AMDiS

Adaptive Multi-Dimensional Simulations

AMDiS - Poisson equation

```cpp
#include <amdis/AMDiS.hpp>                                  // Include essential headers
#include <amdis/LocalOperators.hpp>
int main(int argc, char** argv)
{
  using namespace AMDiS;                                    // Namespace of all AMDiS functions
  using namespace Dune::Functions::BasisFactory;            // Use DUNE facilities directly

  Environment env{argc, argv};                              // Initialize Initfile, MPI...

  Dune::YaspGrid<2> grid{ {1.0,1.0}, {8,8} };               // Grid: Unit square domain
  ProblemStat prob{"example", grid, lagrange<2>()};         // Lagrange elements of deg. 2
  prob.initialize(INIT_ALL);                                // Prepare data structures

  auto c = [](auto x) { ... };
  prob.addMatrixOperator(sot(1.0));                         // a(v,u) = <grad(v), grad(u)>
  prob.addVectorOperator(zot(c, 6));                        // b(v) = <v, c(x)>
  prob.addDirichletBC([&](auto x) { return b(x); },         // boundary
                      [&](auto x) { return g(x); });        // u = g | on boundary

  AdaptInfo adaptInfo{"adapt"};
  prob.assemble(adaptInfo);                                 // Assemble the linear system
  prob.solve(adaptInfo);                                    // Solve the linear system
  prob.writeFiles(adaptInfo);                               // Write solution to file
}
```

Installing AMDiS and its dependencies

Find install instructions in README.md file or at https://amdis.readthedocs.io

1. Download DUNE modules: common, geometry, grid, localfunctions, istl, typetree, functions
2. Download AMDiS (into the same `<dune-base>` directory as all DUNE modules)

```
git clone https://gitlab.mn.tu-dresden.de/amdis/amdis-core.git
```

Installing AMDiS and its dependencies

Find install instructions in README.md file or at https://amdis.readthedocs.io

1. Download DUNE modules: common, geometry, grid, localfunctions, istl, typetree, functions
2. Download AMDiS (into the same `<dune-base>` directory as all DUNE modules)

```
git clone https://gitlab.mn.tu-dresden.de/amdis/amdis-core.git
```

3. Configure and build everything

```
./dune-common/bin/dunecontrol cmake -DCMAKE_BUILD_TYPE=Release
./dune-common/bin/dunecontrol make -j10
```

My first AMDiS project

Similar to DUNE, we provide a script `amdisproject` that helps you create a simple initial empty project:

```
./amdis-core/bin/amdisproject my-first-project
```

Some question are asked about name, version, maintainer, and dependencies. After answering everything, a directory `my-first-project` is created in `<dune-base>` with a `CMakeLists.txt` file and some example code.

My first AMDiS project

Similar to DUNE, we provide a script `amdisproject` that helps you create a simple initial empty project:

```
./amdis-core/bin/amdisproject my-first-project
```

Some question are asked about name, version, maintainer, and dependencies. After answering everything, a directory `my-first-project` is created in `<dune-base>` with a `CMakeLists.txt` file and some example code.

```
- README.md
- CMakeLists.txt
- config.h.cmake
- dune.module
- src/CMakeLists.txt
- src/my-first-project.cpp
- amdis/my-first-project/CMakeLists.txt
- amdis/my-first-project/MyFirstProject.hpp
- doc/doxygen/CMakeLists.txt
- doc/doxygen/Doxylocal
- cmake/modules/CMakeLists.txt
- cmake/modules/MyFirstProjectMacros.cmake
- init/CMakeLists.txt
- init/my-first-project.dat
- macro/CMakeLists.txt
- macro/my-first-project.Xd.amc
```

My first AMDiS project

The source file `src/my-first-project.cpp` contains the following code snippet:

```
#include <amdis/AMDiS.hpp>

using namespace AMDiS;
int main(int argc, char** argv)
{
  Environment env{argc, argv};

  // your code comes here
}
```

My first AMDiS project

The source file `src/my-first-project.cpp` contains the following code snippet:

```
#include <amdis/AMDiS.hpp>

using namespace AMDiS;
int main(int argc, char** argv)
{
  Environment env{argc, argv};

  // your code comes here
}
```

- Main include header `<amdis/AMDiS.hpp>` includes *some* parts of the library.

My first AMDiS project

The source file `src/my-first-project.cpp` contains the following code snippet:

```
#include <amdis/AMDiS.hpp>

using namespace AMDiS;
int main(int argc, char** argv)
{
  Environment env{argc, argv};

  // your code comes here
}
```

- Main include header `<amdis/AMDiS.hpp>` includes *some* parts of the library.
- All classes and functions can be found in the namespace `AMDiS`

My first AMDiS project

The source file `src/my-first-project.cpp` contains the following code snippet:

```
#include <amdis/AMDiS.hpp>

using namespace AMDiS;
int main(int argc, char** argv)
{
  Environment env{argc, argv};

  // your code comes here
}
```

- Main include header `<amdis/AMDiS.hpp>` includes *some* parts of the library.
- All classes and functions can be found in the namespace `AMDiS`
- `Environment` initializes some global states, like MPI and PETSc

My first AMDiS project

The source file `src/my-first-project.cpp` contains the following code snippet:

```
#include <amdis/AMDiS.hpp>

using namespace AMDiS;
int main(int argc, char** argv)
{
  Environment env{argc, argv};

  // your code comes here
}
```

- Main include header `<amdis/AMDiS.hpp>` includes *some* parts of the library.
- All classes and functions can be found in the namespace `AMDiS`
- `Environment` initializes some global states, like MPI and PETSc
- The `CMakeLists.txt` file in the `src/` folder contains the executable.

My first AMDiS project

The directory `amdis/my-first-project/` can be used to place header files shared my several source files in this project. An example is given in `MyFirstProject.hpp`:

```
#pragma once

namespace AMDiS::MyFirstProject
{
  // add your classes here
}
```

that can be included in the source file:

```
#include <amdis/AMDiS.hpp>
#include <amdis/my-first-project/MyFirstProject.hpp>

using namespace AMDiS;
int main(int argc, char** argv) { /* ... */ }
```

My first AMDiS project

The directory `amdis/my-first-project/` can be used to place header files shared my several source files in this project. An example is given in `MyFirstProject.hpp`:

```
#pragma once

namespace AMDiS::MyFirstProject
{
  // add your classes here
}
```

that can be included in the source file:

```
#include <amdis/AMDiS.hpp>
#include <amdis/my-first-project/MyFirstProject.hpp>

using namespace AMDiS;
int main(int argc, char** argv) { /* ... */ }
```

Configure, compile and link it against all dependencies, by using `dunecontrol`:

```
./dune-common/bin/dunecontrol --only=my-first-project all
```

AMDiS - The `ProblemStat` class

AMDiS - Variational problem

Find $(u_0, u_1) \in U_0 \times U_1$ s.t.

$$A((v_0, v_1), (u_0, u_1)) = B((v_0, v_1)) \quad \forall (v_0, v_1) \in U_0' \times U_1'$$

with (bi-)linear form

$$A((v_0, v_1), (u_0, u_1)) = a_{00}(v_0, u_0) + a_{01}(v_0, u_1) + a_{10}(v_1, u_0) + a_{11}(v_1, u_1) + \bar{a}((v_0, v_1), (u_0, u_1))$$
$$B((v_0, v_1)) = b_0(v_0) + b_1(v_1) + \bar{b}((v_0, v_1))$$

Here a_{ij}, $\bar{a}$, b_i, and $\bar{b}$ are "simple" components of the (bi-)linear form and are called *Operators* in the following.

AMDiS - Variational problem

Often, the *Operators* have a common structure

$$a(v,u) = \int_\Omega \mathfrak{a}(c,v,u,\nabla v,\nabla u)\,\mathrm{dx} = \sum_{T\in\mathcal{T}_h}\int_T \mathfrak{a}(c,v,u,\nabla v,\nabla u)\,\mathrm{dx}$$

with $c : \Omega \to \mathbb{R}$ a coefficient function. Sometimes we have vector-valued or matrix-valued coefficients. All these are defined with the generic c expression.

We call the integrand $\mathfrak{a}$ the *OperatorTerm* associated to the *Operator* a. The operator is an operator term bound to the grid (view) it is assembled on.

Examples:

- $\mathfrak{a}(c,v,u,\nabla v,\nabla u) = c(x)v(x)u(x)$ called `test_trial` or `zot`.
- $\mathfrak{a}(c,v,u,\nabla v,\nabla u) = c(x)\nabla v(x)\nabla u(x)$ called `gradtest_gradtrial` or `sot`.

AMDiS - Variational problem

The *Problem* is described by

- A *grid* it is defined on
- The (composite) *basis* $U_0 \times U_1$
- *Constraints* on the trial and test basis, e.g., Dirichlet constraints or periodic constraints
- An operator *matrix* A and an operator *rhs*-vector B
  - Composed of individual operators, defined by operator terms
- A *solution* vector $U = (u_0, u_1)$
  - Might be decomposed into components

All these problem components + some additional management/visualization tools are combined into the class `ProblemStat`.

AMDiS - The `ProblemStat` class

Main interface class: The class template `ProblemStat` is parametrized with `BasisTraits`, i.e., a description of the grid and global basis.

```
template <class BasisTraits>
class ProblemStat;
```

AMDiS - The `ProblemStat` class

Main interface class: The class template `ProblemStat` is parametrized with `BasisTraits`, i.e., a description of the grid and global basis.

```
template <class BasisTraits>
class ProblemStat;
```

Can be constructed in multiple ways

```
ProblemStat (std::string name);                 // requires `BasisTraits`

template <class Grid>
ProblemStat (std::string name, Grid& grid);     // requires `BasisTraits`

template <class Grid, class Basis>
  requires Concepts::GlobalBasis<Basis>
ProblemStat (std::string name, Grid& grid, Basis& globalBasis);         // CTAD

template <class Grid, class BF>
  requires Concepts::PreBasisFactory<BF>
ProblemStat (std::string name, Grid& grid, BF const& basisFactory);     // CTAD
```

AMDiS - The `ProblemStat` class

Main Setup functions

Create the basis and the grid, initialize internal data structures for the solution vector and the linear system, create adaption markers and file writers.

```
// Initialize the problem.
//
// Parameters read in initialize()
//   - `[GRID_NAME]->global refinements`:  nr of initial global refinements
//
void initialize (Flag initFlag, ProblemStat* adoptProblem = [...], Flag adoptFlag = [...]);

// Read the grid and solution from backup files and initialize the problem
//
// Parameters read in restore() for problem with name 'PROB'
//   - `[PROB]->restore->grid`:      name of the grid backup file
//   - `[PROB]->restore->solution`:  name of the solution backup file
//
void restore (Flag initFlag);
```

AMDiS - The `ProblemStat` class

Access to grid and basis

After initialization you can access the internally stored grid and global basis:

```
// Return the stored grid
std::shared_ptr<Grid> grid ();

// Return the stored basis
std::shared_ptr<GlobalBasis> globalBasis ();

// Return the gridView of the basis
GridView gridView () const;
```

AMDiS - The `ProblemStat` class

Access to data structures

The `ProblemStat` internally stores the data structures for the linear system to assemble and for the solution vector. All these data structure can be accessed directly:

```
// Return the system-matrix -> BiLinearForm<GlobalBasis, GlobalBasis, [...]>
std::shared_ptr<SystemMatrix> systemMatrix();

// Return the system rhs vector -> LinearForm<GlobalBasis, [...]>
std::shared_ptr<SystemVector> rhsVector();

// Return the solution vector -> DOFVector<GlobalBasis, [...]>
std::shared_ptr<SolutionVector> solutionVector();
```

AMDiS - The `ProblemStat` class

Access to data structures

The `ProblemStat` internally stores the data structures for the linear system to assemble and for the solution vector. All these data structure can be accessed directly:

```
// Return the system-matrix -> BiLinearForm<GlobalBasis, GlobalBasis, [...]>
std::shared_ptr<SystemMatrix> systemMatrix();

// Return the system rhs vector -> LinearForm<GlobalBasis, [...]>
std::shared_ptr<SystemVector> rhsVector();

// Return the solution vector -> DOFVector<GlobalBasis, [...]>
std::shared_ptr<SolutionVector> solutionVector();
```

The solution can also be extracted as *DiscreteFunction* $\rightarrow$ see later

```
// Return a discrete function wrapper representing a (sub-)view onto the solution
template <class Range = void, class... Indices>
auto solution (Indices... ii);
```

AMDiS - The `ProblemStat` class

Describe your PDE

(Bi-)linear form is composed of smaller components, called *OperatorTerms*.

```
// Add an operator term to the bilinear form
template <class OperatorTerm, class RowTreePath = [...], class ColTreePath = [...]>
void addMatrixOperator (OperatorTerm const& op, RowTreePath row = {}, ColTreePath col = {})

// Add an operator term to the linear form
template <class OperatorTerm, class TreePath = [...]>
void addVectorOperator (OperatorTerm const& op, TreePath path = {})
```

AMDiS - The `ProblemStat` class

Describe your PDE

(Bi-)linear form is composed of smaller components, called *OperatorTerms*.

```
// Add an operator term to the bilinear form
template <class OperatorTerm, class RowTreePath = [...], class ColTreePath = [...]>
void addMatrixOperator (OperatorTerm const& op, RowTreePath row = {}, ColTreePath col = {})

// Add an operator term to the linear form
template <class OperatorTerm, class TreePath = [...]>
void addVectorOperator (OperatorTerm const& op, TreePath path = {})
```

If terms are to be evaluated on (part of) the boundary of the grid only:

```
// Add an operator term to the bilinear form
template <class OperatorTerm, class RTP = [...], class CTP = [...]>
void addMatrixOperator (BoundaryType b, OperatorTerm const& op, RTP row = {}, CTP col = {})

// Add an operator term to the linear form
template <class OperatorTerm, class TP = [...]>
void addVectorOperator (BoundaryType b, OperatorTerm const& op, TP path = {})
```

AMDiS - The `ProblemStat` class

Describe your PDE

Constraints can be added by

```
// Enforce Dirichlet boundary values for the solution vector on boundary regions
template <class Boundary, [class RowTreePath, class ColTreePath,] class Values>
void addDirichletBC (Boundary const& predicate,
                     [RowTreePath row, ColTreePath col,]
                     Values const& values);

// Add a periodic boundary conditions to the system, by specifying a face transformation
// y = A*x + b of coordinates. We assume, that A is orthonormal.
void addPeriodicBC (BoundaryType id, WorldMatrix const& A, WorldVector const& b);

// General constraints specification
void addConstraint (BoundaryCondition<SystemMatrix,SolutionVector,SystemVector> constraint);
```

AMDiS - The `ProblemStat` class

Adaptation and solving the problem

The following steps can be called to solve the PDE system:

- `markElements`: use added markers to mark elements for coarsen/refine
- `adaptGrid`: coarsen/refine the grid
- `assemble`: assemble the stiffness matrix and right hand side
- `solve`: solve the linear system above
- `writeFiles`: write files to output

AMDiS - The `ProblemStat` class

Adaptation and solving the problem

The following steps can be called to solve the PDE system:

- `markElements`: use added markers to mark elements for coarsen/refine
- `adaptGrid`: coarsen/refine the grid
- `assemble`: assemble the stiffness matrix and right hand side
- `solve`: solve the linear system above
- `writeFiles`: write files to output

Default implementations handle most cases

- `ProblemStat` interface allows to customize markers, interpolation during refinement, file writers, linear solvers [, assemblers],...
- Many solvers and file writers supported out of the box

AMDiS - The `ProblemStat` class

Customization

User can inherit from `ProblemStat` and replace the implementations freely, e.g. override the `assemble()` method

- adaptation subroutines can be called individually, so a user can stop after a certain step, run some user code and then resume with the default AMDiS implementations
- almost all member functions can be accessed / overridden to allow maximum flexibility

Exercise 6

Exercise 6

- Create a new amdis module `amdis-exercise` using the tool `amdisproject`
- Copy the initial example of the Poisson equation in the file `src/amdis-exercise.cc`
- Implement the missing functions with some meaningful content.
- Configure, compile and run the code.

Exercise 6

- See the default init-file in the project folder, adapt its content to allow configuration of the Poisson problem. See https://amdis.readthedocs.io/en/latest/reference/Initfile/ for some infos about the parameter file
- Configure the automatic output of the solution of the problem
- Run your code again with

```
./build-cmake/src/amdis-exercise init/amdis-exercise.dat
```

- Visualize the result in ParaView