 code

MATHEON

View doyxgen documentation online!



Installation

- Go to the Kaskade 7 website
  http://www.zib.de/en/numerik/software/kaskade-7.html
- download and unpack Kaskade7.2
- download and unpack installer shellscripts for the installation of needed third-party software
- run installer script

Installation of Kaskade 7 code

Next steps:

- change to Install_Kaskade7 directory

  use some install-xxx.Local file as a template for the install.Local file

- execute shellscript: ./install.sh

  prompts for some additional information, e.g., directory with Kaskade7 source files

  installs the required third-party software

  creates a suitable Makefile.Local in the Kaskade7 directory

  And finally run the commands

  make install
  make tutorial

  completing the installation of the Kaskade7 code

Installation of Kaskade 7 code

Some details:

- review the install.Local file, adapt it to your installation needs, e.g.,name and location of your LAPACK/BLAS library
- on Linux 64bit systems we recommend to use a ACML library instead of LAPACK/BLAS
- read the file README_BEFORE_STARTING before starting the installation, check for availability of the prerequites mentioned in this file
- take special attention to other notes marked by ***
- ...

MATHEON

Dune Grids

Dune Grid Interface

Motivation

- implementing grid management is hard
- several FE codes with unstructured grid manager available (UG, Alberta, ...)
- but different interfaces for accessing them (lock-in)

➡ create uniform interface to existing grid managers to be re-used (P. Bastian)

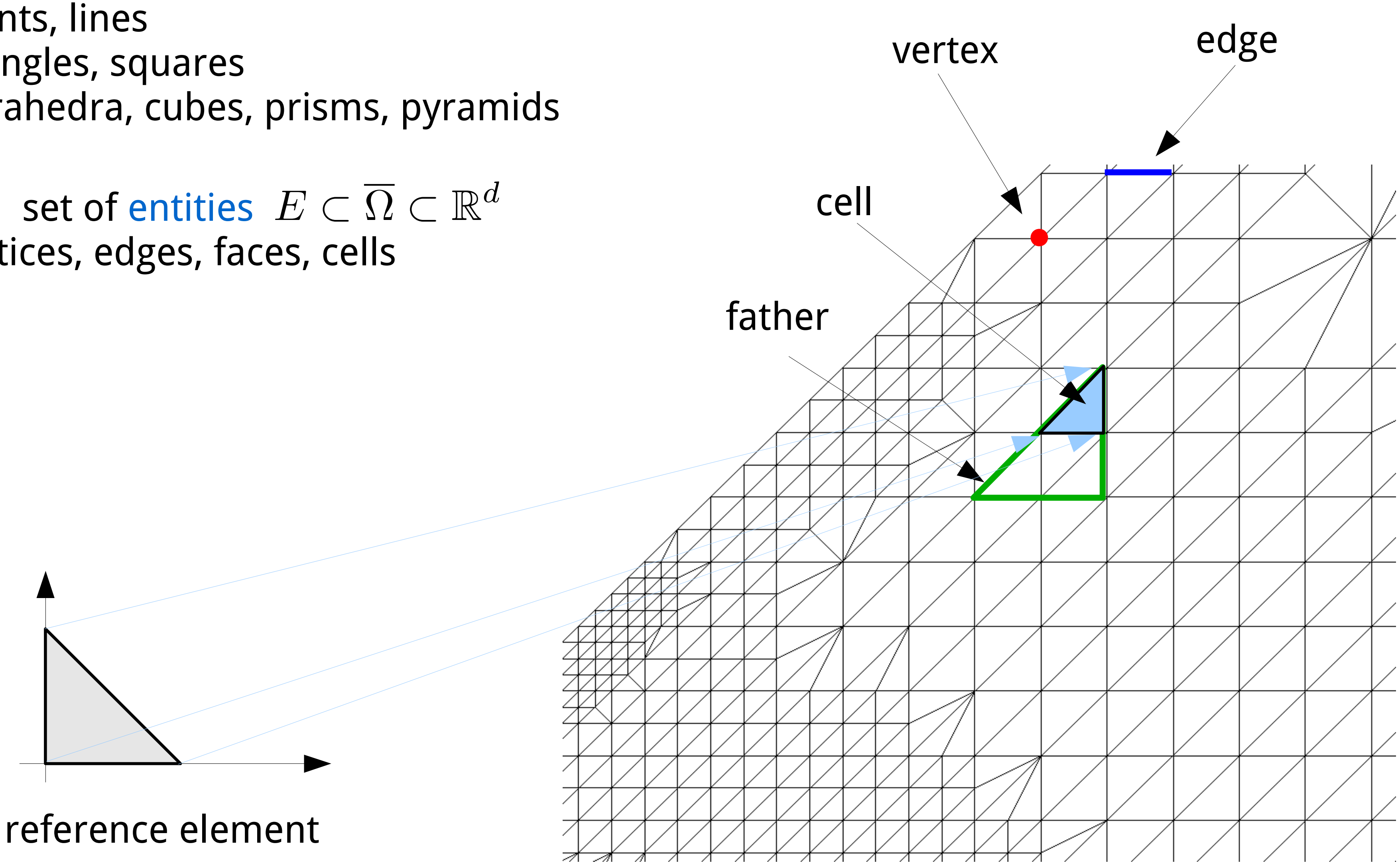
Current state

- comprehensive grid interface to several grid managers
- additional modules:
  - sparse linear algebra & iterative solvers (ISTL)
  - IO for different file formats
  - FE discretizations (dune-FEM, dune-PDElab, dune-fufem)
  - subgrid, multidomainsubgrids

Abstract Grid Concept

reference elements: convex polytopes $E_r \subset \mathbb{R}^k, \quad k = 0, \ldots, d$
- points, lines
- triangles, squares
- tetrahedra, cubes, prisms, pyramids

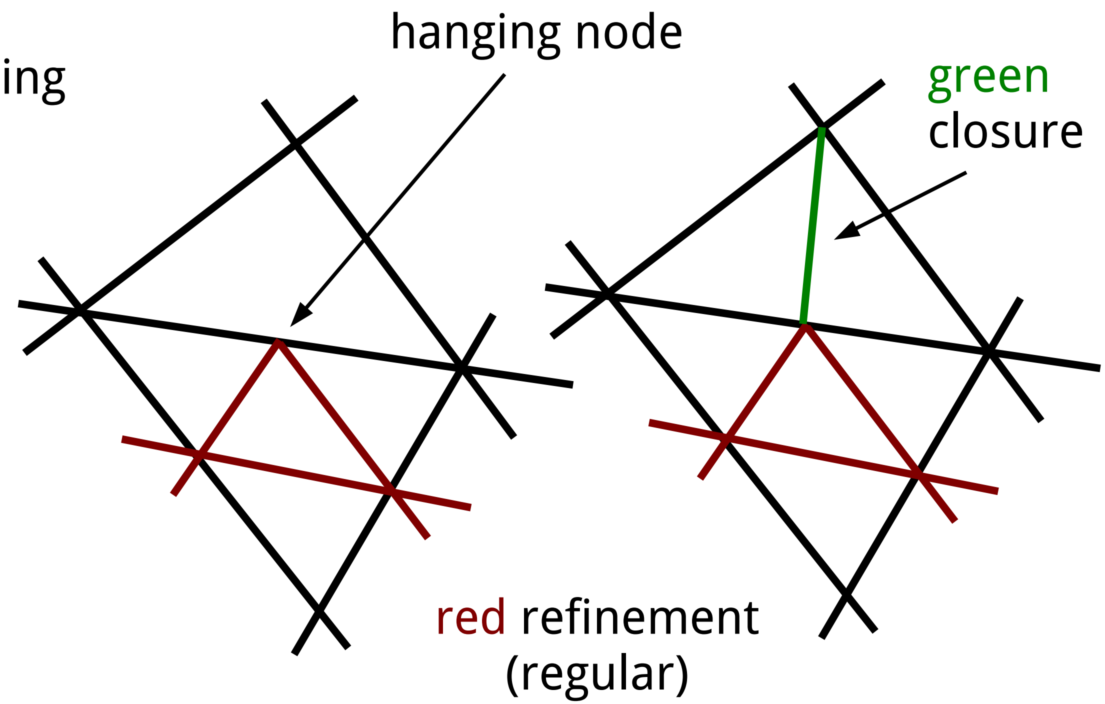
grid: set of entities $E \subset \overline{\Omega} \subset \mathbb{R}^d$
- vertices, edges, faces, cells

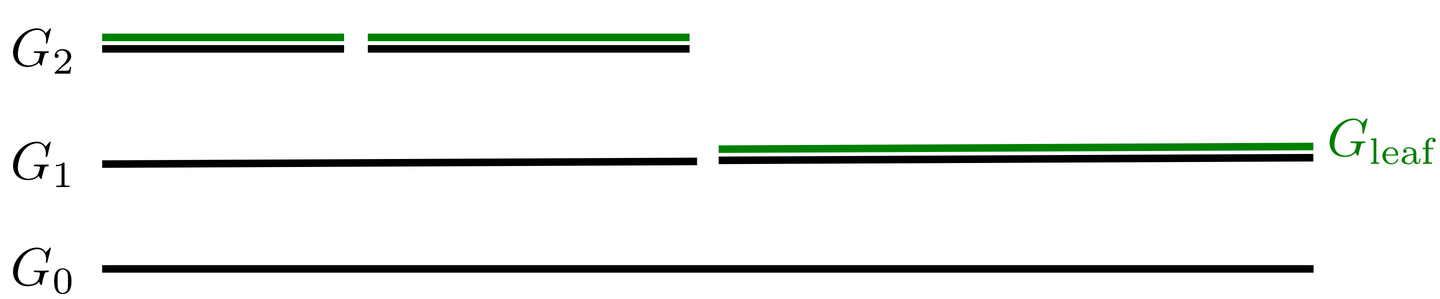


Abstract Grid Concept

grid hierarchy: grids $G_0, \ldots, G_j$ covering $\Omega_0 \supset \cdots \supset \Omega_j$ such that each cell in G_i is a union of cells from G_{i+1} or not contained in Ω_{j+1}

leaf grid: grid consisting of all non-refined cells of a grid hierarchy





Grid Views

View Concept

viewing data from different perspectives without involving recomputations leads to many copies of the same data

Name	peter	petra	piotr	petrus	biera
Salary	25000	40000	30000	35000	50000
Age	30	25	27	28	45

Data

Who earns most?

Name	biera	petra	petrus	piotr	peter
Salary	50000	40000	35000	30000	25000
Age	45	25	28	27	30

First Copy

Who is oldest?

Name	biera	peter	petrus	piotr	peter
Salary	50000	25000	35000	30000	40000
Age	45	30	28	27	25

Second Copy

Grid Views

MATHEON ZIB

View Concept

instead of copying and rearranging data, use a view object to change the perspective

Name	peter	petra	piotr	petrus	biera
Salary	25000	40000	30000	35000	50000
Age	30	25	27	28	45

Data

Who earns most?

Name			
Salary			
Age			

No copy, just remember the correct column

Who is oldest?

Name			
Salary			
Age			

No copy, just remember the correct column

Grid Views

View Concept

instead of copying and rearranging data, use a view object to change the perspective

Name	peter	petra	piotr	petrus	biera
Salary	25000	40000	30000	35000	50000
Age	30	25	27	28	45

Data

Who earns most?

Name					
Salary					
Age					

No copy, just remember the correct column

Who is oldest?

Name					
Salary					
Age					

No copy, just remember the correct column

Grid Views

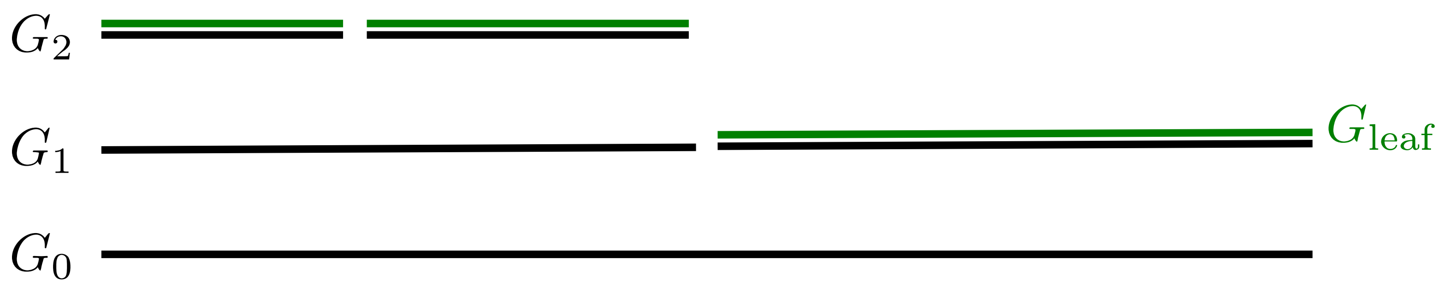
View Concept

instead of copying and rearranging data, use a view object to change the perspective

Here:

Leaf View

only show leaf entities of grid hierarchy (most refined cells)



Level View

only show entities of one level of the grid hierarchy

Grid Creation

Dune Grid factory

generic factory class for unstructured grids
provides unified interface for grid creation

More examples on grid creation:
io/amirameshreader.hh

```
// two-dimensional space: dim=2
typedef Dune::UGGrid<dim> Grid;
Dune::GridFactory<Grid> factory;

// insert vertices
Dune::FieldVector<double,dim> v;
v[0]=0; v[1]=0; factory.insertVertex(v);          // vertex id: 0
v[0]=1; v[1]=0; factory.insertVertex(v);          // vertex id: 1
v[0]=1; v[1]=1; factory.insertVertex(v);          // vertex id: 2
v[0]=0; v[1]=1; factory.insertVertex(v);          // vertex id: 3

// triangle defined by 3 vertex indices
std::vector<unsigned int> vid(dim+1);             // connectivity
Dune::GeometryType gt(Dune::GeometryType::simplex,dim);
vid[0]=0; vid[1]=1; vid[2]=2; factory.insertElement(gt,vid);
vid[0]=0; vid[1]=2; vid[2]=3; factory.insertElement(gt,vid);

// factory.createGrid() return type is Grid*
std::unique_ptr<Grid> grid( factory.createGrid() );

// access leaf grid
auto leafGrid = grid->leafView();
```

Grid Creation

Grid manager

- manages communication between grid and FE-functions, i.e. guarantees consistency of FE-function after grid refinement/coarsening
- 2 Types: - GridManager (default)
  - DeformingGridManager (with capabilities to move grid points)
- Constructor arguments:
  1. pointer to grid, this can be a
     - Dune::GridPtr<Grid>,
     - std::unique_ptr<Grid>&& or
     - Grid*&& (r-value reference of pointer to Grid)
  2. bool verbose=false

```
// a gridmanager is constructed
// as connector between geometric and algebraic information
GridManager<Grid> gridManager( factory.createGrid() );
```

Index and Id Set Concept

MATHEON

index set: mapping from grid entities of same dimension to 0,...,n
- contiguous numbering
- convenient for associating data with cells/vertices (store in arrays)
- mapping changes on grid refinement/coarsening

id set: mapping from grid entities to 0,...
- non-contiguous numbering
- mapping preserved on grid refinement/coarsening
- use this to associate data with cells/vertices during mesh modifications (hash maps)

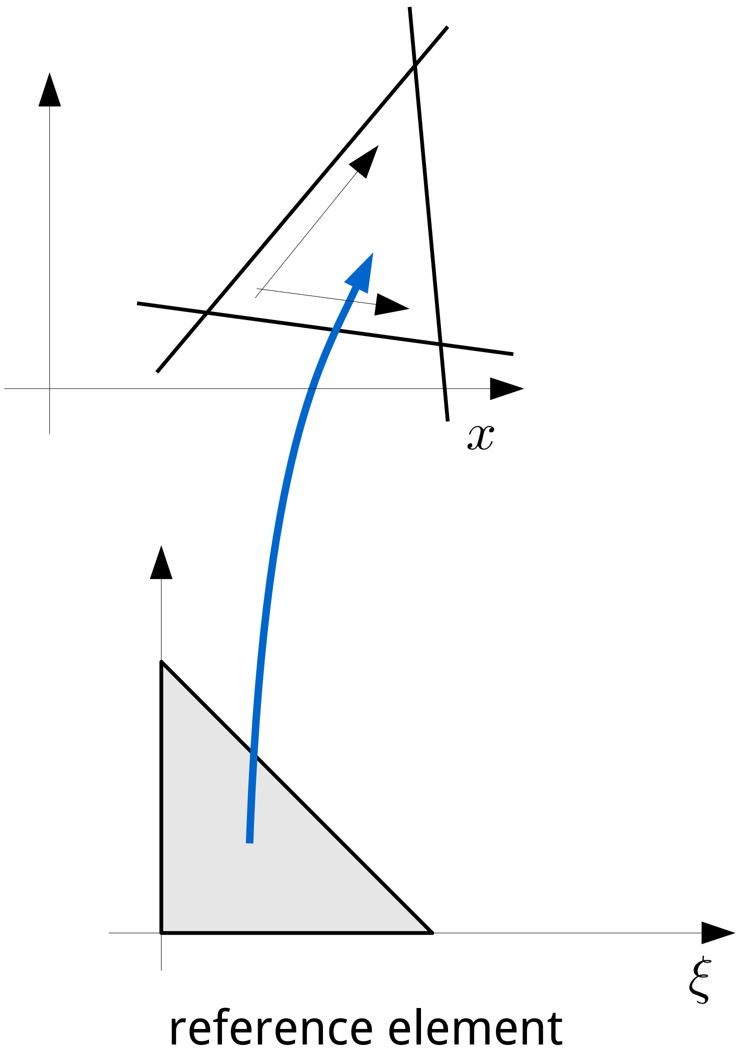
Accessing Grid Entities

```
GridView view;

// iterate over all cells (codimension 0 entities)
for (auto ci = view.template begin<0>();
          ci != view.template end<0>();
          ++ci)
{
   auto& cell = *ci; // obtain reference to cell
   std::cout << "cell number: " << view.indexSet().index(cell);
}

// iterate over all vertices (codimension dim entities)
int const dim = GridView::dimension;
for (auto vi = view.template begin<dim>();
          vi != view.template end<dim>();
          ++vi)
{
   auto& vertex = *vi; // obtain reference to vertex
   std::cout << "vertex number: " << view.indexSet().index(vertex);
}
```

see also fem/forEach.hh:

- forEachCell(…)
- forEachFace(…)
- …

Geometries

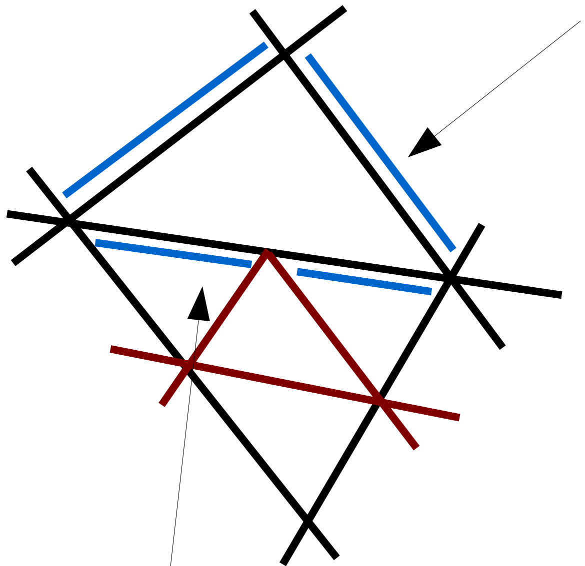



```
// obtaining a geometry for a cell
auto geo = cell.geometry();

// the Jacobian determinant
double dx = geo.integrationElement();

// global coordinate of local
auto x = geo.global(xi);

// local coordinate of global
auto xi = geo.local(x);

// type of reference element
auto gt = geo.type();
```

Intersections

MATHEON

intersections: segments of
cell boundary



Intersections depend
on the grid level!

Intersection are **not**
codim-1-geometries!

```
// step through all intersections
for (auto face=cell.ileafbegin();
          face!=cell.ileafend(); ++face {
  // obtain outer normal vector
  auto n = face->unitOuterNormal();

  // for integration over the boundary
  auto ndx = face->integrationOuterNormal();

  // access neighbor cell
  if (face->neighbor())
    auto co = face->outside();

  // access own cell
  auto ci = face->inside();
}
```

Example: Finding Cells in the Grid

```
template <class Grid>
typename Grid::template Codim<0>::EntityPointer
findCell(Grid const& grid, FieldVector<typename Grid::ctype,
                                       Grid::dimensionworld> global)
{
  typedef typename Grid::template Codim<0>::Entity  Cell;

  // Do linear search on coarse grid (level 0).
  auto inside = [&](Cell const& cell){
      return checkInside(cell.type(),cell.geometry().local(global)) < tol;
  };
  auto coarseIterator = std::find_if(grid.template lbegin<0>(0),
                                     grid.template lend<0>(0),
                                     inside);

  // Do a hierarchical search for a leaf cell containing the point.
  typename Grid::template Codim<0>::EntityPointer ci(coarseIterator);

  for(int level=0; level<=grid.maxLevel(); level++) {
    if(ci->isLeaf()) return ci;

    for (auto hi=ci->hbegin(level+1); hi!=ci->hend(level+1); ++hi)
      if(checkInside(hi->type(),hi->geometry().local(global)) < tol) {
        ci = typename Grid::template Codim<0>::EntityPointer(hi);
        break;
      }}}
```

Finite Element Spaces

MATHEON

Math concepts

FE spaces

scalar: $V_h = \{u \in C(\Omega) \mid \forall T : u|_T \in \mathbb{P}_k\}$ scalar

Nedelec: $V_h = \{u \in L^2(\Omega)^d \mid \forall F : [t^T u] = 0\}$ vectorial

bases $V_h = \mathrm{span}\{\varphi_i \in V_h \mid i = 1, \dots, n\}$

global basis functions defined in terms of local shape functions on reference elements

$$\varphi_i(x) = \psi_T(x) \left(\sum_j K_{T,ij} \phi_j(\xi(x)) \right)$$

global basis function

converter: linear mapping of shape function values

combiner: linear combination of shape functions

shape functions on reference element

Finite Element Spaces

MATHEON

$$\varphi_i(x) = \psi_T(x) \left(\sum_j K_{T,ij} \phi_j(\xi(x)) \right)$$

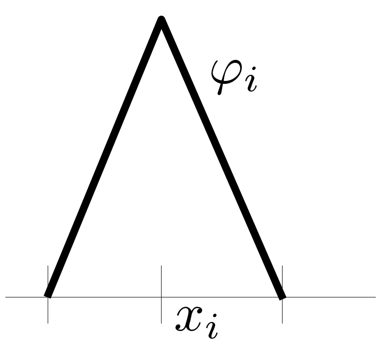
global basis function

converter: linear mapping of shape function values

combiner: linear combination of shape functions

shape functions on reference element

Example 1D linear FE



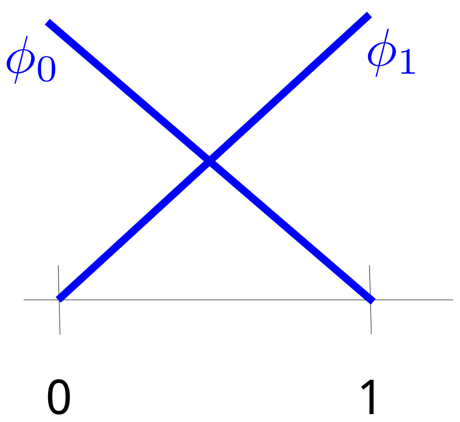
nodal hat function

$$\phi_T \equiv 1$$

trivial converter

$$K_T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

trivial combiner



Finite Element Spaces

$$\varphi_i(x) = \psi_T(x) \left(\sum_j K_{T,ij} \phi_j(\xi(x)) \right)$$

Example 2D hierarchic FE $V_h = \{u \in C(\Omega) \mid u|_T \in \mathbb{P}_3\}$

shape functions

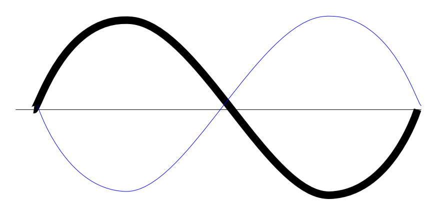
$$\mathrm{span}\{\phi_0, \phi_1, \phi_2\} = \mathbb{P}_1 \quad \text{linear FE}$$

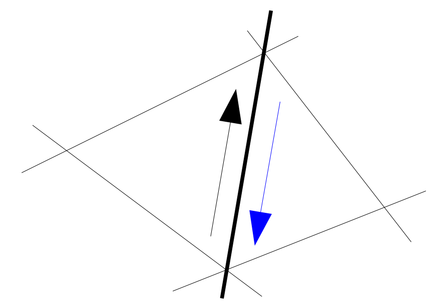
$$\mathbb{P}_1 \oplus \mathrm{span}\{\phi_3, \phi_4, \phi_5\} = \mathbb{P}_2 \quad \text{quadratic bubbles}$$

$$\mathbb{P}_2 \oplus \mathrm{span}\{\phi_6, \ldots, \phi_9\} = \mathbb{P}_3 \quad \text{cubic bubbles}$$

on triangle edge $[0, 1]$

$$\phi_6 = x(x - \frac{1}{2})(x - 1)$$





globally continuous ansatz functions $\Rightarrow$ $\phi|_{T_1} = \phi_6 \qquad \phi|_{T_2} = -\phi_6$

$$K_{T,ij} \in \{-1, 1\}$$

Kaskade Concepts

Shape functions

```
// Abstract base class for shape functions
// G: grid type
// T: scalar field type (usually double)
// comp: number of vectorial components (scalar=1)
template <class G, class T, int comp=1>
class ShapeFunction
{
  typedef FieldVector<typename G::ctype,G::dimension> Position;

  virtual FieldVector<T,comp>
  evaluateFunction(Position const& x) const = 0;

  virtual FieldMatrix<T,comp,G::dimension>
  evaluateDerivative(Position const& x) const = 0;
};
```

Kaskade Concepts

Shape function sets

```
// Base class for shape function sets
// G: grid type
// T: scalar field type (usually double)
// comp: number of vectorial components (scalar=1)
template <class G, class T, int comp=1>
class ShapeFunctionSet
{
  virtual ShapeFunction<G,T,comp> const& operator[](int i) const;
  int order() const; // maximal polynomial order
  Dune::GeometryType type() const;
};
```

available shape function sets:

scalar:
- Lagrange shape functions
  arbitrary order on simplices, up to 2 on hexahedra
- hierarchic shape functions
  arbitrary order on simplices

vectorial:
- Nedelec
  first order on simplices

Kaskade Concepts

MATHEON ZIB

Finite Element spaces

```
// Class for all finite element spaces
// LocalToGlobalMapper: defines shape & ansatz functions
template <class LocalToGlobalMapper>
class FEFunctionSpace {
  // Constructor
  template <typename... Args>
  FEFunctionSpace(GridManagerBase<Grid>& gridMan,
                  GridView const& gridView_,
                  Args... args);

  // type of m-component FE functions from this space
  template <int m>
  struct Element {
      typedef FunctionSpaceElement<FEFunctionSpace,m> type;
  };

  // helper class for efficient evaluation of FE functions
  struct Evaluator;
  …
};
```

Kaskade Concepts

MATHEON ZIB

Finite Element spaces

Local to global mapper: - manages (global) degrees of freedom
- converts between global ansatz functions and local shape functions by defining shape function set, combiner, converter

available mappers:

scalar:
- (dis-)continuous Lagrange
- (dis-)continuous hierarchic
- constant
- boundary spaces

vectorial:
- Nedelec

```
// create continuous Lagrange P2 space
typedef FEFunctionSpace<ContinuousLagrangeMapper<double,
                                                LeafView>> H1;
H1 h1(gridManager,gridManager.grid().leafView(),2);

// create discontinuous hierarchic P0 space
typedef FEFunctionSpace<DiscontinuousHierarchicMapper<double,
                                                     LeafView>> L2;
L2 l2(gridManager,gridManager.grid().leafView(),0);
```

Finite Element Functions

MATHEON ZIB

```
// create a scalar FE function
typename L2::Element<1>::type heatCapacity(l2);

// create a multi-component FE function
typename H1::Element<3>::type displacement(h1);

// create a vectorial FE function
typename Nedelec::Element<1>::type efield(nedelec);
```

$\in L^2(\Omega)$

$\in H^1(\Omega)^3$

$\in H_{\text{rot}}(\Omega)$

FE functions register at their space for
- access to basis functions for evaluation
- being informed about prolongation on mesh refinement

use scalar spaces for - scalar functions
- multi-component (tensor product valued) functions

use vectorial spaces for - vectorial functions (with special connection between vectorial components)

Finite Element Functions

```
// assign constant value to all FE coefficients
heatCapacity = 1.0;

// evaluate at point x in domain (SLOW!)
FieldVector<double,2> x; x[0]=8.5; x[1]=3.8;
FieldVector<double,1> c = heatCapacity.value(x);

// evaluate derivative at local coordinate in cell
FieldVector<double,2> xi; xi[0]=0.5; xi[1]=0.2;
FieldMatrix<double,1,2> dc = heatCapacity.gradient(cell,xi);

// efficient evaluation of multiple functions
typename H1::Evaluator eval(h1);
eval.moveTo(cell);
eval.evaluateAt(xi);
auto c = heatCapacity.value(eval);
auto dp = pressure.gradient(eval);

// access FE coefficients
heatCapacity.coefficients()[0] = 0.0;
```

Variational Problems

Problem Definition

MATHEON

Variational functionals

$$\min_{u \in U} \int_\Omega f(x, u_1, \nabla u_1, \dots, u_n, \nabla u_n)\, dx + \int_{\partial\Omega} g(x, u_1, \dots, u_n)\, ds$$

Kaskade problem definition: a class implementing f, g
and providing meta information

Weak formulations

$$\int_\Omega (F_i(x, u, \nabla u) v_i + \hat{F}_i(x, u, \nabla u) \nabla v_i)\, dx + \int_{\partial\Omega} G(x, u) v_i\, ds = 0, \quad i = 1, \dots, m$$

Newton's Perspective

Variational functional $\min_u J(u)$

linearization point (origin)

Newton's method: $J''(u_k)\delta u_k = -J'(u_k)$

$$u_{k+1} = u_k + \delta u_k$$

quadratic functionals: one Newton step gives exact solution

Linear equations $Au = b \quad \Leftrightarrow \quad J''(0)\delta u = -J'(0), \quad u = 0 + \delta u$

for consistency: evaluate $-b$

solve $Au = -(-b)$

Problem Definition

Poisson problem

$$J(u) = \int_\Omega \left(\frac{1}{2} |\nabla u|^2 - f u \right) dx + \int_{\partial\Omega} 10^9 u^2 \, ds$$

```
// Poisson problem definition
template <class Variables>
struct Poisson {

  // define types of ansatz and test variables
  typedef Variables AnsatzVars;
  typedef Variables TestVars;
  typedef Variables OriginVars;

  // define kind of problem
  static ProblemType const type = VariationalFunctional;
  static constexpr int dim = AnsatzVars::Grid::dimension;

...
```

Problem Definition

$$J(u) = \int_\Omega \left(\frac{1}{2}|\nabla u|^2 - fu\right) dx + \int_{\partial\Omega} 10^9 u^2 \, ds$$

```
// implement f
struct DomainCache : public CacheBase<Poisson,DomainCache> {
  // for computed gradient of uh
  Dune::FieldMatrix<double,1,dim> du;
  Dune::FieldVector<double,1> f;
  LinAlg::EuclideanScalarProduct sp;

  DomainCache(){}

  // specify evaluation point x & gradient of uh
  void moveTo(Cell& cell) { } // nothing to do
  void evaluateAt(Position& xi, Evaluators const& evals) {
    du = at_c<0>(vars.data).gradient(at_c<0>(evals));
  }

  // evaluate f
  double d0() const { return 0.5 * sp(du,du); }

...
```

Problem Definition

$$J(u) = \int_\Omega \left(\frac{1}{2} |\nabla u|^2 - f u \right) dx + \int_{\partial\Omega} 10^9 u^2 \, ds$$

```
    // evaluate f' (directional derivative)
    template <int row>
    double d1_impl (VariationalArg<double,dim,1> const& arg) const
    {
      return sp(du,arg.gradient);
    };

    // evaluate f'' (directional second derivative)
    template <int row, int col>
    double d2_impl(VariationalArg<double,dim,1> const& test,
                   VariationalArg<double,dim,1> const& ansatz) const
    {
      return sp(test.gradient,ansatz.gradient);
    }
  }; // end of DomainCache

...
```

Problem Definition

$$J(u) = \int_\Omega \left(\frac{1}{2}|\nabla u|^2 - fu\right) dx + \int_{\partial\Omega} 10^9 u^2 \, ds$$

```
// implement g
struct BoundaryCache {
  // for computed value of uh
  Dune::FieldVector<double,1> u;

  BoundaryCache();

  // specify evaluation point x & compute value of uh
  void moveTo(Intersection& bdry) { } // nothing to do
  void evaluateAt(Position& xi, Evaluators const& evals) {
    u = at_c<0>(vars.data).value(at_c<0>(evals));
  }

  // evaluate g
  double d0() const { return 0.5e9 * u*u; }

...
```

Problem Definition

$$J(u) = \int_\Omega \left(\frac{1}{2} |\nabla u|^2 - f u \right) dx + \int_{\partial\Omega} 10^9 u^2 \, ds$$

```
    // evaluate g' (directional derivative)
    template <int row>
    double d1_impl(VariationalArg<double,dim,1> const& arg) const
    {
      return 1e9 * u * arg.value;
    };

    // evaluate g'' (directional second derivative)
    template <int row, int col>
    double d2_impl(VariationalArg<double,dim,1> const& test,
                   VariationalArg<double,dim,1> const& ansatz) const
    {
      return 1e9 * test.value * ansatz.value;
    }
  }; // end of BoundaryCache

...
```

Problem Definition

```
  // provide static meta information
  template <int row>
  struct D1
  {
    static bool const present   = true;  // structurally nonzero
  };

  template <int row, int col>
  struct D2
  {
    static bool const present   = true;  // structurally nonzero
    static bool const symmetric = true;
    static bool const lumped    = false; // consider only diagonal
  };
}; // end of variational functional
```

Problem Definition

MATHEON

```
// evaluate f' (directional derivative)
    template <int row>
    FieldVector<double, TestVars::template Components<row>::m>
    d1(VariationalArg<double,dim> const& arg) const {
      return du * arg.gradient;
    };
```

Remember: scalar ansatz functions are used for multi-component variables!

`arg` is scalar, but **`d1`** returns an m-vector

$$r_i = f'(x,u)(v\, e_i)$$

return value

arg

Problem Definition Utilities

```
#include „fem/functional_aux.hh“

// convenience base class for variational functionals
template <ProblemType Type>
class FunctionalBase;
```

Defines **`type`** member and default **`D1/D2`** meta information.

```
// interface wrapper for domain/boundary caches
template <class Functional, class Cache>
class CacheBase;
```

Defines domain/boundary caches in terms of provided class **`Cache`** with simpler interface using vectorial test functions: $r = f'(x,u)v$

```
template <int row>
double d1_impl(VariationalArg<double,dim,m> const& arg);
```

Assembly Process

MATHEON

Assemble linearization $\rightarrow F_h'(x), F_{hh}''(x)$

```
#include „fem/assemble.hh“

...
typedef VariationalFunctionalAssembler<LinearizationAt<Functional> >
    Assembler;
Assembler assembler(spaces);
...
assembler.assemble(linearization(F,x));
// or with verbosity and fixed number of threads
int nThreads = 4;           // number of threads used for parallel assembly
bool verbose = true;
assembler.assemble(linearization(F,x),7,nThreads,verbose);
// or (same as first call of 'assemble')
assembler.assemble(LinearizationAt<Functional>(F,x),
                     Assembler::VALUE|Assembler::RHS|Assembler::MATRIX);
```

specify what to assemble

Accessing Data – d1

MATHEON

Accessing the first derivative (vector)
(Functional::DomainCache::d1 + Functional::BoundaryCache::d1)

```
VariableSetDescription::template CoefficientVector<>::type
                                                    rhs1(assembler.rhs());

// access d1 for particular row
VariableSetDescription::template CoefficientVector<rowId,rowId+1>::type
                          rhs2(assembler.template rhs<rowId,rowId+1>());
```

Accessing data of row rowId via rhs.data
Type of rhs.data: boost::fusion::vector<
Dune::BlockVector< Dune::FieldVector<Scalar,nComponents> >
>

```
auto rowVar1 = boost::fusion::at_c<rowId>(rhs1.data); // Dune::BlockVector<...>
auto rowVar2 = boost::fusion::at_c<0>(rhs2.data);
```

For Newton steps: F''(x)dx = -F'(x)

```
row *= -1.0;
```

Newton Method

MATHEON

Semi-linear minimization problem:
$$\min_u J(u) = \int_\Omega \left(\frac{1}{2}|\nabla u|^2 + \frac{1}{4}u^4 - fu \right) dx + \int_{\partial\Omega} 10^9 u^2 \, ds$$

```
for(int step=1; step<101; ++step) {                  // second termination criterion:
  std::cout << "Step " << step << std::flush;           // max. 100 steps
  assembler.assemble(linearization(F,u));           // assemble linearization at u

  CoefficientVectors rhs(assembler.rhs());
  rhs *= -1.0;

  AssembledGalerkinOperator<Assembler> A(assembler);  // get differential operator
  // compute Newton step F''(u)du=-F'(u)
  directInverseOperator(A,directType,property).apply(rhs,solution);

  boost::fusion::at_c<0>(u.data) +=                     // new iterate u = u + du
  boost::fusion::at_c<0>(solution.data);

  double energyNorm = sqrt(rhs*solution);
  std::cout << ". Energy norm of du: " << energyNorm << std::endl;

  if(energyNorm < 2e-15) break;                         // stop if energy norm of
}                                                       // du is small
```

Newton Method

Semi-linear minimization problem:

$$\min_u J(u) = \int_\Omega \left(\frac{1}{2}|\nabla u|^2 + \frac{1}{4}u^4 - fu \right) dx + \int_{\partial\Omega} 10^9 u^2 \, ds$$

Grid: 65536 triangles,
98560 edges,
33025 points

number of degrees of freedom = 131585
number of nonzero elements in the stiffness matrix: 820481

Step 1. Energy norm of solution:0.187468
Step 2. Energy norm of solution:0.00155002
Step 3. Energy norm of solution:3.70296e-07
Step 4. Energy norm of solution:2.13386e-14

quadratic convergence (to discrete solution)

Step 5. Energy norm of solution:2.27793e-15
Step 6. Energy norm of solution:2.04263e-15
Step 7. Energy norm of solution:1.99053e-15

machine accuracy reached

total computing time: 25.54s

Accessing Data – d2

Accessing the second derivative (matrix)
(Functional::DomainCache::d2 + Functional::BoundaryCache::d2)

```
AssembledGalerkinOperator<Assembler> A(assembler);

// access d2 for particular ( or consecutive union of) row and column
AssembledGalerkinOperator<Assembler,row,row+1,col,col+1> B(assembler);
AssembledGalerkinOperator<Assembler,firstRow,lastRow,firstCol,lastCol>
                                         C(assembler,onlyLowerTriangle);
```

assemble only lower triangle of matrix

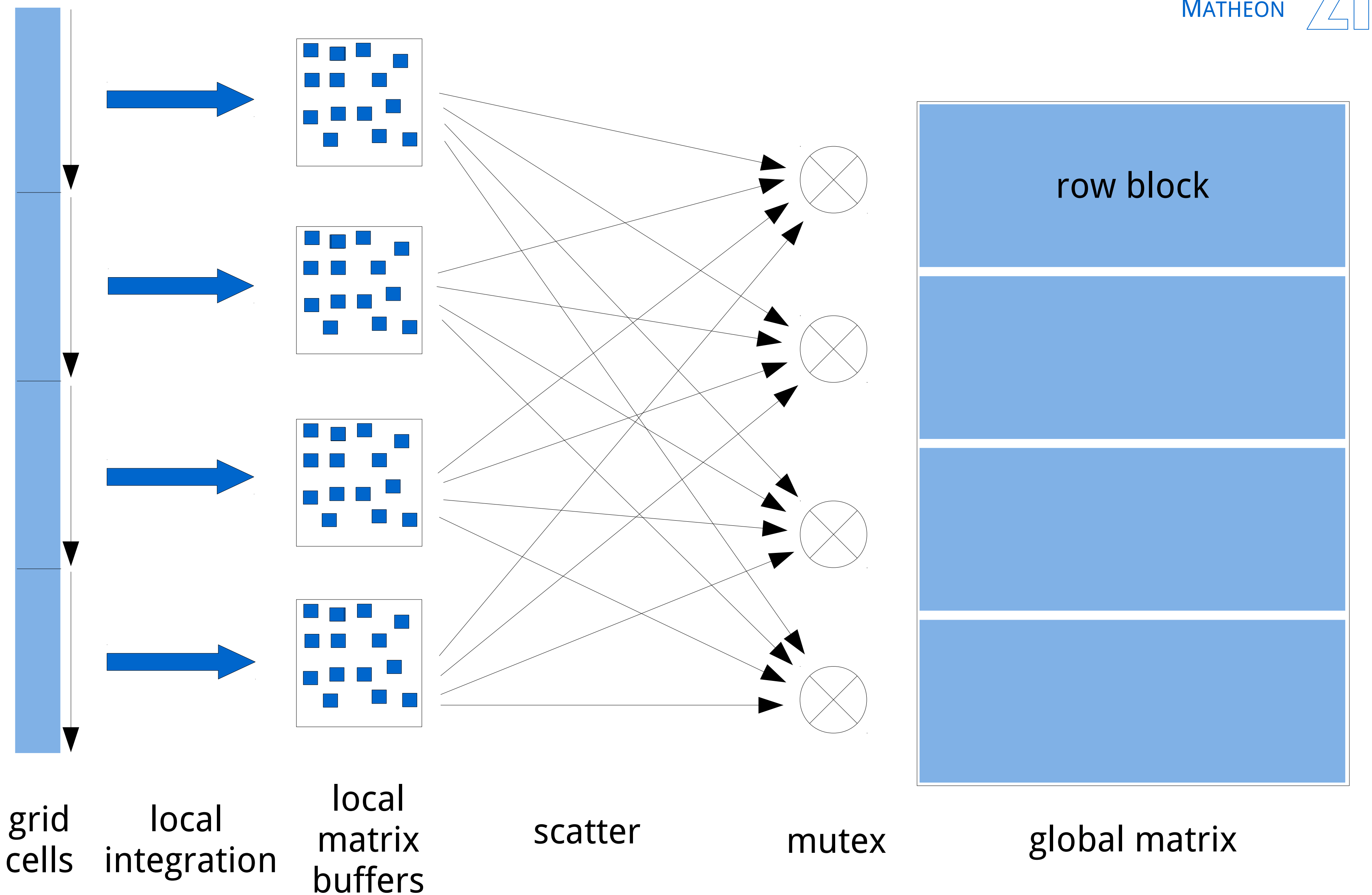
Accessing data

supports move-semantics

```
MatrixAsTriplet<double> M1 = A.template get<MatrixAsTriplet<double> > ();

std::unique_ptr<Dune::BCRSMatrix<Dune::FieldMatrix<double,1,1> > > M2 =
A.template getPointer<Dune::BCRSMatrix<Dune::FieldMatrix<double,1,1> > >();
```

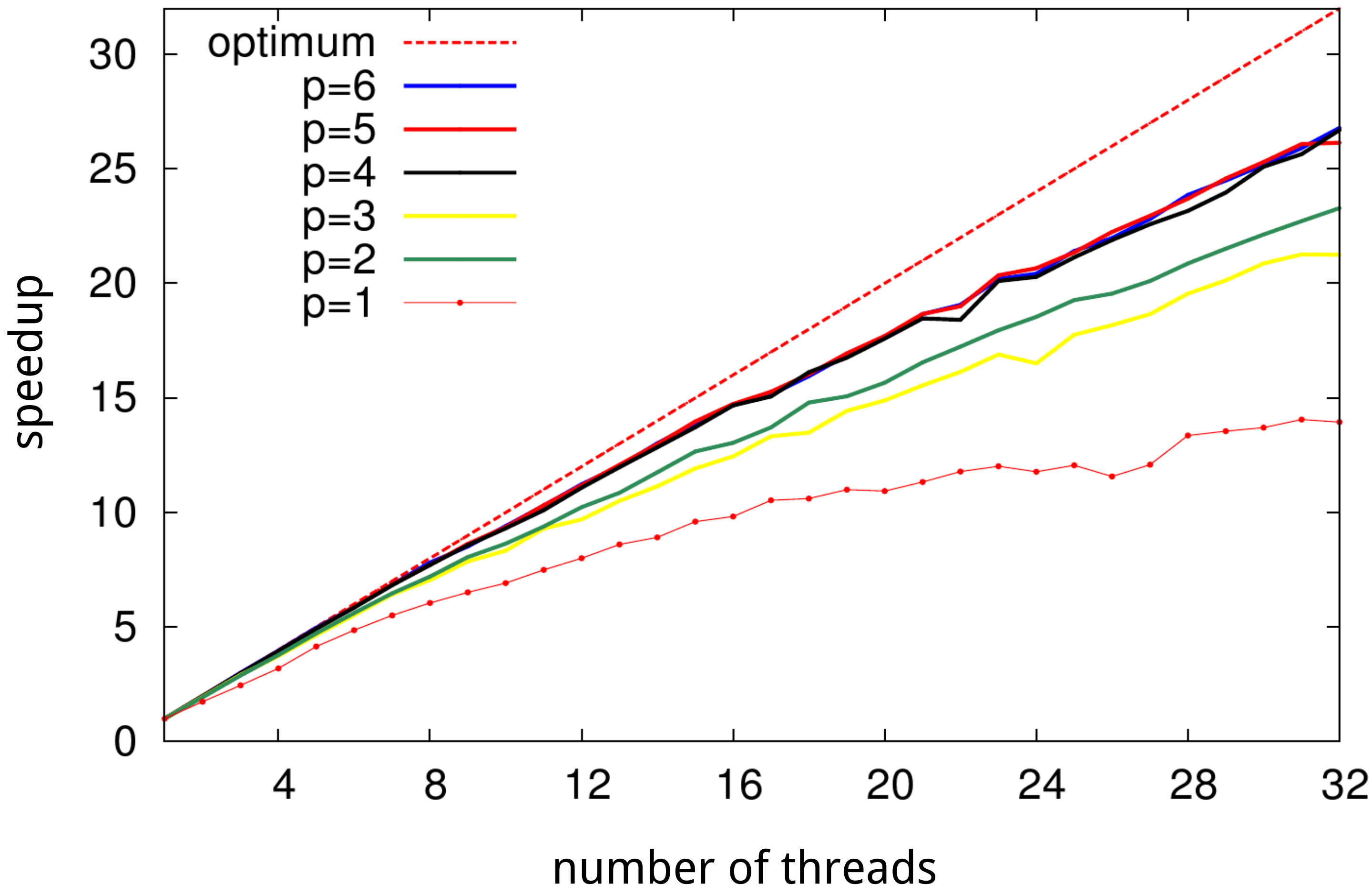
does not (yet) support move-semantics

Multithreaded Assembly



Strong Scaling

- unit cube Poisson problem
- ~ 10^6 degrees of freedom
- 4 x Octacore Xeon E5-4640



Solvers

Interface – Direct Solvers

MATHEON

Motivation

- use Dune concepts to define InverseLinearOperator and DirectSolver

Dune::LinearOperator concept (used for InverseLinearOpertor) $X \to Y$

- template parameters:
  X = domain function space element, Y = range function space element

- public template types:

```
typedef X domain_type;
typedef Y range_type;
typedef typename X::field_type field_type;
```

- virtual functions

```
void apply(const X& x, Y& y) const;
void applyscaleadd(field_type alpha, const X& x, Y& y) const;
```

Interface – Direct Solvers

MATHEON

Motivation

- use Dune concepts to define InverseLinearOperator and DirectSolver

Dune::InverseOperator concept (used for DirectSolver)

$X \to Y$

- template parameters:
  X = domain function space element, Y = range function space element

- public template types:

```
typedef X domain_type;
typedef Y range_type;
typedef typename X::field_type field_type;
```

- virtual functions

```
void apply(X& x, Y& y, Dune::InverseOperatorResult& res);
void apply(X& x, Y& y, double reduction,
                       Dune::InverseOperatorResult& res);
```

for iterative solvers

some information, especially for iterative solvers

Interface – Direct Solvers

Motivation

- use Dune concepts to define InverseLinearOperator and DirectSolver

Dune::Preconditioner concept (used for DirectSolver)

$X \to Y$

- template parameters:
  X = domain function space element (of forward operator),
  Y = range function space element (of forward operator)

- public template types:

```
typedef X domain_type;
typedef Y range_type;
typedef typename X::field_type field_type;
```

- virtual functions

```
void pre(X& x, Y& y);
void apply(X& x, const Y& y);
void post(X& x);
```

Interface – Direct Solvers

MATHEON

Motivation

- use Dune concepts to define InverseLinearOperator and DirectSolver
- wrap different direct solvers into DirectSolver via implementation of Factorization concept

class DirectSolver : public Dune::InverseLinearOperator<Domain,Range>,
public Dune::Preconditioner<Domain,Range>

$$X \to Y$$

- template parameters:
  Domain = domain function space element, Range = range function space element

- public template types:

```
typedef Domain domain_type;
typedef Range range_type;
typedef typename Domain::field_type field_type;
```

default arguments

- concept

```
template <class AssembledGOP>
explicit DirectSolver(AssembledGOP const& A,
DirectType directType=UMFPACK, MatrixProperties properties=GENERAL);
void pre(Domain& x, Range& y){}
void apply(Domain& x, Range& y);                // + other more apply-overloads
void post(Domain& x){}
```

Interface – Direct Solvers

MATHEON

Motivation

- use Dune concepts to define InverseLinearOperator and DirectSolver
- wrap different direct solvers into DirectSolver via implementation of Factorization concept

class InverseLinearOperator : public Dune::LinearSolver<Domain,Range> $X \to Y$

- template parameters:
  InverseOperator, i.e. DirectSolver<...>

- public template types:

```
typedef typename InverseOperator::domain_type domain_type;
typedef typename InverseOperator::range_type range_type;
typedef typename domain_type::field_type field_type;
```

- concept

```
template <class AssembledGOP>
explicit InverseLinearOperator(InverseOperator const& op);
void apply(Domain const& x, Range& y);
void applyscaleadd(field_type alpha, Domain const& x, Range& y);
```

Direct Solver – Usage

Available direct solvers: MUMPS, UMFPACK, UMFPACK3264, UMFPACK64, SUPERLU

Available matrix properties (for MUMPS only!): GENERAL, SYMMETRIC,
POSITIVEDEFINITE,
SYMMETRICSTRUCTURE

UMFPACK solver (default), general matrix (default)

```
directInverseOperator(A).apply(rhs,solution);
```

Direct Solver – Usage

Available direct solvers: MUMPS, UMFPACK, UMFPACK3264, UMFPACK64, SUPERLU

Available matrix properties (for MUMPS only!): GENERAL, SYMMETRIC,
POSITIVEDEFINITE,
SYMMETRICSTRUCTURE

SUPERLU solver, general matrix (default)

```
directInverseOperator(A,SUPERLU).apply(rhs,solution);
```

Direct Solver – Usage

MATHEON ZIB

Available direct solvers: MUMPS, UMFPACK, UMFPACK3264, UMFPACK64, SUPERLU

Available matrix properties (for MUMPS only!): GENERAL, SYMMETRIC,
POSITIVEDEFINITE,
SYMMETRICSTRUCTURE

MUMPS solver, symmetric matrix

```
directInverseOperator(A,MUMPS,SYMMETRIC).apply(rhs,solution);
```

Direct Solver – Usage

MATHEON

Available direct solvers: MUMPS, UMFPACK, UMFPACK3264, UMFPACK64, SUPERLU

Available matrix properties (for MUMPS only!): GENERAL, SYMMETRIC, POSITIVEDEFINITE, SYMMETRICSTRUCTURE

MUMPS solver, symmetric matrix

```
directInverseOperator(A,MUMPS,SYMMETRIC).apply(rhs,solution);
```

If factorization shall be reused (type of auto: InverseLinearOperator<DirectSolver<...>>)

```
auto solver = directInverseOperator(A,MUMPS,SYMMETRIC);
solver.apply(rhs,solution);
```

or

```
Dune::InverseOperator<Domain,Range> const& solver =
                  directInverseOperator(A,MUMPS,SYMMETRIC);
```

Iterative Solvers

MATHEON

Implemented inexact solvers

- Dune:
  - Krylov-subspace solvers: CG, BICGSTAB, MINRES, RestartedGMRES
  - other: gradient solver, loop solver

- Kaskade:
  - Uzawa (for saddle point systems, see tutorial/stokes/stokes.cpp)
  - improved implementations of some of the Dune-Krylov solvers
  - Multigrid

Parameters (except multi grid solver)

- linear operator (derived from Dune::LinearOperator)
- preconditioner (derived from Dune::Preconditioner)
- termination criteria, i.e. max. #steps, (relative) tolerance criterion

Iterative Solvers

MATHEON

CG with Jacobi preconditioner (from tutorial/stationary_heattransfer/ht.cpp)
(=> http://www.dune-project.org/doc-2.2.1/doxygen/html/modules.html
=> Iterative Solvers)

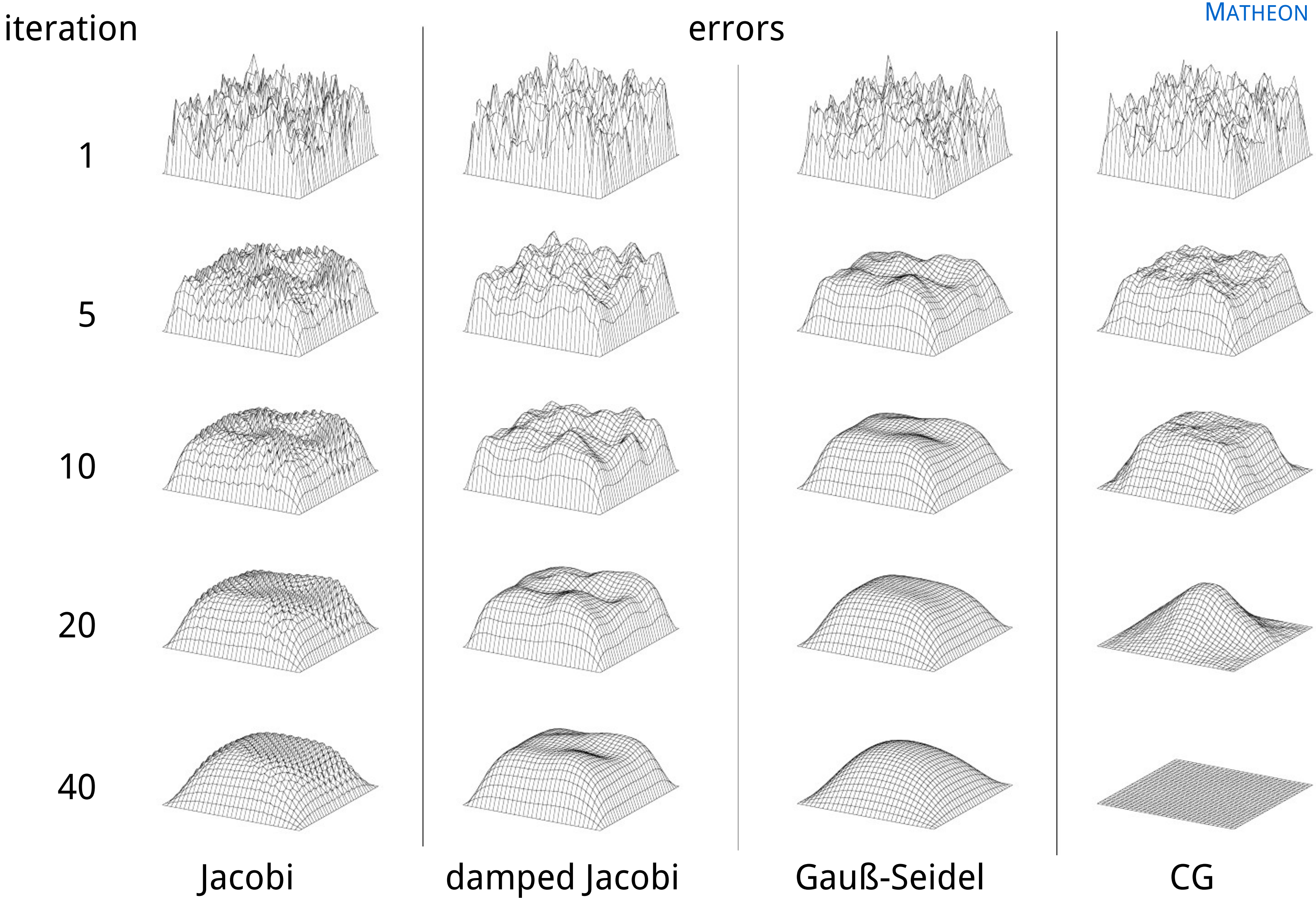
```
typedef VariableSet::CoefficientVectorRepresentation<>::type CoefficientVector;
CoefficientVector rhs(assembler.rhs()),
         sol(VariableSet::CoefficientVectorRepresentation<>::init(variableSet);
rhs *= -1.;              // Newton step => rhs = -F'

AssembledGalerkinOperator<Assembler> A(assembler);

Dune::InverseOperatorResult res;      // stores interesting information on iterative solution process

JacobiPreconditioner<AssembledGalerkinOperator<Assembler> > jacobi(A,1.0);

Dune::CGSolver<CoefficientVector> cg(A,jacobi,iteEps,iteSteps,verbosity);
cg.apply(solution,rhs,res);
```

damping

CG (Dune)
1. termination criterion: rel. residual l^2 decrease
2. termination criterion: max #steps reached
3. optionally: provide own scalar product

more advanced termination criteria available in Kaskade 7

Multigrid: Smoothers



Multigrid

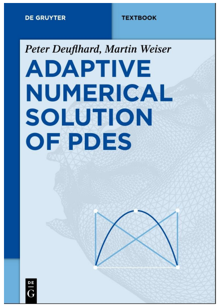
Basic idea for elliptic (diffusive) equations

iterative solvers (damped Jacobi, Gauß-Seidel) remove only spatially high-frequent error components

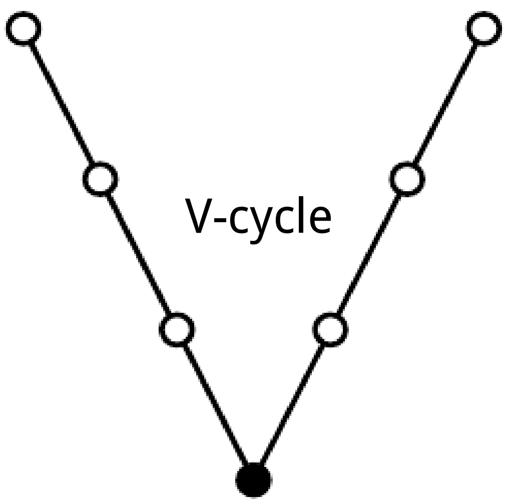
remaining low-frequent error components are themselves high-frequent on coarser grids

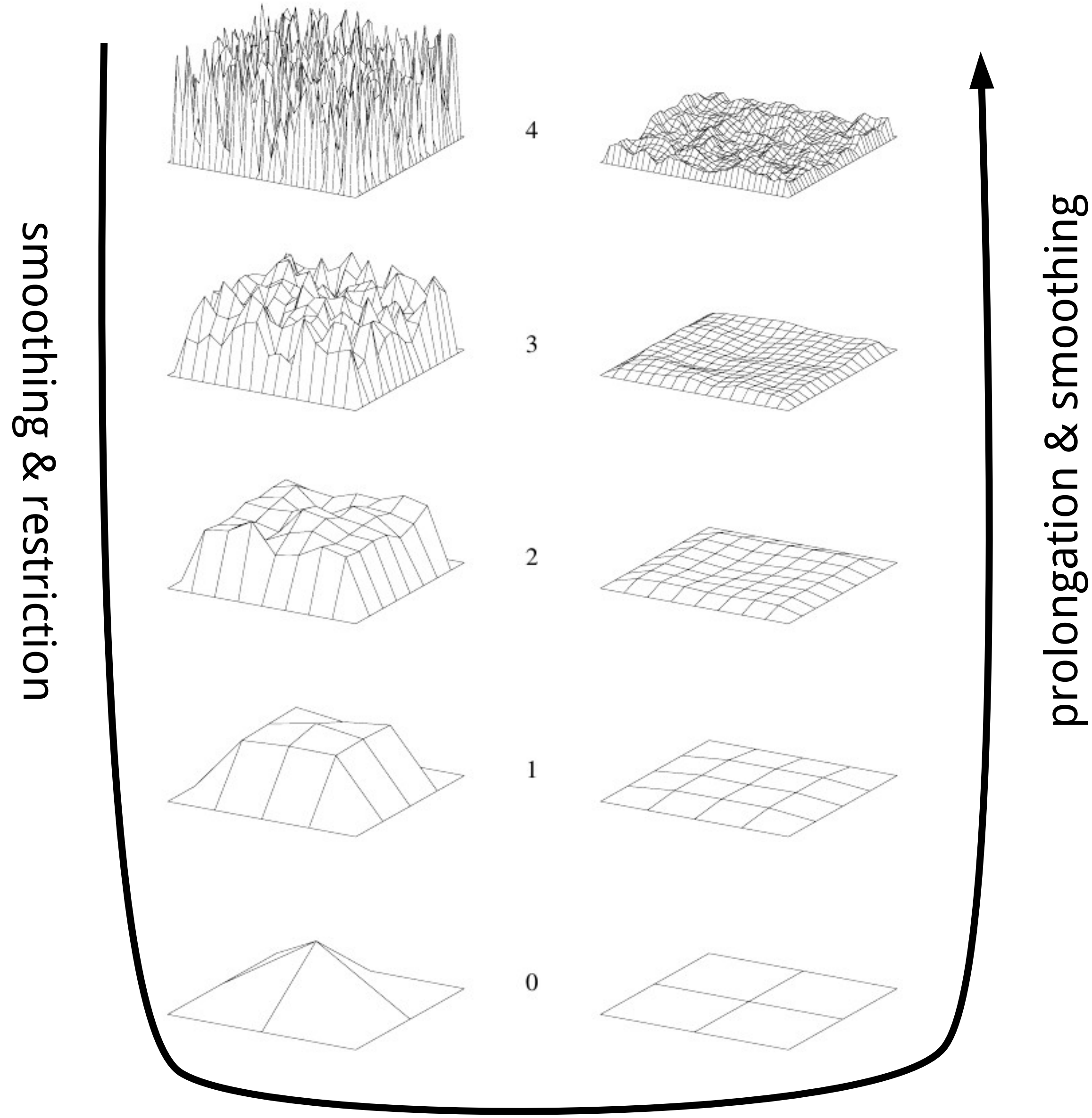
- use smoother on fine grid to remove oscillatory error components
- use smoother on coarse grid to remove low-frequent error

for more details see



Multigrid V-Cycle





Kaskade Multigrid

Parameters (exemplary with default parameters)

```
typename MultiGridSolver<Grid>::Parameters parameter;
parameter.maxSteps = 100;                  // maximal multi grid iterations
parameter.maxSmoothingSteps = 10;          // # pre/post-smoothings
parameter.maxTol = 1e-9;                   // relative tolerance for multi grid
parameter.relaxation = 0.5;        // relaxation factor for Jacobi smoother
parameter.verbose = false;                 // stop talking
```

Usage

```
// solve Ax=b by multigrid
using namespace boost::fusion;
MultigridSolver<Grid> mgSolver(A,gridManager.grid(),parameter);
mgSolver.apply(at_c<id>(x.data),at_c<id>(b.data));
// or
MultigridSolver<Grid>(A,gridManager.grid()).apply(...);
```

Kaskade Multigrid

Notes

- type of A: Dune::BCRSMatrix<...> or AssembledGalerkinOperator<...>
  representing the stiffness matrix on the leaf level

- Works on block-structured data structures
  => Use for PDEs only, in applications to systems with variables that differ in the number of components or the underlying grid dimension code does not compile!
  => Use Dune::BlockVector<...> (i.e. the return type of boost::fusion::at_c<id>(x.data)) instead of VariableSetDescription::CoefficientVectorRepresentation!

Input/Output

Input

Grid generation

- grid factory (Dune, already seen)
- "utilities/gridGeneration.hh":
- createCuboid, createRectangle, createLShape, i.e.

```
GridManager<Grid>
gridManager( createRectangle<Grid>(c0,dc,1.0,true) );
```

Read grid from file

- read output files of 'triangle' (2d): readPolyData
- read Ansys files (works for simplicial and quadrilateral grids): AnsysMeshReader
- read Amira files: AmiraMeshReader, i.e.

```
GridManager<Grid>
gridManager( AmiraMeshReader::readGrid<Grid>(fileName) );
```

- AmiraMeshReader also admits reading different types of data (i.e. function space element's coefficients, material ids, boundary ids, ...)

Output

Write complete VariableSet to VTK-file

```
writeVTKFile(leafView,u,"temperature",options,order);
```

Write function space element to VTK-file

```
auto element = boost::fusion::at_c<0>(u.data);
writeVTK(leafView,element,"temperature",options,order);
```

Write complete VariableSet to AMIRA-file (only linear elements, i.e. point-wise data)

```
writeAMIRAFile(leafView,u,"temperature",options);
```

MATHEON

Error Estimation

Error Estimation Concepts

$$-\Delta u = f$$

Residual error estimators

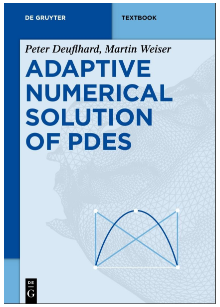
$$\|u_h - u\|_A^2 \le c \left(\sum_{T \in \mathcal{T}} h_T \|\Delta u_h + f\|_{L^2(T)}^2 + \sum_{E \in \mathcal{E}} h_E^{1/2} \|[n^T \nabla u_h]\|_{L^2(E)}^2 \right)$$

Averaging error estimators

∇u_h discontinuous, but ∇u is continuous

➡ project gradient into continuous FE space (improving consistency), difference is gradient error estimator

for more details see



Error Estimation Concepts

Hierarchical error estimators

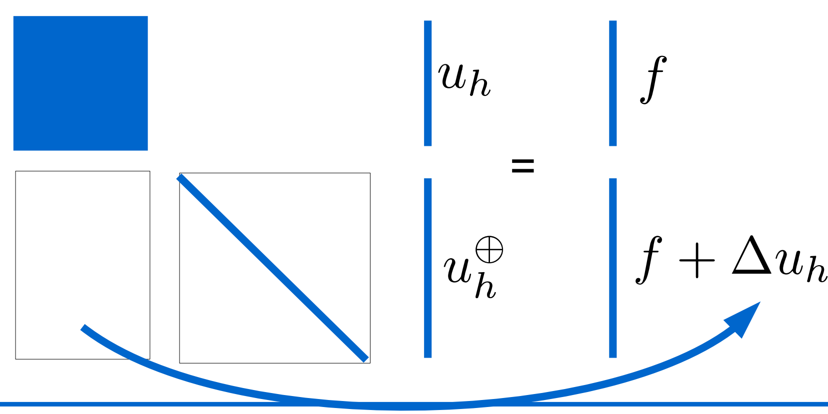
$$u_h \in V_h, \quad u_h^+ \in V_h^+, \quad V_h \oplus V_h^{\oplus} = V_h^+$$

$$\|u_h - u\|_A \approx \|u_h - u_h^+\|_A \qquad \text{if} \quad \|u_h^+ - u\|_A \ll \|u_h - u\|_A$$

embedded: - solve for u_h^+
- obtain u_h by interpolation

DLY: - solve for u_h
- obtain $u_h^{\oplus}$ by approximately solving

$$\int_\Omega \nabla v^T \nabla u_h^{\oplus} \, dx = \int_\Omega (vf - \nabla v^T \nabla u_h) \, dx$$



Kaskade Embedded Estimator

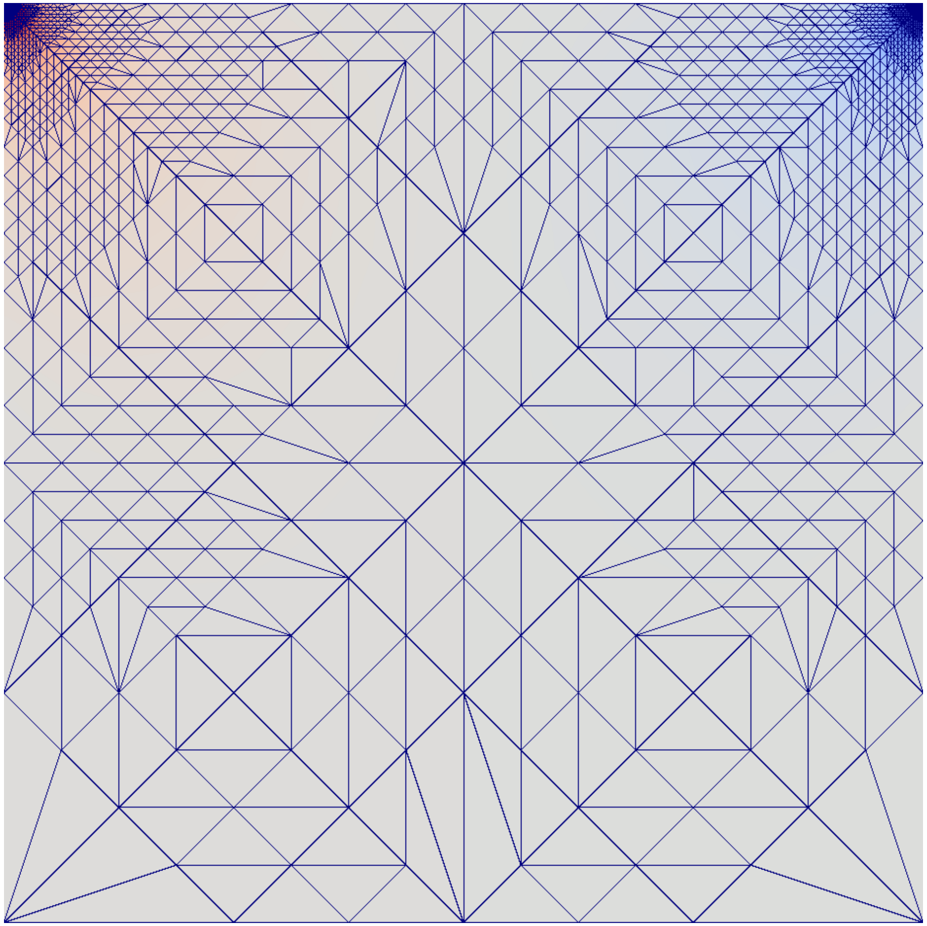
MATHEON

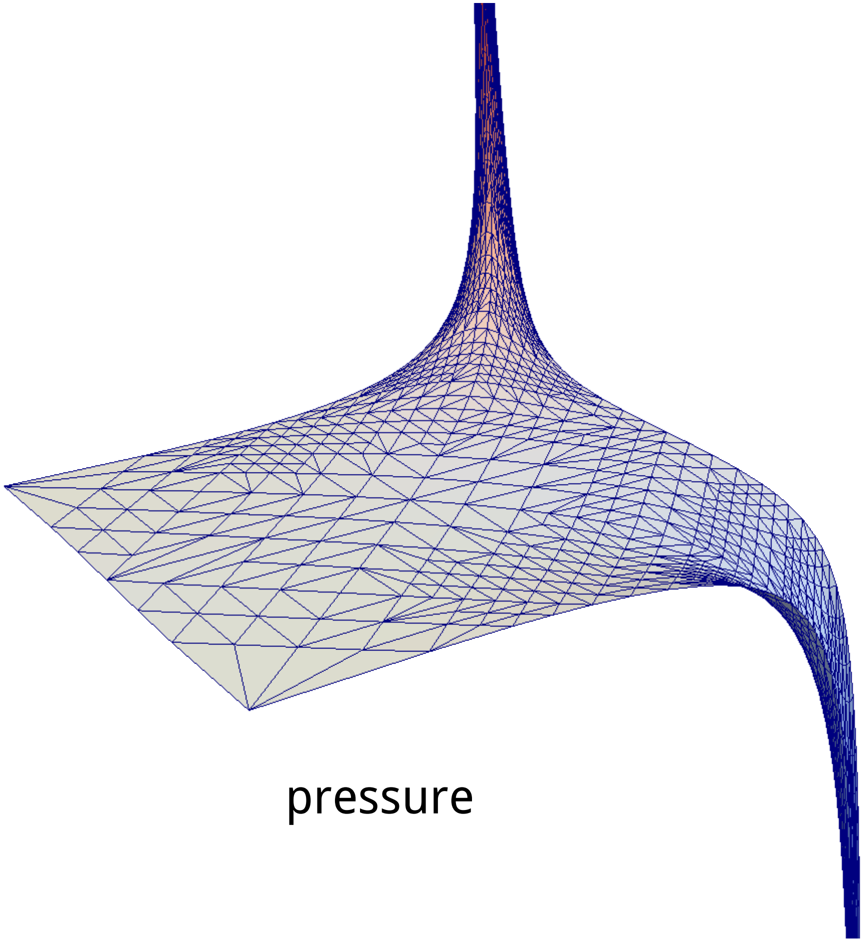
```
#include "fem/embedded_errorest.hh"

…
  // u is VariableSet representation (i.e. bag of FE functions)

  // obtain coarser interpolant
  auto err = u;
  projectHierarchically(varDesc,err);

  // difference is approximate error (of coarser approximation)
  err -= u;

  // define tolerances                                 absolute  relative
  std::vector<std::pair<double,double> > tol = { {1e1, 1e-2} };

  // estimate the error & mark & refine
  EmbeddedErrorEstimator<VarDesc> estimator(gridManager,varDesc);
  bool accurate = estimator.setTolerances(tol).estimate(err,u);
```

accurate: $\|e_i\|_{L^2(\Omega)} \le \mathrm{aTOL}_i^2 + \mathrm{rTOL}_i^2 \|u_i\|_{L^2(\Omega)} \quad \forall i$

Kaskade Embedded Error Estimator

Example: Stokes driven cavity on P3/P2 elements





pressure

Kaskade DLY Estimator

MATHEON

```
typedef FEFunctionSpace<
           ContinuousHierarchicMapper<double,LeafView>> SpaceL;
SpaceL sl(gridManager,gridManager.grid().leafView(),1); // linear FE

typedef FEFunctionSpace<
           ContinuousHierarchicExtensionMapper<double,LeafView>> SpaceQ;
SpaceL sl(gridManager,gridManager.grid().leafView(),2); // quadratic FE

// define VariableSet for quadratic extension
… ExVariableSetDesc;

// define variational functional for extension
typedef HierarchicErrorEstimator<LinearizationAt<Functional>,
                                 ExVariableSetDesc,ExVariableSetDesc> EFunc;

// assemble extension system
VariationalFunctionalAssembler<EFunc> eAssembler(gridManager.signals,spaces);
eAssembler.assemble(Efunc(linearization(func,u0),u));
```

original variational functional

linearization point

linear FE solution

Adaptive Mesh Refinement

```
// mark certain cells for refinement
for (auto ci=gridView.template begin<0>();
                ci!=gridView.template end<0>();
                ++ci)
  if (wantRefine(*ci))
    gridManager.mark(*ci);

// actually refine the grid
gridManager.adaptAtOnce();
```

calls FE spaces to automatically prolongate all FE functions

mesh refinement example

MATHEON

Time Stepping

Parabolic Problems

MATHEON

general form $B(u,t)\dot{u} = F(u,t)$ d1

examples

$\dot{u} = \mathrm{div}(\sigma \nabla u)$ (heat equation)

$\dot{u} = \mathrm{div}(\sigma \nabla u) + u(u-.1)(u-1)$ (Kolmogorov equation)

$B = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \longrightarrow$ $\boxed{\begin{aligned} \dot{u} &= s + \nu \Delta u - u_x u - \nabla p \\ 0 &= \mathrm{div}\, u \end{aligned}}$ (Navier Stokes)

Problem definition

```
   // in DomainCache
   template <int row, int col>
   double b2_impl(VariationalArg<double,dim,m> const& test,
                  VariationalArg<double,dim,m> const& ansatz);

// in weak formulation
template <int row, int col>
struct B2 {
   static int const present = row==col;
};
```

Time Stepping

MATHEON

$$B\dot{u} = F(u)$$

Linearly implicit Euler $$B\frac{u_{k+1}-u_k}{\tau} = F(u_{k+1}) \approx F(u_k) + F'(\tilde{u})(u_{k+1}-u_k)$$

$$\Rightarrow \quad (B - \tau F'(\tilde{u}))\delta u_k = \tau F(u_k)$$

Example: heat equation $$(I - \tau\Delta)\delta u_k = \tau\Delta u_k$$

elliptic differential operator

$\Rightarrow$ solve a stationary 2nd order PDE
(method of time layers, Rothe's method)

Implementation

need a variational functional / weak formulation
of stationary linearly implicit Euler problem

```
typedef SemiImplicitEulerStep<ParabolicEquation> EulerProblem;
EulerProblem eulerProblem(heatEquation,tau);
```

Assembly Process

Assemble time layers (i.e. for semi-linear time stepping schemes)

```
#include „fem/assemble.hh"

…
typedef SemiImplicitEulerStep<HeatEquation> EulerStep;
typedef VariationalFunctionalAssembler<SemiLinearizationAt<EulerStep> > Assembler;
Assembler assembler(spaces);
...
assembler.assemble(semilinearization(EulerStep(heatEquation,dt),x_t,x_d,dx));
```

Assembly Process

MATHEON ZIB

Assemble time layers (i.e. for semi-linear time stepping schemes)

```
#include „fem/assemble.hh“

…
typedef SemiImplicitEulerStep<HeatEquation> EulerStep;
typedef VariationalFunctionalAssembler<SemiLinearizationAt<EulerStep> > Assembler;
Assembler assembler(spaces);
...
assembler.assemble(semilinearization(EulerStep(heatEquation,dt),x_t,x_d,dx));
```

point of linearization for (abstract) time differential operator

point of linearization spatial differential operator

correction of last time step only relevant if B depends on u

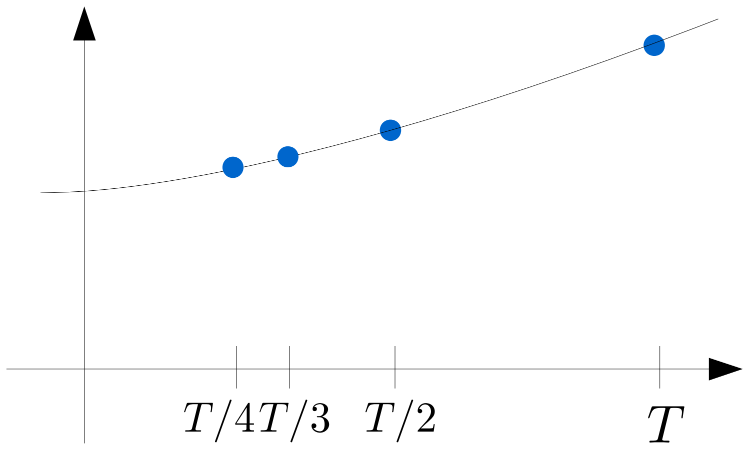
Assembly Process

Assemble time layers (i.e. for semi-linear time stepping schemes)

```
#include „fem/assemble.hh“

…
typedef SemiImplicitEulerStep<HeatEquation> EulerStep;
typedef VariationalFunctionalAssembler<SemiLinearizationAt<EulerStep> > Assembler;
Assembler assembler(spaces);
…
// assemble Euler step
assembler.assemble(semilinearization(EulerStep(heatEquation,dt),x_t,x_d,dx));
AssembledGalerkinOperator<Assembler> a;

// solve system
VariableSetDescription::CoefficientVectorRepresentation<>::type sol(...);
directInverseOperator(a).apply(assembler.rhs(),sol);

// add increment
x_t += sol;
```

Higher Order Integrators

Extrapolation



asymptotic expansion

$$u(T;\tau) = u(T) + \sum_{i=1}^{r} c_i \tau^i + \tau^{r+1} R(T,\tau)$$

polynomial interpolation

$$\hat{u}(T;\tau) = u_0 + \sum_{i=1}^{r} \hat{c}_i \tau^i$$

evaluate at $\tau_i = T/i, \quad i = 1, \dots, r+1$

$$u(T) = u(T;0) \approx \hat{u}(T;0) = u_0$$

stiff problems (parabolic):
linearly implicit Euler (LIMEX)

non-stiff (hyperbolic):
explicit midpoint rule (DIFEX)

LIMEX

```
#include "timestepping/limex.hh"

Limex<HeatEquation> limex(gridManager,heatEquation,
                          variableSetDescription);
auto const& dx = limex.step(x,dt,order);

// estimate error
if (limex.estimateError(x,order,order-1) < rTOL)
  x += dx;
else
  // repeat step
```