R. Hiptmair
L. Scarabosio
C. Urzua Torres

Spring Term 2015

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 10

Problem 10.1 Dirichlet BVP with Delta-Function Right-Hand Side (Core problem)

One of the messages of [NPDE, Section 2.4.2] was that not all linear variational problems on function spaces are well-posed, see [NPDE, Def. 2.4.12]. In this problem we will encounter a specimen of an innocent looking ill-posed linear variational problem that was discussed in detail in [NPDE, Ex. 2.4.18]. We are going to study how the ill-posed nature of the continuous variational problem will be reflected by the behavior of Galerkin solutions. This will be done empirically based on a `C++` implementation using DUNE.

Template files for the new classes you will need to write are available in the lecture `svn` repository

`assignments_codes/assignment10/Problem1`

The idea is that you extend your own code, reason why it does not contain the files you already implemented in the previous assignments. Please do not forget to include them when you submit your work. If your own implementation is still not working, you may use the classes provided in `assignments_codes/solutions`.

In this problem we consider the linear variational problem

$$u \in H_0^1(\Omega): \quad \int_\Omega \operatorname{grad} u(\boldsymbol{x}) \cdot \operatorname{grad} v(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = v(0) \quad \forall v \in H_0^1(\Omega), \tag{10.1.1}$$

posed on a polygon Ω (with $0 \in \Omega$) and attempt its numerical solution by means of a linear finite element Galerkin discretization.

(10.1a) Describe a physical system for which (10.1.1) provides a model of certain aspects. What is the meaning of u in this model.

HINT: Remember a particular model discussed in [NPDE, Chapter 2].

Solution: This is similar to the membrane problem [NPDE, Section 2.2.1] with an "infinitesimally narrow" point force at $\boldsymbol{x} = 0$. In effect, $f = \delta$, the Dirac delta function.

(10.1b) Which problem haunts the linear variational problem (10.1.1)? What can you say about the existence of a minimizer of the associated quadratic functional?

HINT: Refresh yourself on the contents of [NPDE, Section 2.2.3]. Again study the beginning on [NPDE, Section 2.3] and [NPDE, Ex. 2.4.18].

Solution: As shown in [NPDE, Ex. 2.4.18], the linear form $\ell(v) = v(0)$ is not continuous with respect to $\|v\|_{\mathsf{a}} = \left(\int_\Omega \|\operatorname{grad} v\|^2 \,\mathrm{d}\boldsymbol{x}\right)^{\frac{1}{2}}$ on $H_0^1(\Omega)$. The associated quadratic functional will not be bounded from below and has no minimizer.

(10.1c) Let $\mathcal{M}$ be a triangular mesh of Ω. The Galerkin discretization of (10.1.1) based on $\mathcal{S}_{1,0}^0(\mathcal{M})$ leads to a discrete variational problem, *cf.* [NPDE, Section 3.2]. Explain why the associated discrete quadratic minimization problem (see [NPDE, Eq. (1.5.6)]) always has a unique solution.

Solution: On the restricted space $\mathcal{S}_{1,0}^0(\mathcal{M})$, ℓ *is* in fact continous with respect to $\|\cdot\|_{\mathsf{a}}$. (All functionals are continuous in finite dimensional spaces.)

(10.1d) Let $\mathcal{M}_k$ be a sequence of triangular meshes, where $\mathcal{M}_{k+1}$ is generated from $\mathcal{M}_k$ by regular refinement of all triangles. Guess how the minimal value of the quadratic functional associated with the variational problem (10.1.1) will behave as $k \to \infty$.

Solution: Since there is no minimizer, we might expect the value of the quadratic functional to diverge towards negative infinity.

(10.1e) Implement the class LocalDelta which provides a method

```
template <class Element>
  void operator()(Element const& e, ElementVector &local) const;
```

to compute the element vector associated to the specific right-hand side of (10.1.1). The method should check whether 0 belongs to the given element and then compute the corresponding values using linear Lagrangian finite elements.

HINT: Note this is similar to LocalFunction implemented in subproblem (7.4e).

Solution: See Listing 10.1 for the code.

Listing 10.1: Implementation for `LocalDelta`

```
#ifndef LOCALDELTA_HPP_
#define LOCALDELTA_HPP_

#include <stdexcept>
#include <vector>
#include <dune/localfunctions/lagrange/pk.hh>
#include <dune/common/exceptions.hh>
#include <dune/common/fvector.hh>

namespace NPDE15{

  class LocalDelta{
  public:
    using calc_t=double;

    LocalDelta(){ x_0 = 0.0;};

    template <class Vector, class Element>
    void operator()(Element const& e, Vector &local) const;
```

```
  private:
    Dune::FieldVector<double,2> x_0;
  };

  template <class Vector, class Element>
  void LocalDelta::operator()(Element const& e, Vector &local) const{
    Dune::PkLocalFiniteElement<calc_t,calc_t,Element::mydimension,1> localFE;
    assert(localFE.type()==e.type());
    unsigned M = localFE.localBasis().size();
    local = Vector(M);
    for (unsigned i=0;i<M;++i)
      local[i]=0.;
    auto const& egeom = e.geometry();
    auto local_x0 = egeom.local(x_0) ;
    if(local_x0[0]>=0 && local_x0[0]<=1 && 1-local_x0[1] - local_x0[0]>=0 && local_x0[1]>=0 && local_x0[1]<=1){
   std::vector<Dune::FieldVector<calc_t,1> > shapefct_values;
    localFE.localBasis().evaluateFunction(local_x0, shapefct_values);
      for (unsigned i=0;i<M;++i)  local[i] = shapefct_values[i];
    }
  };

}
#endif
```

(10.1f) We want to equip our finite element code with the capability to computer the $L^2(\Omega)$- and $H^1(\Omega)$-seminorm of piecewise linear finite element functions given through their coefficient vectors with respect to the nodal basis of $\mathcal{S}_1^0(\mathcal{M})$.

For this reason, now you are asked to complete the class `Norms` by implementing the methods methods L2Norm, and H1sNorm, whose names already tell their function. They both take as argument a coefficient vector `Q`.

```
template <class DofHandler>
class Norms{
public:
  using calc_t = double;
  using GridView = typename DofHandler::GridView;
  enum{ world_dim = GridView::dimension };
  using MatrixType  = Eigen::SparseMatrix<calc_t,Eigen::RowMajor>;
  using Coordinate  = Dune::FieldVector<calc_t, world_dim>;

  Norms(DofHandler const& dof_handler)
    :  dofh_(dof_handler), gv_(dofh_.gv), N_(dofh_.size()) {};

  template <class Vector>
  double L2Norm(Vector const& Q);
```

```
    template <class Vector>
    double H1sNorm(Vector const& Q);

  private:
    DofHandler dofh_;
    unsigned N_;
    GridView const& gv_;
};
```

HINT: Remember that both norms are the "energy norms" induced by suitable bilinear forms. Thus the Galerkin matrices for these forms can be used to compute the (squares of the) norms.

Solution: See Listing 10.2 for the code.

Listing 10.2: Implementation for `Norms`

```
template <class DofHandler>
template <class Vector>
double Norms<DofHandler>::L2Norm(Vector const& Q) {
  std::vector<Eigen::Triplet<double>> triplets;
  NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh_);
  matAssembler(triplets, NPDE15::AnalyticalLocalMass());

  MatrixType A(dofh_.size(),dofh_.size());
  A.setFromTriplets(triplets.begin(), triplets.end());
  A.makeCompressed();

  return sqrt(Q.transpose()*A*Q);
};

template <class DofHandler>
template <class Vector>
double Norms<DofHandler>::H1sNorm(Vector const& Q) {
  std::vector<Eigen::Triplet<double>> triplets;
  NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh_);
  matAssembler(triplets, NPDE15::AnalyticalLocalLaplace());

  MatrixType A(dofh_.size(),dofh_.size());
  A.setFromTriplets(triplets.begin(), triplets.end());
  A.makeCompressed();

  return sqrt(Q.transpose()*A*Q);
};
```

(10.1g) Now we consider (10.1.1) on the unit disk $\Omega := \{\boldsymbol{x} \in \mathbb{R}^2 : \|\boldsymbol{x}\| < 1\}$. An incomplete file `main.cc` is supplied. The loading and refinement of the mesh, as well as the computation of the stiffness matrix is already implemented.

Extend this code to compute the load vector using the function you wrote in (10.1a), solve the system and obtain the L^2-norm and H^1-seminorm of the solution. Finally plot the L^2-norm and

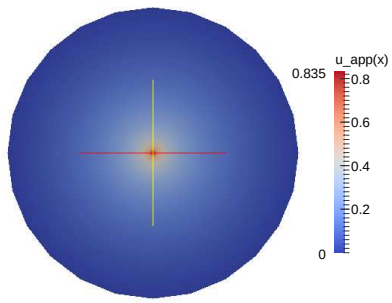
H^1-norm of the solutions vs. the number of degrees of freedom, and also output the solution for the finest grid in `Vtk`-format.

Solution: See Listing 10.3 for the code.

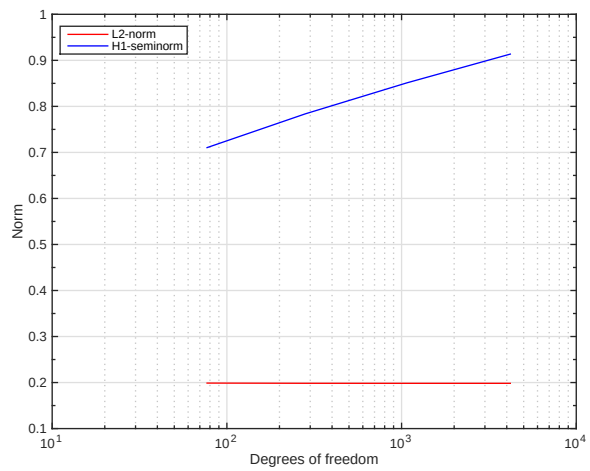
Listing 10.3: Implementation for `main.cc`

```
const int world_dim = 2;
using calc_t = double;
using Matrix = Eigen::SparseMatrix<calc_t, Eigen::RowMajor>;
using Vector = Eigen::VectorXd;
using IndexVector = std::vector<bool>;
using GridType = Dune::ALUSimplexGrid<2, 2>;
using GridView = GridType::LevelGridView;
using DofHandler  = NPDE15::DofHandler<GridView>;

int main(int argc, char *argv[]){
  try{
    // load the grid from file
    std::string fileName = "circle_128.msh";
    int refine = 3;

    // Declare and create mesh using the Gmsh file
    Dune::GridFactory<GridType> gridFactory;
    Dune::GmshReader<GridType>::read(gridFactory, fileName.c_str(),
        false, true);
    GridType *workingGrid = gridFactory.createGrid();
    workingGrid->globalRefine(refine);
    workingGrid->loadBalance();

    Vector Ndofs(4); Ndofs.setZero();
    Vector L2norm(4); L2norm.setZero();
    Vector H1Snorm(4); H1Snorm.setZero();
    for (int level=0; level<=refine; ++level){
      // Get the Gridview
      GridView gv = workingGrid->levelGridView(level);

      // Initialize dof-handler
      DofHandler dofh(gv);

      unsigned N = dofh.size();
      std::cout << "Solving for N =" << N << " unknowns.\n";

      // Get boundary nodes
      IndexVector dirichlet_dofs(N);
      NPDE15::LBoundaryDofs<DofHandler> get_bnd_dofs(dofh);
      get_bnd_dofs(dirichlet_dofs);
      dofh.set_inactive(dirichlet_dofs);

      // assemble rhs and set dirichlet dofs to dirichlet data
      Vector Phi(N); Phi.setZero();
      NPDE15::VectorAssembler<DofHandler> vecAssembler(dofh);
```

```
      vecAssembler(Phi, NPDE15::LocalDelta());
      // Homogeneous dirichlet data
      Vector G(N); G.setZero();
      vecAssembler.set_inactive(Phi, G);

      // assemble the system matrix
      std::vector<Eigen::Triplet<calc_t>> triplets;
      NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh);
      matAssembler(triplets, NPDE15::AnalyticalLocalLaplace());
      matAssembler.set_inactive(triplets);

      Matrix A(N, N);
      A.setFromTriplets(triplets.begin(), triplets.end());
      A.makeCompressed();

      // solution vector U
      Vector U(N); U.setZero();

      // solve the system
      U = Phi/A; // short-hand, see Pardiso.hpp for more information

      // get the errors
      NPDE::Norms<DofHandler> norms(dofh);
      Ndofs[level]  = dofh.size();
      L2norm[level]  = norms.L2Norm(U);
      H1Snorm[level] = norms.H1sNorm(U);

      if(level==refine){
     //Write finest grid solution to vtk file
     Dune::VTKWriter<GridView> vtkwriter(gv);
     std::stringstream outputName;
     outputName << "solution";
     vtkwriter.addVertexData(U, "u_app(x)");
     vtkwriter.write(outputName.str().c_str());
     std::cout << "Done.\n";
      }
    } // end for refinement levels
    std::ofstream outnd( "Ndofs.dat", std::ios::out |
       std::ios::binary );
    outnd << Ndofs;
    outnd.close( );
    std::ofstream outL2n( "L2norm.dat", std::ios::out |
       std::ios::binary );
    outL2n << L2norm;
    outL2n.close( );
    std::ofstream outH1sn( "H1Snorm.dat", std::ios::out |
       std::ios::binary );
    outH1sn << H1Snorm;
    outH1sn.close( );
  }
```



(a) Solution



(b) L^2-norm and H^1-seminorm versus N.

Figure 10.1: Plots for subproblem (10.1c).

(10.1h) Consider an abstract linear variational problem

$$u \in V_0 : \quad \mathsf{a}(u,v) = \ell(v) \quad \forall v \in V_0, \tag{10.1.2}$$

with s.p.d. ($\rightarrow$ Def. [NPDE, Def. 2.2.35]) bilinear form a and a linear form ℓ that is continuous in the energy norm in the sense of [NPDE, Eq. (2.2.48)].

Elaborate the relationship between the energy norm of the solution of (10.1.2) and the minimal value of the associated quadratic functional according to [NPDE, § 2.4.2].

Solution: From the variational problem with $u = v$ we have $\ell(u) = \mathsf{a}(u,u)$. Thus

$$J(u) = \frac{1}{2}\mathsf{a}(u,u) - \ell(u) = -\frac{1}{2}\|u\|_{\mathsf{a}}^2.$$

(10.1i) Explain your observations on the behavior of the H^1-norm in subproblem (10.1g) in light of the theoretical conclusions obtained in subproblems (10.1d) and (10.1g).

Solution: We can see from Figure 10.1 that the H^1-seminorm of the solution appears to diverge (albeit not very quickly), while the L^2-norm converges. This is because the solution of this variational problem is not in H^1 (but it is in L^2), *cf.* subproblem (10.1d).

Problem 10.2 Debugging Finite Element Codes (Core problem)

[NPDE, Chapter 5] confronted you with theoretical results on the asymptotic convergence of finite element Galerkin solutions for 2nd-order elliptic boundary value problems. On the one hand, these estimates can be used to gauge the relative efficiency of different finite element approximations, as was discussed in [NPDE, Section 5.3.5]. On the other hand, expected rates of convergence are a fine probe for detecting errors in a finite element code. For instance, the observed convergence to a known analytic solution should match the theoretical predictions, unless the code is flawed. Another way of using the approximation results of [NPDE, Section 5.3.5] to identify a faulty finite element implementation is demonstrated in this problem.

In detail these considerations are presented in [NPDE, Section 5.8]. This problem complements this section of the lecture material, in particular [NPDE, § 5.8.9]. Study this paragraph again before you continue.

The required files for this problem are available in the lecture `svn` repository

`assignments_codes/assignment10/Problem2`

Three different local assemblers

$$\mathsf{LocalLaplaceQFEX}, \quad \mathtt{X} \in \{1,2,3\}$$

purport to provide the Galerkin matrix and right-hand side vector for the finite element discretization of the variational problem

$$u \in H^1(\Omega): \quad \mathsf{a}(u,v) := \int_\Omega \operatorname{grad} u \cdot \operatorname{grad} v \,\mathrm{d}\boldsymbol{x} = \ell(v) := \int_\Omega f(\boldsymbol{x}) v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} \quad \forall v \in H^1(\Omega) \tag{10.2.1}$$

using *quadratic* Lagrangian finite elements (space $\mathcal{S}_2^0(\mathcal{M})$) on a triagular mesh $\mathcal{M}$ of some polygon $\Omega \subset \mathbb{R}^2$. The local assemblers provide the same basic methods as the `LocalAssembler` object in [NPDE, Code 3.6.49]. The implied ordering of local shape functions is the same as in [NPDE, Ex. 3.6.41], see also [NPDE, Code 3.6.42].

(10.2a) Complete the class

```
template <class DofHandler>
  class InterpolateQFE{
  public:
    using calc_t = double;
    using GridView = typename DofHandler::GridView;
    enum { K=2 };
    enum { world_dim = GridView::dimension };

    InterpolateQFE(DofHandler const& dof_handler) :
       dofh_(dof_handler), gv_(dofh_.gv) {};

    template <class Vector, class Function>
    void operator()(Vector &Phi, Function const& f) const;

  private:
    DofHandler const& dofh_;
    GridView const& gv_;
  };
```

by writintg `operator()` that accepts a function f in procedural form, and fills Phi with the basis coefficients of the nodal interpolant $\mathsf{I}_2 u \in \mathcal{S}_2^0(\mathcal{M})$. This particular piecewise quadratic interpolation is presented in [NPDE, Ex. 3.5.3].

Solution: See Listing 10.4 for the code.

Listing 10.4: Implementation for `InterpolateQFE`

```
    template <class Vector, class Function>
    void operator()(Vector &Phi, Function const& f) const{
      // loop over cells
      for (auto eit=gv_.template begin<0>(); eit!=gv_.template end<0>(); ++eit){
     Vector LocalPhi;
     Dune::PkLocalFiniteElement<calc_t,calc_t, world_dim, 2> localFE;
     assert(localFE.type()==eit->type());
     elem_geom_t const& egeom = eit->geometry();
     unsigned M=localFE.localBasis().size();
     for (unsigned i=0;i<M;++i){
       unsigned loctoglob = dofh_(*eit, i);
       auto pos = this->getLocalPos(i, egeom);
       Phi[loctoglob] = f(pos);
     }
      }
    }

    Coordinate getLocalPos( int local_dof, elem_geom_t const & egeom ) const {
      Coordinate pos;
      switch (local_dof){ // get position of dof according to DUNE convention
      case 0: pos = egeom.corner(0); break;
      case 1: pos = (egeom.corner(0) + egeom.corner(1)); pos *= 0.5; break;
      case 2: pos = egeom.corner(1); break;
      case 3: pos = (egeom.corner(0) + egeom.corner(2)); pos *= 0.5; break;
      case 4: pos = (egeom.corner(1) + egeom.corner(2)); pos *= 0.5; break;
      case 5: pos = egeom.corner(2); break;
      }
      return pos;
    }
```

(10.2b) Determine a sharp bound $T(h_{\mathcal{M}})$ in the estimate

$$|\mathsf{a}(u,u) - \mathsf{a}(\mathsf{I}_2 u, \mathsf{I}_2 u)| \leq CT(h_{\mathcal{M}}), \tag{10.2.2}$$

where $u : \overline{\Omega} \mapsto \mathbb{R}$ is supposed to be smooth and the unknown constant $C > 0$ may depend only on Ω and the shape regularity measure of $\mathcal{M}$.

Use the following result, which is a generalization of [NPDE, Cor. 5.3.43] to quadratic Lagrangian finite element spaces.

Theorem. *Let $\Omega \subset \mathbb{R}^d$, $d = 1, 2, 3$, be a bounded polygonal/polyhedral domain equipped with a simplicial mesh $\mathcal{M}$. Then the following interpolation error estimate holds for the nodal interpo-*

lation operator I_2 *onto* $\mathcal{S}_2^0(\mathcal{M})$

$$\|u - \mathsf{I}_2 u\|_{H^1(\Omega)} \leq C h^{\min\{3,k\}-1} |u|_{H^k(\Omega)} \quad \forall u \in H^k(\Omega), \quad k = 2, 3\,,$$

with a constant $C > 0$ *depending only on* k *and the shape regularity measure* $\rho_{\mathcal{M}}$.

HINT: Recall from [NPDE, Section 5.1] the arguments that suggested that the energy norm provides a relevant norm for measuring the discretization error.

Solution: We have

$$\begin{aligned}
|\mathsf{a}(u,u) - \mathsf{a}(\mathsf{I}_2 u, \mathsf{I}_2 u)| &= |\mathsf{a}(u + \mathsf{I}_2 u, u - \mathsf{I}_2 u)| \\
&\leq \|u + \mathsf{I}_2 u\|_A \|u - \mathsf{I}_2 u\|_A \\
&\leq (\|u\|_A + \|\mathsf{I}_2 u\|_A)\|u - \mathsf{I}_2 u\|_A \\
&\leq (2\|u\|_A + \|u - \mathsf{I}_2 u\|_A)\|u - \mathsf{I}_2 u\|_A \\
&\leq C_1 |u|_{H^1(\Omega)} |u|_{H^3(\Omega)} h^2 + C_2 |u|^2_{H^3(\Omega)} h^4 \\
&\leq C_3 |u|_{H^1(\Omega)} |u|_{H^3(\Omega)} h^2 .
\end{aligned}$$

(10.2c) Write a method

```
template <class LocalAssembler>
double test_assembleQFE(DofHandler const & dofh,
                        LocalAssembler lassemblr)
```

that returns $\mathsf{a}(\mathsf{I}_2 u, \mathsf{I}_2 u)$ for $u(\boldsymbol{x}) = \exp(\|\boldsymbol{x}\|^2)$ and the domain triangulated by the mesh described by the `Dune::GridView` and the finite element space managed by DofHandler dofh. The argument lassemblr passes a local assembler for quadratic Lagrangian finite element with the calling syntax of LocalLaplaceQFEX introduced above.

HINT: Use InterpolateQFE developed in (10.2a).

Solution: See Listing 10.5 for the code.

Listing 10.5: Implementation for `test_assembleQFE`

```
template<class LocalAssembler>
double test_assembleQFE( DofHandler const & dofh, LocalAssembler
    const & lassemblr){
  unsigned N = dofh.size();
  // solution
  auto u_ex=[](Coordinate const& x){ return
      exp(x[0]*x[0]+x[1]*x[1]); };
  // interpolate function
  Vector U_int(N); U_int.setZero();
  NPDE15::InterpolateQFE<DofHandler> interpolator(dofh);
  interpolator(U_int, u_ex);
  // get stiffness matrix for bilinear form
  std::vector<Eigen::Triplet<double>> triplets;
  NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh);
  matAssembler(triplets, lassemblr);
  Matrix A(N,N);
  A.setFromTriplets(triplets.begin(), triplets.end());
```

```
  A.makeCompressed();
  // computes a(Iu, Iu)
  return U_int.transpose()*A*U_int;
}
```

(10.2d) Complete the file `main.cc` so for each refinement level and for each of the local assemblers LocalLaplaceQFEX, X $\in \{1, 2, 3\}$, it computes $errX = |\mathsf{a}(u, u) - \mathsf{a}(\mathsf{I}_2 u, \mathsf{I}_2 u)|$ for the function $u(\boldsymbol{x}) = \exp(\|\boldsymbol{x}\|^2)$ from (10.2c). Finally plot $errX$ against the number of elements in a suitable scale.

HINT: Use the method test_assembleQFE implemented in (10.2c).

Also, you may use $|u|^2_{H^1(\Omega)} = 23.7608$ for $\Omega = (0, 1)^2$.

Solution: See Listing 10.6 for the code.

Listing 10.6: Implementation for `main`

```
int main(int argc, char *argv[]){
  try{
    // load the grid from file
    std::string fileName = "square_64.msh";
    int refine = 3;

    // Declare and create mesh using the Gmsh file
    Dune::GridFactory<GridType> gridFactory;
    Dune::GmshReader<GridType>::read(gridFactory, fileName.c_str(),
        false, true);
    GridType *workingGrid = gridFactory.createGrid();
    workingGrid->globalRefine(refine);
    workingGrid->loadBalance();

    Eigen::Matrix4d H1SNerr; H1SNerr.setZero();
    Vector Ndofs(4); Ndofs.setZero();
    for (int lev=0; lev<=refine; ++lev){
      // Get the Gridview
      GridView gv = workingGrid->levelGridView(lev);

      // Initialize dof-handler
      DofHandler dofh(gv);
      std::cout << "level : " << lev << " N : " << dofh.size() <<
          std::endl;
      Ndofs[lev]      = dofh.size();
      H1SNerr(lev,0) = fabs(23.76088 - test_assembleQFE(dofh,
                                NPDE15::LocalLaplaceQFE1()) );
      H1SNerr(lev,1) = fabs(23.76088 - test_assembleQFE(dofh,
                                NPDE15::LocalLaplaceQFE2()) );
      H1SNerr(lev,2) = fabs(23.76088 - test_assembleQFE(dofh,
                                NPDE15::LocalLaplaceQFE3()) );
    }
    std::ofstream outnd("Ndofs.dat", std::ios::out |
        std::ios::binary );
```

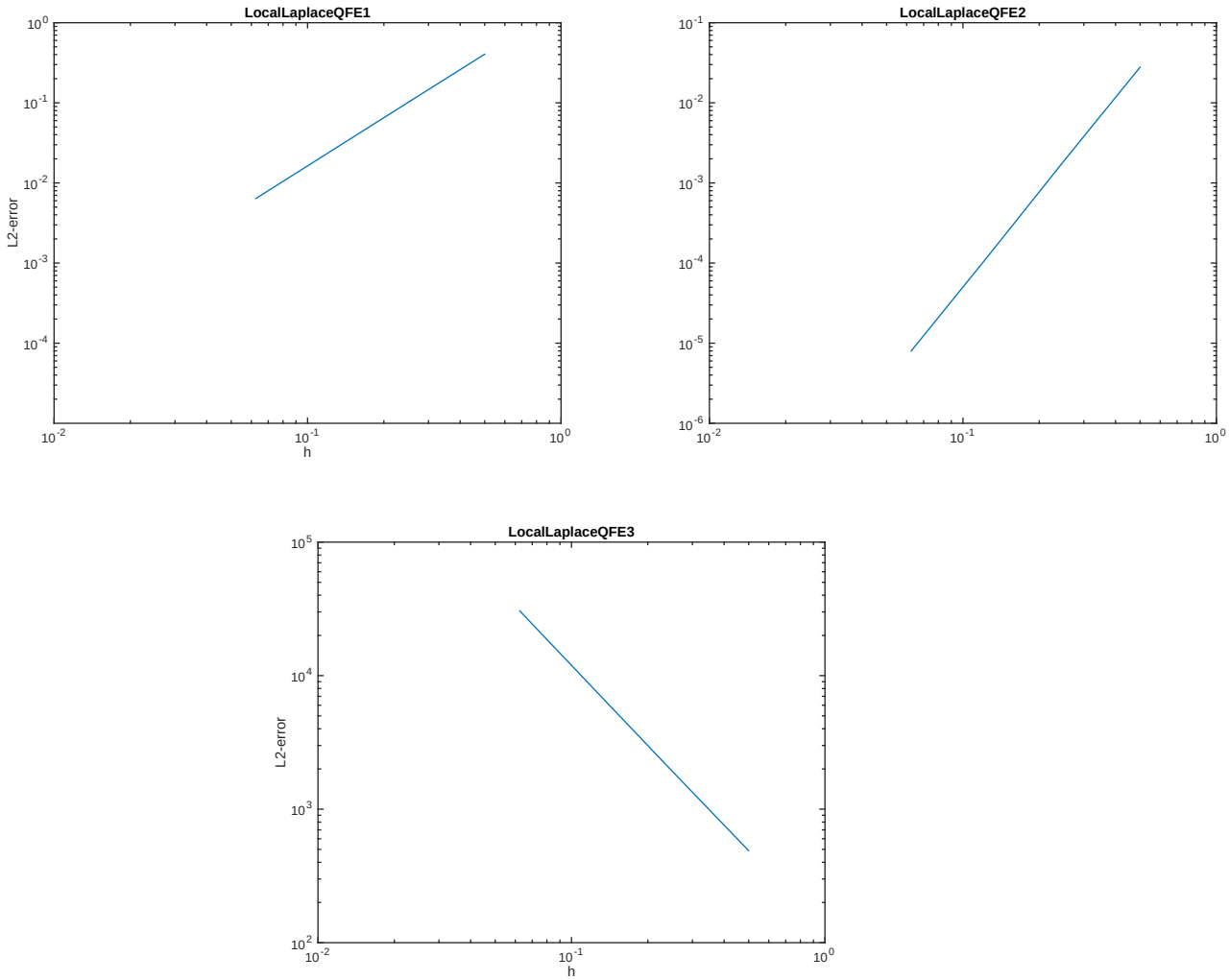


Figure 10.2: Convergence plots for subproblem (10.2e).

```
    outnd << Ndofs;
    outnd.close( );
    std::ofstream outH1se( "H1sError.dat", std::ios::out |
        std::ios::binary );
    outH1se << H1SNerr.block<4,3>(0,0);
    outH1se.close( );

  }
```

(10.2e) Which implementations of the assembly routine are wrong, which are correct? Explain your answer.

Solution: We see that only the second test converges to zero with rate 2, the third test does not converge at all, and the first test converges with the wrong rate. Thus, LocalLaplaceQFE2 is correct. The others are not.

Problem 10.3 Crouzeix-Raviart Finite Elements

In this problem we come across an alien "non-conforming" piecewise linear finite element space. It has no direct relevance for scalar second-order elliptic boundary value problems, but permits us to practice notions and techniques for finite elements.

Let a triangular mesh $\mathcal{M}$ of a 2D polygonal bounded domain $\Omega \subset \mathbb{R}^2$ be given and write $\mathcal{N} = \{\boldsymbol{m}_1, \dots, \boldsymbol{m}_N\}$ for the set of the midpoints of its edges. A numbering of these points is assumed.

The so-called Crouzeix-Raviart finite element space $\mathcal{CR}(\mathcal{M}) \subset L^2(\Omega)$ on the mesh $\mathcal{M}$ is defined as the span of the functions b_N^j, $j = 1, \dots, N$, which satisfy

$$b^i_{N\,|K} \in \mathcal{P}_1(K) \quad \forall K \in \mathcal{M} \quad , \quad b^i_N(\boldsymbol{m}_j) = \begin{cases} 1 & \text{, if } i = j \ , \\ 0 & \text{else,} \end{cases} \quad i, j \in \{1, \dots, N\} \ . \tag{10.3.1}$$

(10.3a) Show that (10.3.1) provides a valid definition of the functions b_N^j.

Solution: An affine linear function $b \in \mathcal{P}_1(\mathbb{R}^2)$ is uniquely defined by specifying its values in three non-collinear points. Since the triangle K must not be degenerate, the three midpoints of its edges cannot be collinear.

(10.3b) Show that the set of functions $\{b_N^j : j = 1, \dots, N\}$ is linearly independent.

Solution: To show the definition of linear independence, namely

$$\sum_{j=1}^{N} \alpha_j b_N^j(\boldsymbol{x}) = 0 \quad \forall \boldsymbol{x} \in \Omega \quad \Rightarrow \quad \alpha_j = 0 \quad \forall j \ .$$

Choosing $\boldsymbol{x} := \boldsymbol{m}_k$, (10.3.1) implies $\alpha_k = 0$, $k = 1 \dots, N$.

(10.3c) Show that $\mathcal{CR}(\mathcal{M}) \not\subset H^1(\Omega)$.

Solution: In fact, the basis functions are discontinuous: $b_N^i(\boldsymbol{x}_n) = -1$ if $\boldsymbol{x}_n$ is opposite of $\boldsymbol{m}_i$ on some triangle, but $b_N^i = 0$ on all other triangles touching $\boldsymbol{x}_n$, thus we have "different values from different sides". Since b_N^j is piecewise smooth, the assumptions of [NPDE, Thm. 2.3.20] are violated.

(10.3d) Describe the support of a basis function b_N^i.

Solution: The support of the function b_N^i coincides with the union of the two triangles on either side of edge i.

(10.3e) We use the b_N^i from (10.3.1) as global shape functions. Show that the local shape functions for $\mathcal{CR}(\mathcal{M})$ on a triangle K can be expressed in terms of the barycentric coordinate functions λ_i on K as follows:

$$b^j_{N\,|K} = 1 - 2\lambda_{\mathrm{opp}(j)} \ , \quad j = 1 \dots, N \ , \tag{10.3.2}$$

where $\mathrm{opp}(j)$ is the local index of the vertex opposite of $\boldsymbol{m}_j$ on triangle $K \in \mathcal{M}$.

Solution: It is enough to see that the given formula agrees on all the midpoints of K (or vertices, or any other three non-collinear points). Recall sub-problem (10.3a). We have $\lambda_{\mathrm{opp}(j)}(\boldsymbol{m}_j) = 0$, so $b_N^j\,|K(\boldsymbol{m}_j) = 1$. On the other edge midpoints we have $\lambda_{\mathrm{opp}(j)}(\boldsymbol{m}_n) = \frac{1}{2}$, so $b_N^j\big|_K(\boldsymbol{m}_n) = 0$.

(10.3f) Compute the element (Galerkin) matrix for the finite element space $\mathcal{CR}(\mathcal{M})$ and the bilinear form

$$\mathsf{a}(u,v) = \int_\Omega uv \,\mathrm{d}\boldsymbol{x} \;, \quad u,v \in L^2(\Omega) \;.$$

HINT: Use the integral formula for barycentric coordinate functions on a triangle K:

$$\int_K \lambda_1^{\alpha_1}\lambda_2^{\alpha_2}\lambda_3^{\alpha_3} \,\mathrm{d}\boldsymbol{x} = |K| \frac{\alpha_1!\alpha_2!\alpha_3!2!}{(\alpha_1+\alpha_2+\alpha_3+2)!} \;.$$

Solution:

$$\begin{aligned}
(\mathsf{a}_K(1-2\lambda_i, 1-2\lambda_j))_{i,j} &= \left(\int_K 1 \,\mathrm{d}\boldsymbol{x} - 2\int_K (\lambda_i+\lambda_j)\,\mathrm{d}\boldsymbol{x} + 4\int_K \lambda_i\lambda_j \,\mathrm{d}\boldsymbol{x}\right)_{i,j} \\
&= \left(|K| - 2\cdot 2\frac{|K|}{3}\right)\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix} + \frac{|K|}{3}\begin{pmatrix}2&1&1\\1&2&1\\1&1&2\end{pmatrix} \\
&= \frac{|K|}{3}\begin{pmatrix}1&&\\&1&\\&&1\end{pmatrix}.
\end{aligned}$$

For the implementation of finite element methods based on Crouzeix-Raviart finite element spaces in Dune we use the b_N^j from (10.3.1) as global shape functions. We number them taking into account the numbering of edges in Dune, see [NPDE, § 3.6.21].

As usual, template files for the new classes you will need to write are available in the lecture `svn` repository

`assignments_codes/assignment10/Problem3`

The idea is that you extend your own code, reason why it does not contain the files you already implemented in the previous assignments. Please do not forget to include them when you submit your work. If your own implementation is still not working, you may use the classes provided in `assignments_codes/solutions`.

(10.3g) Implement the class CRDofHandler as you did for DofHandler but considering now the edges (co-dimension 1) as degrees of freedom instead of the vertices (co-dimension 2).

HINT: This means you have to change **operator**() and size()

Solution: See Listing 10.7 for the code.

Listing 10.7: Implementation for `CRDofHandler`

```
template <class GridView_t>
class CRDofHandler{
public:
    using calc_t     = double;
    using GridView   = GridView_t;
    using index_t    = typename GridView::IndexSet::IndexType;
```

```
    using FlagVector = std::vector<bool>;
    enum { world_dim = GridView::dimension };
    enum { K=1 };

    CRDofHandler(GridView const& gridview) : gv(gridview), set(gv.indexSet()) {
        inactive_dofs = FlagVector(size(), false); // all dofs are active by default
    };

    template <class Element>
    index_t operator()(Element const& e, index_t dof) const{
        index_t corners = e.geometry().corners(); // number of corners
        assert(dof<corners);
        return set.subIndex(e, dof, 1);
    }

    void set_inactive(FlagVector const& inactive_dofs){
        assert(inactive_dofs.size()==size());
        this->inactive_dofs = inactive_dofs;
    }

    bool active(index_t global_idx) const{
        return !inactive_dofs[global_idx];
    }

    template <class Element>
    size_t size_loc(Element const &e) const { return 3; }

    template <class Element, class Coordinate>
    void getDofPosition(Element const& e, index_t dof, Coordinate &x) const{
      auto egeom = e.geometry();
      switch (dof){ // get position of dof according to DUNE convention
      case 0: x = (egeom.corner(0) + egeom.corner(1)); x *= 0.5; break;
      case 1: x = (egeom.corner(0) + egeom.corner(2)); x *= 0.5; break;
      case 2: x = (egeom.corner(1) + egeom.corner(2)); x *= 0.5; break;
      }
    }

    size_t size() const{
        return set.size(1);
    }

    GridView const& gv;

```

```
private:
    typename GridView::IndexSet const& set;
    FlagVector inactive_dofs;

};
```

(10.3h) We introduce the mesh-dependent bilinear form

$$\mathsf{a}_{\mathcal{M}}(u,v) := \sum_{K\in\mathcal{M}} \int_K \mathbf{grad}\, u \cdot \mathbf{grad}\, v \,\mathrm{d}\boldsymbol{x} \,. \tag{10.3.3}$$

Derive a formula for the entries of the element matrices for the Galerkin discretization of $\mathsf{a}_{\mathcal{M}}$ based on $\mathcal{CR}(\mathcal{M})$. The entries of the element matrix for triangle K should be expressed in terms of the angles ω_k, $k = 1,2,3$ of K, see [NPDE, Fig. 85]. Follow the convention that the i-th local edge is opposite to the i-th local vertex, $i = 1,2,3$.

HINT: Use (10.3.2) and [NPDE, Eq. (3.3.21)].

Solution: We see that $\mathbf{grad}(1-2\lambda_j) = -2\,\mathbf{grad}\,\lambda_j$, so the matrix entries should be simply four times those of the regular LFE matrix. The numbering is the same, as well.

(10.3i) Implement the class CRLocalLaplace which provides a method

```
template <class Element, class Matrix>
void operator()(Element const& e, Matrix &local) const;
```

that computes the element matrix for $\mathsf{a}_{\mathcal{M}}$ using the Crouzeix-Raviart finite element space.

HINT: The local numbering scheme of the previous sub-problem does not apply, use the Dune intrinsic numbering convention ([NPDE, Rem. 3.6.24]).

HINT: Recall the result of the subproblem (10.3h) and use AnalyticalLocalLaplace from subproblem (7.4b) that computes the element matrices for $\mathsf{a}_{\mathcal{M}}$ for the piecewise linear Lagrangian finite element spaces.

Solution: See Listing 10.8 for the code. As we saw in subproblem (10.3h), it is 4 times the element matrix in AnalyticalLocalLaplace with the indices 0 and 2 permuted.

Listing 10.8: Implementation for `CRLocalLaplace`

```
    template <class Element>
    void operator()(Element const& e, ElementMatrix &local) const{
      local=0;
      // get element's geometry and area
      auto const& egeom = e.geometry();
      auto elem_area = egeom.volume();
      // compute gradients using analytical formula (rotated)
      // Omit factor 1/(2*area) as it will be computed at the end
      Dune::FieldMatrix<calc_t,3,2> grad;
      grad[2] = (egeom.corner(1)- egeom.corner(2)); //edge 2 -
          vertix 0
      grad[1] = (egeom.corner(2)- egeom.corner(0)); //edge 1 -
          vertix 1
      grad[0] = (egeom.corner(0)- egeom.corner(1)); //edge 0 -
          vettix 2
```

```
      // Finally compute the matrix (4 times usual Laplace element
         matrix)
      for (unsigned i=0;i<local.N();++i){
     for (unsigned j=0;j<local.M();++j){
       local[i][j]= grad[i]*grad[j]/(elem_area);
     }
      }
```

(10.3j) Prove that the Galerkin matrices for $\mathsf{a}_{\mathcal{M}}$ and the finite element spaces $\mathcal{CR}(\mathcal{M})$ are always positive semidefinite.

Solution: This follows directly from the positive semidefiniteness of $\mathsf{a}_{\mathcal{M}}$. Indeed, let $u \in \mathcal{CR}(\mathcal{M})$, and let $\vec{\mu}$ be its coefficient vector. Then,

$$\vec{\mu}^\top \mathbf{A}\vec{\mu} = \mathsf{a}_{\mathcal{M}}(u_N, u_N) = \sum_K \int_K \|\mathbf{grad}\, u_N\|^2 \,\mathrm{d}\boldsymbol{x} \geq 0. \tag{10.3.4}$$

(10.3k) Assume that Ω is connected. Show that the kernel of the Galerkin matrix arising from the discretization of $\mathsf{a}_{\mathcal{M}}$ based on the basis $\{b_N^j\}_{j=1}^N$ of $\mathcal{CR}(\mathcal{M})$ from (10.3.1) is one-dimensional and spanned by the vector with all entries $= 1$.

Solution: First, assume that $\mathbf{A}\vec{\mu} = 0$. Then

$$\mathsf{a}_{\mathcal{M}}(u_N, u_N) = 0 \implies \mathbf{grad}\, u_N|_K = 0 \qquad \forall K \in \mathcal{M},$$

so $u_N|_K$ is constant. Since two neighboring meshes share a node, that means u_N must be constant everywhere, since Ω is connected.

For the converse, let $\vec{\mu}$ be the all-one vector, and note that

$$(\mathbf{A}\vec{\mu})_i = \sum_j \mathsf{a}_{\mathcal{M}}(b_N^i, b_N^j) = \mathsf{a}_{\mathcal{M}}\left(\sum_j b_N^j, b_N^i\right) = \mathsf{a}_{\mathcal{M}}(1, b_N^i) = 0.$$

(10.3l) Implement a method

```
template <class Function>
double L2Norm(DofHandler const& dofh, Vector const& Q, Function
    const& u)
```

in `main.cc` that takes the DofHandler dofh for a particular GridView, a coefficient vector `Q` of length N describing a function $u_N \in \mathcal{CR}(\mathcal{M})$ (in `Q`), and a Function u in procedural form, representing $u : \Omega \mapsto \mathbb{R}$ (in `u`). The return value should provide an approximation for $\|u - u_N\|_{L^2(\Omega)}$ computed by means of the local numerical quadrature offered by Dune with order 10.

HINT: Notice we want to compute the $\|u - u_N\|_{L^2(\Omega)}$, where u is a continuos function and u_N a finite element approximation function, i.e., $u_N(x) = \sum_{k=0}^M Q_j b_N^j(x)$. Therefore, in each element you need to use the local basis functions to reconstruct $u(x)_N$ and then use quadrature integrate $|u(x) - u_N(x)|^2$. You may follow the structure of the alternative solution for the H1 semi-norm in (8.2k). If you do, don't forget to consider the basis functions instead of the gradients.

Solution: See Listing 10.9 for the code.

Listing 10.9: Implementation for `L2Norm`

```
double L2Norm(DofHandler const& dofh, Vector const& Q, Function const& u) {
  GridView const& gv = dofh.gv;
  Dune::PkLocalFiniteElement<calc_t,calc_t,world_dim,1> localFE;
  typedef typename Dune::QuadratureRule<calc_t, world_dim> QuadRule_t;
  typedef typename Dune::QuadratureRules<calc_t, world_dim> QuadRules;
  const QuadRule_t & quadRule = QuadRules::rule(localFE.type(), 10);

  calc_t L2n=0.;
  for (auto eit = gv.template begin<0>(); eit!= gv.template end<0>(); ++eit){
    auto egeom = eit->geometry();
    assert(localFE.type()==eit->type());
    for (auto qr : quadRule){
      auto const& local_pos=qr.position();
      std::vector<Dune::FieldVector<calc_t,1> > shapef_vals;
      localFE.localBasis().evaluateFunction(local_pos, shapef_vals);
      double jac_det = egeom.integrationElement(local_pos);
      double valueQ = 0.0;
      for (unsigned i=0;i<shapef_vals.size();++i){
     unsigned globalidx=dofh(*eit, i);
     valueQ += Q[globalidx]*(1 - 2*shapef_vals[2-i]);
      }
      // and add weighted difference to the l2 error
      double diff = valueQ-u(egeom.global(local_pos));
      L2n += diff*diff*qr.weight()*jac_det;
    }
  }
  return sqrt(L2n);
}
```

(10.3m) Given a triangular mesh $\mathcal{M}$ of Ω and a continuous function $g : \partial\Omega \mapsto \mathbb{R}$, we consider the following discrete variational problem: seek $u_N \in \mathcal{CR}(\mathcal{M})$ such that

$$u_N(\boldsymbol{m}) = g(\boldsymbol{m}) \quad \forall \boldsymbol{m} \in \mathcal{N}_\partial \quad , \quad \mathsf{a}_{\mathcal{M}}(u_N, v_N) = 0 \quad \forall v_N \in \mathcal{CR}_0(\mathcal{M}) \ . \tag{10.3.5}$$

Here the following notations have been used:

- $\mathcal{N}_\partial := \{\boldsymbol{p} \in \mathcal{N} : \ \boldsymbol{p} \in \partial\Omega\}$ (midpoints of edges on the boundary) ,
- $\mathcal{CR}_0(\mathcal{M}) := \{v \in \mathcal{CR}(\mathcal{M}) : \ v(\boldsymbol{m}) = 0 \ \forall \boldsymbol{m} \in \mathcal{N}_\partial\}$.

Implement a method `void solve(DofHandler const & dof, Vector & uN)` in `main.cc` which:

- Marks the boundary Dofs and set them inactive

- Computes the (global) Galerkin matrix for the bilinear form $\mathsf{a}_{\mathcal{M}}$ discretized by means of a Crouzeix Raviart finite element space defined on a triangular mesh. The choice of global shape functions and numbering schemes as introduced above still apply.
- Assembles the right hand side vector.
- Solves the system, i.e. computes the coefficient vector (w.r.t. the basis $\left\{b_N^j\right\}_{j=1}^N$ from (10.3.1)) for the solution $u_N \in \mathcal{CR}(\mathcal{M})$ of (10.3.5). The obtained coefficient vector for u_N should have N components, where N is the number of *all* edges in the mesh $\mathcal{M}$.

HINT: You may rely on the function CRLocalLaplace from subproblem (10.3i) and CRBoundaryDofs in the repository.

Solution: See Listing 10.10 for the code.

Listing 10.10: Implementation for `solve()`

```
void solve(DofHandler dofh, Vector & uN){
  unsigned N = dofh.size();

  // Get boundary nodes
  IndexVector dirichlet_dofs(N);
  NPDE15::CRBoundaryDofs<DofHandler> get_bnd_dofs(dofh);
  get_bnd_dofs(dirichlet_dofs);
  dofh.set_inactive(dirichlet_dofs);

  Vector G(N); G.setZero();
  // loop over cells
  for (auto eit=dofh.gv.template begin<0>(); eit!=dofh.gv.template
     end<0>(); ++eit){
    auto egeom = eit->geometry();
    for (unsigned i=0;i<3;++i){
      unsigned loctoglob=dofh(*eit, i);
      if (!dofh.active(loctoglob)){
     Coordinate pos;
     dofh.getDofPosition(*eit, i, pos);
     G[loctoglob] = u_ex(pos);
      }
    }
  }

  // assemble rhs and set dirichlet dofs to dirichlet data
  Vector Phi(N); Phi.setZero();
  NPDE15::VectorAssembler<DofHandler> vecAssembler(dofh);
  vecAssembler.set_inactive(Phi, G);

  // assemble the system matrix
  std::vector<Eigen::Triplet<calc_t>> triplets;
  NPDE15::MatrixAssembler<DofHandler> matAssembler(dofh);
  matAssembler(triplets, NPDE15::CRLocalLaplace());
  matAssembler.set_inactive(triplets);
```

```
  Matrix A(N, N);
  A.setFromTriplets(triplets.begin(), triplets.end());
  A.makeCompressed();

  // solve the system
  uN = Phi/A; // short-hand, see Pardiso.hpp for more information
}
```

(10.3n) Somebody claims that the Crouzeix-Raviart finite element space and, in particular, the method solve() from sub-problem (10.3m) can be used to solve the boundary value problem

$$-\Delta u = 0 \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega\,, \tag{10.3.6}$$

though $\mathcal{CR}(\mathcal{M}) \not\subset H^1(\Omega)$! Test this claim numerically for $\Omega =]0,1[^2$ by using $u(\boldsymbol{x}) = \log(\left\|\boldsymbol{x} + \binom{1}{0}\right\|)$, which satisfies $\Delta u = 0$ on Ω, and computing $\|u - u_N\|_{L^2(\Omega)}$ for four different meshes arising from global regular refinement of an initial coarse mesh.

To that end complete the implementation of `main.cc` so it otputs a vector of the values $\|u - u_N\|_{L^2(\Omega)}$ and a vector of the number of elements for each mesh.

Solution: See Listing 10.11 for the code.

Listing 10.11: Implementation for `main`

```
int main(int argc, char *argv[]){
  try{
    std::string fileName = "../_input_meshes/square_4.msh";
    int refine = 3;

    // Declare and create mesh using the Gmsh file
    Dune::GridFactory<GridType> gridFactory;
    Dune::GmshReader<GridType>::read(gridFactory, fileName.c_str(),
        false, true);
    std::unique_ptr<GridType> workingGrid(gridFactory.createGrid());
    workingGrid->globalRefine(refine);
    workingGrid->loadBalance();

    Vector L2error(4); L2error.setZero();
    Vector Ndofs(4); Ndofs.setZero();
    for (int level=0; level<=refine; ++level){
      // Get the Gridview
      GridView gv = workingGrid->levelGridView(level);

      // Initialize dof-handler
      DofHandler dofh(gv);

      // solution vector U
      Vector uN(dofh.size()); uN.setZero();
      solve(dofh, uN);
      L2error[level]  = L2Norm(dofh, uN, u_ex);
      Ndofs[level]  = dofh.size();
```

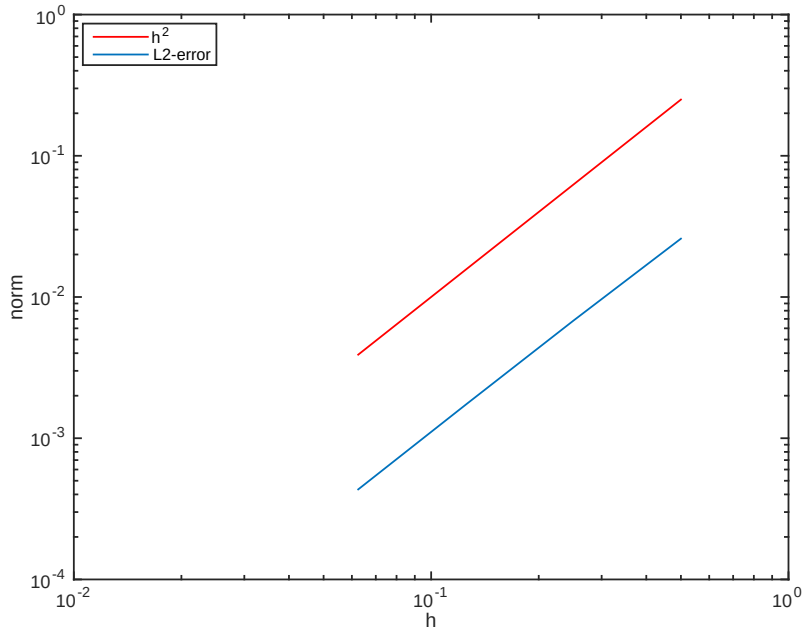


Figure 10.3: Plots for subproblem (10.3o).

```
    }

    std::ofstream outnd( "Ndofs.dat", std::ios::out |
        std::ios::binary );
    outnd << Ndofs;
    outnd.close( );
    std::ofstream outL2e( "L2Error.dat", std::ios::out |
        std::ios::binary );
    outL2e << L2error;
    outL2e.close( );
    std::cout << L2error << std::endl;

  }
```

(10.3o) Based on the numerical results of the previous sub-problem, describe qualitatively and quantitatively the observed convergence of $\|u - u_N\|_{L^2(\Omega)}$ as the mesh-width tends to 0.

HINT: The return values you should get can be loaded from the MATLAB data file `l2err_CR.dat`.

Solution: We find algebraic convergence with approximate order 2, so we have good reason to believe the mysterious claimant from subproblem (10.3n).

Published on 29.04.2015.

To be submitted on 06.05.2015.

References

[NPDE] Lecture Slides for the course "Numerical Methods for Partial Differential Equations".SVN revision # 75873.

[NCSE] Lecture Slides for the course "Numerical Methods for CSE".

Last modified on May 7, 2015