Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

deal2lkit: A toolkit library for high performance programming in deal.II

Alberto Sartori [*], Nicola Giuliani, Mauro Bardelloni, Luca Heltai

SISSA — International School for Advanced Studies, Via Bonomea 265, 34136 Trieste, Italy

A R T I C L E   I N F O

Article history:
Received 17 August 2016
Received in revised form 30 August 2018
Accepted 18 September 2018

Keywords:
Object-orientation
Software design
Finite element methods
C++

A B S T R A C T

We propose a software design for the efficient and flexible handling of the building blocks used in high performance finite element simulations, through the pervasive use of parameters (parsed through parameter files). In the proposed design, all the building blocks of a high performance finite element program are built according to the command and composite design patterns.

We present version 1.1.0 of the deal2lkit (deal.II ToolKit) library, which is a collection of modules and classes aimed at providing high level interfaces to several deal.II classes and functions, obeying the command and composite design patterns, and controlled via parameter files.

Code metadata

Current code version	1.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-16-00065
Legal Code License	LGPL version 2.1
Code versioning system used	git
Software code languages, tools, and services used	C++, MPI, TBB, Travis-CI
Compilation requirements, operating environments & dependencies	C++11, Unix-like, deal.II
If available Link to developer documentation/manual	http://mathlab.github.io/deal2lkit/
Support email for questions	luca.heltai@sissa.it, alberto.sartori@sissa.it

Acronyms

ASSIMP	ASSet IMPort library
BEM	Boundary Element Method
BICSTAB	Bi-Conjugate STABilized gradient
CG	Conjugate Gradient
deal.II	Differential Equations Analysis Library
FGMRES	Flexible GMRES
GMRES	Generalized Minimal RESidual method
GUI	Graphical User Interface
IDA	Implicit Differential Algebraic
IMEX	IMplicit EXplicit stepper
MINRES	MINimal RESidual method
MPI	Message Passing Interface
PDE	Partial Differential Equation
PDEs	Partial Differential Equations
PETSc	Portable Extensible Toolkit for Scientific Computation
QMRS	Quasi Minimal Residual Method
SUNDIALS	SUite of Nonlinear and DIfferential/ALgebraic equation Solvers
TBB	Intel Threading Building Blocks

[*] Corresponding author.
E-mail address: alberto.sartori@sissa.it (A. Sartori).

https://doi.org/10.1016/j.softx.2018.09.004

1. Motivation and significance

The solution of Partial Differential Equations (PDEs) by means of the finite element method always requires at least the following steps:

- generation of a geometrical grid to represent the domain of the simulation;
- definition of the discrete functional space(s) for the solution;
- assembly of the (often non-linear) variational formulation;
- application of proper boundary conditions;
- actual solution of the algebraic problem (often iterated in non-linear iteration steps);
- post-processing of the result (data output and error analysis).

Many commercial and open source solutions exist that offer graphical user interfaces (sometimes driven by parameter files), that address all of the above steps in a conveniently packaged, easy to use, and efficient user interface. In many cases, however, the use of a commercial software, or the use of a pre-packaged solution, for the solution of custom PDEs, lacks the required flexibility, and researcher need to turn to more low level solutions, like finite element libraries, that offer building blocks for the construction of PDE solvers, (like, for example, the deal.II library [1]).

Such a solution usually implies that different programs used to solve different PDEs share a considerable amount of code that must be written by the user. Often the solution of the same PDE with different boundary conditions, material parameters, or simply the output of different post-processing data, requires the user to partially rewrite his or her code, and to go through the process of rewriting/debugging/recompiling the end program again.

In this work we present a solution that is intermediate between a purely low level solution, where the user combines low level building blocks of a finite element library into a custom program (as done, for examples, in the 50+ example programs of the deal.II library [1]), and a fully-packaged solution, where the user acts on an (externally defined) Graphical User Interface (GUI).

The solution we propose is to gather as separate high level modules the above steps, all behaving in a uniform way with respect to how they are initialized (i.e., by parameter files), so that the writing of high performance custom PDE solvers can be simplified and made more flexible at the same time.

An easy and efficient handling of the above steps by means of parameter files has several advantages from the user perspective. It guarantees in fact both fast exploration of scenarios without the need to modify and eventually recompile the source code, and it prevents him/her from the insertion of new bugs.

We propose a software design for the handling of parameter files suitable for high performance scientific computing, which is flexible, meaning that the library or software can be easily enriched with new features, and generates self-documented applications. We implemented such design in the deal2lkit library easing the prototyping of new high performance applications, with particular emphasis on finite element methods.

Several scientific libraries based on deal2lkit have already been released: a fluid structure interaction Boundary Element Method (BEM) solver aimed at simulating ship wave interaction WaveBEM [2], a parallel BEM solver π-BEM [3], and a parallel multi-physics solver π-DoMUS [4],

Using the same design principle, the library provides interfaces to some of the most efficient high performance libraries such as Differential Equations Analysis Library (deal.II) [1] for the resolution of PDEs, SUite of Nonlinear and DIfferential/ALgebraic equation Solvers (SUNDIALS) [5] (with interfaces to Implicit Differential Algebraic (IDA) [6], KINSOL, and ARKode) for the resolution of non linear differential and algebraic equations, odes, and general nonlinear problems, and ASSet IMPort library (ASSIMP) [7] to import complex geometries coming from 3D model formats.

2. Software description

The main design principle behind deal2lkit is that each building block (i.e., almost each class that is necessary to completely define a finite element solver) should be managed through a *parameter file*. In general, a *parameter file* is used to steer the execution of a program at run time, without the need to recompile the executable, with clear advantages in terms of *human-time*, and lowered chance to introduce bugs. The efficient handling of parameter files is at the heart of deal2lkit.

Generation, validation, and parsing of parameter files are encapsulated in a class (ParameterAcceptor) which obey to the Command design pattern [8], requiring the user to interact only with ParameterAcceptor.

A class that has its own parameters to be managed through parameter files may inherit from ParameterAcceptor and exploit a simple and minimal interface, that guarantees the correct handling of parameters, with strong type checking, and automatic parsing.

In the sequel, we refer to this kind of classes as *parametrized* classes, and their instantiations to *parametrized* objects.

ParameterAcceptor implements also the Composite design pattern [8] to compose parametrized objects into tree structures that represent part-whole hierarchies. It is thus possible to nest parametrized classes and objects preserving the hierarchies of the parameters. In this way, for example, a class ParsedBoundaryConditions can have a ParsedFunction as a member object, and the structure of the parameter file will reflect such hierarchy.

The supported formats for parameter files are the standard JSON and XML plus a custom text format (with conventional extension .prm) which resembles bash files with support for sections and subsections. For the XML format a graphical user interface can be used as well.[1]

At the time of this publication, the efficient implementation of our chosen design strategy consists of 79,062 source code lines which have been developed by 4 main developers and 6 other contributors over 863 different commits. We use continuous integration (with 164 unit tests) via TravisCI to ensure that every feature and capability is maintained.

2.1. Software architecture

deal2lkit features a collection of classes aimed at handling each of the steps mentioned in Section 1, that are required to solve a finite element problems defined on a domain of topological dimension dim immersed in a Euclidean space of dimension spacedim. All these classes are derived from ParameterAcceptor as depicted in Fig. 1. In the following we provide a brief overview of some of the utilities that the deal2lkit library provides. The template parameters of the classes have been omitted for the sake of brevity.

[1] https://github.com/dealii/parameter_gui

2.2. Parsed grid generator

The interface for generating a grid through a parameter file is managed by the ParsedGridGenerator class. We can specify either the geometry of the grid that must be generated (e.g., rectangle, sphere, ball, etc.) or read it from a file. In the first case, we exploit the functions of the deal.II library, effectively providing a parametrized wrapper around it. Otherwise, we simply specify the file to be read, which must be in any of the format recognized by the deal.II library. It is worth mentioning that through the parameter file we can also specify the manifold that describes the geometry. Manifolds are used in the context of adaptive mesh refinement, to ensure that new nodes are placed conforming to the actual geometry [9].

ParsedGridGenerator allow for both serial and distributed meshes (for the latter, the p4est library is required).

2.3. Parsed finite element

The existence and stability of solutions to Partial Differential Equation (PDE) is strictly dependent on the selected solution space, which defines the finite element to use. The class ParsedFiniteElement allows the definition of the finite element type from a parameter file.

2.4. Parsed functions and boundary conditions

Forcing terms and boundary conditions are often expressed by means of functions, and one would like to have the possibility to change them without recompiling the user code. The deal2lkit library offers three classes to define such functions at run time as parsed objects:

- ParsedFunction;
- ParsedMappedFunctions;
- ParsedDiricheltBCs.

2.5. Parsed solver

The class ParsedSolver derived from both ParameterAcceptor and from the LinearOperator class [10] of deal.II, which allows the use of natural syntax for complex (serial or parallel) linear algebra objects, making the solution of the system computable with the very simple expression:

C++ code

```cpp
solution = matrix_inv * rhs;
```

The supported solver types are those provided by the deal.II library, namely, Conjugate Gradient (CG), Bi-Conjugate STABilized gradient (BICSTAB), Generalized Minimal RESidual method (GMRES), Flexible GMRES (FGMRES), MINimal RESidual method (MINRES), Quasi Minimal Residual Method (QMRS) and Richardson. One of the main advantage of such a solver is the possibility to combine it using the expression syntax of LinearOperators in the construction, for example, of block preconditioners for complex problems.

2.6. Post-processing and error analysis

The post processing is usually accomplished relying on the ParsedDataOut class (aimed at saving output to files) and ErrorHandler class to produce convergence tables.

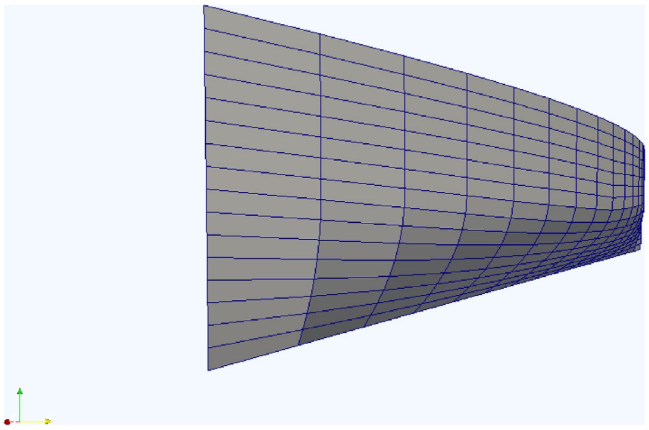
Fig. 2. Half hull of a boat imported from a CAD file. The mesh is automatically refined *on the geometry* of the hull.

The class ParsedDataOut provides a wrapper for the data out class of deal.II, to write solution vectors in any of the formats supported by deal.II, automatically splitting the output of several files at once when the code is run in parallel. If an exact (or a reference) solution is known, the class ErrorHandler gives the possibility to calculate the error of the numerical solution in various norms (i.e., L^2, H^1, L^∞, W_1^∞) where both the norms and the exact solution can be parsed from a parameter file.

2.7. Advanced solvers

deal2lkit features several interfaces for the SUNDIALS [5] libraries. In particular, deal2lkit fosters IDAInterface, which is an interface to the IDA [6] and KINSOL solvers, provided within the SUNDIALS library. Moreover, deal2lkit provides a custom IMplicit EXplicit stepper (IMEX) solver.

The advanced solvers interfaces exploit many C++11 features in order to provide more flexibility to the user, by basing the interface of the solvers over on public std::functions members, typically implemented as *lambda functions* by the users.

2.8. Importing complex geometries

Most industrial applications use computational meshes which are discretizations of CAD geometries (e.g. Fig. 2). deal2lkit offers an interface to ASSIMP, which is used to extend the compatibility of the deal.II library towards grid generation software and 3D CAD manipulation tools. The namespace AssimpInterface is used to convert an ASSIMP compatible file into a deal.II grid.

2.9. Parallel computations

deal2lkit supports massively parallel applications (as deal.II), using the Message Passing Interface (MPI), and all the solver are template on the vector type. In this way it easy to switch, for example, between Portable Extensible Toolkit for Scientific Computation (PETSc) and Trilinos vectors. deal2lkit also allows for shared memory parallelism through Intel Threading Building Blocks (TBB) [11].

3. Illustrative examples

deal2lkit is shipped with several examples featuring the major functions of the library. To highlights the possible benefits of the

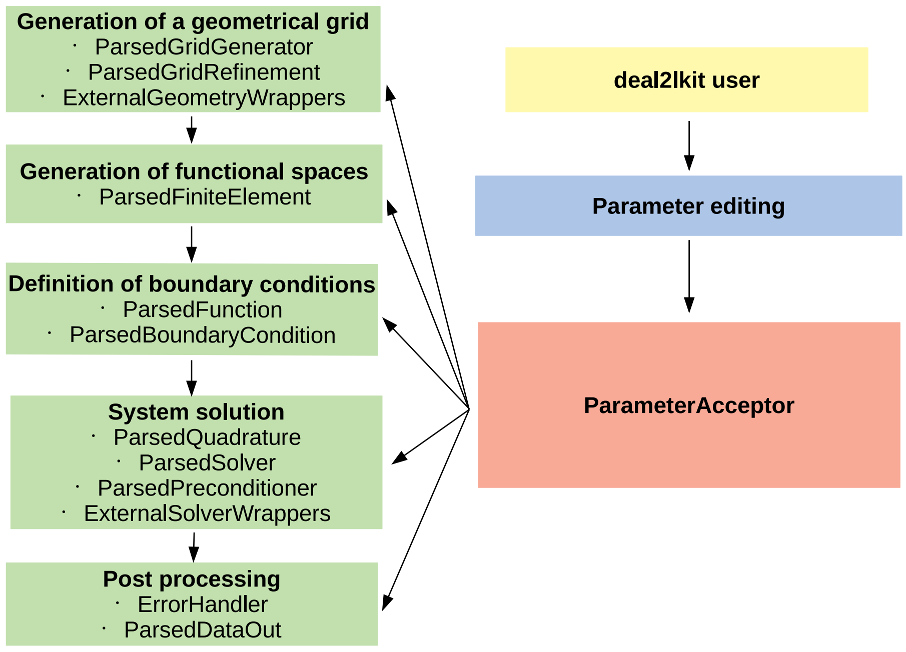

Fig. 1. Layer diagram depicting the structure of deal2lkit.

presented library, we rewrote the step-7 example of the deal.II library.[2] The deal2lkit-version of the code can be found in Appendix. It is worth mentioning that from 733 lines of code we have only 354. Moreover, the new version is much more flexible because forcing term, right-hand side, finite element, triangulation, grid refinement, convergence tables are no more hard-coded and can be easily managed through the automatically generated parameter file.

4. Impact

deal2lkit is developed on top of the deal.II library and it supports and enhances all features of deal.II [1,12], such as massively parallel simulations [13], hp adaptivity [14], geometric multi-grid [15], discontinuous finite elements [16], matrix free simulations [17], and support for linear operator [10]. It is worth mentioning that more than 150 papers have been published using deal.II in the last two years, and its developers have been awarded the J.H. Wilkinson Prize for Numerical Software in 2007. Moreover deal.II is part of the computing industry standard SPEC CPU2006 and SPEC CPU2017 benchmarks. A higher level library could enlarge the basin of users of the library itself, filling the gap between high performance computing and easiness of use.

This goal is achieved handling efficiently complex parametrized objects through the class ParameterAcceptor. We combine different design patterns in order to access most of the repetitive tasks related to writing complex scientific (e.g. finite element) codes using parameter files. The object oriented programming allows the user to easily exploit, and even extend, the capabilities of deal2lkit. This approach greatly reduces the line of codes for a new software limiting also the maintenance needed for such new application. Concurrently, the user can focus mainly on the mathematical formulation of each different problem without caring about many implementation details of the most common steps shared by a finite element calculation.

5. Conclusions

In this paper we presented version 1.1.0 of the deal2lkit library [18], which is a collection of modules for deal.II designed to provide a *high performance programming* experience to both beginner and advanced users of the deal.II library. One of the key feature of deal2lkit is the possibility to access most of the repetitive tasks related to writing complex scientific codes using parameter files.

deal2lkit is in continuous development and new functionalities are constantly implemented to enrich those tools that are useful to develop prototype finite element codes efficiently, in a well tested environment.

It is worth mentioning that the proposed design might be adopted also in other fields where an easy and flexible handling of parameter files is needed.

Acknowledgments

This work was partially supported by the project OpenViewSHIP, “Sviluppo di un ecosistema computazionale per la progettazione idrodinamica del sistema elica-carena”, supported by Regione FVG - PAR FSC 2007–2013, Fondo per lo Sviluppo e la Coesione and by the project “TRIM - Tecnologia e Ricerca Industriale per la Mobilità Marina”, CTN01-00176-163601, supported by MIUR, the Italian Ministry of Instruction, University and Research.

[2] The original code without comments can be found at https://dealii.org/8.4.0/doxygen/deal.II/step_7.html#PlainProg.

Appendix. Revised step-7 code

Here below we report how we would rewrite the step-7 example of the deal.II library exploiting the features of deal2lkit.

```
#include <deal.II/base/convergence_table.h>
#include <deal.II/base/function.h>
#include <deal.II/base/logstream.h>
#include <deal.II/base/mpi.h>
#include <deal.II/base/quadrature_lib.h>
#include <deal.II/base/smartpointer.h>

#include <deal.II/dofs/dof_accessor.h>
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/dofs/dof_renumbering.h>
#include <deal.II/dofs/dof_tools.h>

#include <deal.II/fe/fe_q.h>
#include <deal.II/fe/fe_values.h>

#include <deal.II/grid/grid_generator.h>
#include <deal.II/grid/grid_refinement.h>
#include <deal.II/grid/tria.h>
#include <deal.II/grid/tria_accessor.h>
#include <deal.II/grid/tria_iterator.h>

#include <deal.II/lac/constraint_matrix.h>
#include <deal.II/lac/dynamic_sparsity_pattern.h>
#include <deal.II/lac/full_matrix.h>
#include <deal.II/lac/precondition.h>
#include <deal.II/lac/solver_cg.h>
#include <deal.II/lac/sparse_matrix.h>
#include <deal.II/lac/vector.h>

#include <deal.II/numerics/data_out.h>
#include <deal.II/numerics/error_estimator.h>
#include <deal.II/numerics/matrix_tools.h>
#include <deal.II/numerics/vector_tools.h>

#include <deal2lkit/error_handler.h>
#include <deal2lkit/parsed_data_out.h>
#include <deal2lkit/parsed_finite_element.h>
#include <deal2lkit/parsed_function.h>
#include <deal2lkit/parsed_grid_generator.h>
#include <deal2lkit/parsed_grid_refinement.h>

#include <fstream>
#include <iostream>
#include <memory>

namespace Step7
{
  using namespace dealii;
  using namespace deal2lkit;

  template <int dim>
  class HelmholtzProblem
  {
  public:
    HelmholtzProblem();

    ~HelmholtzProblem();

    void run();

  private:
    void setup_system();
  void assemble_system();
  void solve();
  void refine_grid();
  void process_solution(const unsigned int cycle);


  ParsedGridGenerator<dim> pgg;
  ParsedFiniteElement<dim> pfe;

  ParsedFunction<dim> exact_solution;
  ParsedFunction<dim> right_hand_side;

  std::unique_ptr<Triangulation<dim>> triangulation;
  std::unique_ptr<DoFHandler<dim>>    dof_handler;

  std::unique_ptr<FiniteElement<dim>> fe;

  ConstraintMatrix hanging_node_constraints;

  SparsityPattern      sparsity_pattern;
  SparseMatrix<double> system_matrix;

  Vector<double> solution;
  Vector<double> system_rhs;

  ParsedGridRefinement pgr;
  ErrorHandler<1>      eh;
  ParsedDataOut<dim>   data_out;
};

template <int dim>
HelmholtzProblem<dim>::HelmholtzProblem() :
  pgg("Grid", "rectangle","","-1,-1", "1,1"),
  exact_solution("Exact solution",1,"exp(-((x+0.5)^2 +
   (y-0.5)^2)*64.) + exp(-((x+0.5)^2 + (y+0.5)^2)*64.) + exp(-((x-0.5)^2
   + (y+0.5)^2)*64.)"),
  right_hand_side("Right-hand side",1,"-4096*(2.0*x + 1.0)^2*exp(-64.*(x
   + 0.50)^2 - 64.*(y + 0.50)^2) - 4096*(2.0*y + 1.0)^2*exp(-64.*(x +
   0.50)^2 - 64.*(y + 0.50)^2) - 4096*(2.0*x + 1.0)^2*exp(-64.*(x +
   0.50)^2 - 64.*(y - 0.50)^2) - 4096*(2.0*y - 1.0)^2*exp(-64.*(x +
   0.50)^2 - 64.*(y - 0.50)^2) - 4096*(2.0*x - 1.0)^2*exp(-64.*(x -
   0.50)^2 - 64.*(y + 0.50)^2) - 4096*(2.0*y + 1.0)^2*exp(-64.*(x -
   0.50)^2 - 64.*(y + 0.50)^2) + 257.*exp(-64.*(x + 0.50)^2 - 64.*(y +
   0.50)^2) + 257.*exp(-64.*(x + 0.50)^2 - 64.*(y - 0.50)^2) +
   257.*exp(-64.*(x - 0.50)^2 - 64.*(y + 0.50)^2)"),
  pgr("Grid refinement", "number",0.3,0.03)
{}

template <int dim>
HelmholtzProblem<dim>::~HelmholtzProblem()
{
  dof_handler->clear();
}

template <int dim>
void HelmholtzProblem<dim>::setup_system()
{
  dof_handler->distribute_dofs(*fe);
  DoFRenumbering::Cuthill_McKee(*dof_handler);

  hanging_node_constraints.clear();
  DoFTools::make_hanging_node_constraints(*dof_handler,
                                          hanging_node_constraints);
  hanging_node_constraints.close();

  DynamicSparsityPattern dsp(dof_handler->n_dofs(), dof_handler->n_dofs());
```

```
    DoFTools::make_sparsity_pattern(*dof_handler, dsp);
    hanging_node_constraints.condense(dsp);
    sparsity_pattern.copy_from(dsp);

    system_matrix.reinit(sparsity_pattern);

    solution.reinit(dof_handler->n_dofs());
    system_rhs.reinit(dof_handler->n_dofs());
  }

  template <int dim>
  void HelmholtzProblem<dim>::assemble_system()
  {
    QGauss<dim>     quadrature_formula(fe->degree + 1);
    QGauss<dim - 1> face_quadrature_formula(fe->degree + 1 );

    const unsigned int n_q_points      = quadrature_formula.size();
    const unsigned int n_face_q_points = face_quadrature_formula.size();

    const unsigned int dofs_per_cell = fe->dofs_per_cell;

    FullMatrix<double> cell_matrix(dofs_per_cell, dofs_per_cell);
    Vector<double>     cell_rhs(dofs_per_cell);

    std::vector<types::global_dof_index> local_dof_indices(dofs_per_cell);

    FEValues<dim> fe_values(*fe,
                            quadrature_formula,
                            update_values | update_gradients |
                            update_quadrature_points | update_JxW_values);

    FEFaceValues<dim> fe_face_values(*fe,
                                     face_quadrature_formula,
                                     update_values | update_quadrature_points |
                                     update_normal_vectors |
                                     update_JxW_values);

    std::vector<double> rhs_values(n_q_points);

    typename DoFHandler<dim>::active_cell_iterator cell = dof_handler
                                                          ->begin_active(),
                                                   endc = dof_handler->end();
    for (; cell != endc; ++cell)
      {
        cell_matrix = 0;
        cell_rhs    = 0;

        fe_values.reinit(cell);

        right_hand_side.value_list(fe_values.get_quadrature_points(),
                                   rhs_values);

        for (unsigned int q_point = 0; q_point < n_q_points; ++q_point)
          for (unsigned int i = 0; i < dofs_per_cell; ++i)
            {
              for (unsigned int j = 0; j < dofs_per_cell; ++j)
                cell_matrix(i, j) += ((fe_values.shape_grad(i, q_point) *
                                       fe_values.shape_grad(j, q_point) +
                                     fe_values.shape_value(i, q_point) *
                                       fe_values.shape_value(j, q_point)) *
                                    fe_values.JxW(q_point));

              cell_rhs(i) += (fe_values.shape_value(i, q_point) *
                              rhs_values[q_point] * fe_values.JxW(q_point));
            }
      for (unsigned int face_number = 0;
           face_number < GeometryInfo<dim>::faces_per_cell;
           ++face_number)
        if (cell->face(face_number)->at_boundary() &&
            (cell->face(face_number)->boundary_id() == 1))
          {
            fe_face_values.reinit(cell, face_number);

            for (unsigned int q_point = 0; q_point < n_face_q_points;
                 ++q_point)
              {
                const double neumann_value =
                  (exact_solution.gradient(
                     fe_face_values.quadrature_point(q_point)) *
                   fe_face_values.normal_vector(q_point));

                for (unsigned int i = 0; i < dofs_per_cell; ++i)
                  cell_rhs(i) +=
                    (neumann_value * fe_face_values.shape_value(i, q_point) *
                     fe_face_values.JxW(q_point));
              }
          }

      cell->get_dof_indices(local_dof_indices);
      for (unsigned int i = 0; i < dofs_per_cell; ++i)
        {
          for (unsigned int j = 0; j < dofs_per_cell; ++j)
            system_matrix.add(
              local_dof_indices[i], local_dof_indices[j], cell_matrix(i, j));

          system_rhs(local_dof_indices[i]) += cell_rhs(i);
        }
    }

  hanging_node_constraints.condense(system_matrix);
  hanging_node_constraints.condense(system_rhs);

  std::map<types::global_dof_index, double> boundary_values;
  VectorTools::interpolate_boundary_values(
    *dof_handler, 0, exact_solution, boundary_values);
  MatrixTools::apply_boundary_values(
    boundary_values, system_matrix, solution, system_rhs);
}

template <int dim>
void HelmholtzProblem<dim>::solve()
{
  SolverControl solver_control(1000, 1e-12);
  SolverCG<>    cg(solver_control);

  PreconditionSSOR<> preconditioner;
  preconditioner.initialize(system_matrix, 1.2);

  cg.solve(system_matrix, solution, system_rhs, preconditioner);

  hanging_node_constraints.distribute(solution);
}

template <int dim>
void HelmholtzProblem<dim>::refine_grid()
{
  Vector<float> estimated_error_per_cell(triangulation->n_active_cells());

  KellyErrorEstimator<dim>::estimate(*dof_handler,
                                     QGauss<dim - 1>(3),
                                     typename FunctionMap<dim>::type(),
                                       solution,
                                       estimated_error_per_cell);

    pgr.mark_cells(estimated_error_per_cell, *triangulation);
    triangulation->execute_coarsening_and_refinement();
  }

  template <int dim>
  void HelmholtzProblem<dim>::process_solution(const unsigned int cycle)
  {
    eh.error_from_exact(*dof_handler, solution, exact_solution);
    data_out.prepare_data_output(*dof_handler, std::to_string(cycle));
    data_out.add_data_vector(solution, "solution");
    data_out.write_data_and_clear();

    const unsigned int n_active_cells = triangulation->n_active_cells();
    const unsigned int n_dofs         = dof_handler->n_dofs();

    std::cout << "Cycle " << cycle << ':' << std::endl
              << "   Number of active cells:       " << n_active_cells
              << std::endl
              << "   Number of degrees of freedom: " << n_dofs << std::endl;
  }

  template <int dim>
  void HelmholtzProblem<dim>::run()
  {
    const unsigned int n_cycles = 7;
    for (unsigned int cycle = 0; cycle < n_cycles; ++cycle)
      {
        if (cycle == 0)
          {
    triangulation.reset(pgg.serial());
            fe.reset(pfe());
    dof_handler.reset(new DoFHandler<dim>{*triangulation});
    triangulation->refine_global(2);
          }
        else
          refine_grid();

        setup_system();

        assemble_system();
        solve();

        process_solution(cycle);
      }
    eh.output_table();
  }

} // namespace Step7

int main(int argc, char **argv)
{
  const unsigned int               dim = 2;

  try
    {
      using namespace dealii;
      using namespace Step7;

      HelmholtzProblem<dim> helmholtz_problem_2d;
      deal2lkit::ParameterAcceptor::initialize("parameters.prm",
                                               "used_parameters.prm");
      helmholtz_problem_2d.run();
    }
    catch (std::exception &exc)
      {
        std::cerr << std::endl
                  << std::endl
                  << "----------------------------------------------------"
                  << std::endl;
        std::cerr << "Exception on processing: " << std::endl
                  << exc.what() << std::endl
                  << "Aborting!" << std::endl
                  << "----------------------------------------------------"
                  << std::endl;
        return 1;
      }
    catch (...)
      {
        std::cerr << std::endl
                  << std::endl
                  << "----------------------------------------------------"
                  << std::endl;
        std::cerr << "Unknown exception!" << std::endl
                  << "Aborting!" << std::endl
                  << "----------------------------------------------------"
                  << std::endl;
        return 1;
      }

    return 0;
  }
  }
```

References

[1] Bangerth W, Davydov D, Heister T, Heltai L, Kanschat G, Kronbichler M, et al. The deal. II library, version 8.4. J Numer Math 2016;24.

[2] Mola A, Heltai L, DeSimone A. A fully nonlinear potential model for ship hydrodynamics directly interfaced with CAD data structures. In: Proceedings of the twenty-sixth (2016) international ocean and polar engineering conference. 2016. p. 511–18.

[3] Giuliani N, Mola A, Heltai L. π - BEM : A flexible parallel implementation for adaptive, geometry aware, and high order boundary element methods. Adv Eng Softw 2018;121:39–58. http://dx.doi.org/10.1016/j.advengsoft.2018.03.008.

[4] The π-DoMUS Authors. π-DoMUS. Parallel deal. II MUlti-physics Solver. 2016. https://github.com/mathLab/pi-DoMUS.

[5] Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, et al. Sundials: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans Math Software 2005;31(3):363–96. http://dx.doi.org/10.1145/1089014.1089020, URL http://doi.acm.org/10.1145/1089014.1089020.

[6] Hindmarsh AC, Serban R, Collier A. User documentation for ida v2. 8.1 (sundials v2. 6.1). 2015.

[7] Open Asset Import Library Web page. 2016. URL http://assimp.sourceforge.net/index.html.

[8] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: Elements of reusable object-oriented software. Addison-Wesley Professional; 1994, URL https://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201633612.

[9] Mola A, Sartori A, Bardelloni M, Heltai L, Bangerth W. Using exact geometries in finite element computations. 2018;[in preparation].

[10] Maier M, Bardelloni M, Heltai L. LinearOperator—a generic, high-level expression syntax for linear algebra. Comput Math Appl 2016;72(1):1–24. http://dx.doi.org/10.1016/j.camwa.2016.04.024.

[11] Reinders J. Intel threading building blocks, 1st ed. O'Reilly & Associates, Inc.; 2007.

[12] Bangerth W, Hartmann R, Kanschat G. Deal. II – a general purpose object oriented finite element library. ACM Trans Math Software 2007;33(4):24/1–27.

[13] Bangerth W, Burstedde C, Heister T, Kronbichler M. Algorithms and data structures for massively parallel generic adaptive finite element codes. ACM Trans Math Software 2012;38(2):14:1–28. http://dx.doi.org/10.1145/2049673.2049678, URL http://doi.acm.org/10.1145/2049673.2049678.

[14] Bangerth W, Kayser-Herold O. Data structures and requirements for Hp finite element software. ACM Trans Math Software 2009;36(1):4:1–31. http://dx.doi.org/10.1145/1486525.1486529, URL http://doi.acm.org/10.1145/1486525.1486529.

[15] Janssen B, Kanschat G. Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods. SIAM J Sci Comput 2011;33(4):2095–114. http://dx.doi.org/10.1137/090778523.

[16] Kanschat G. Multilevel methods for discontinuous Galerkin {FEM} on locally refined meshes. Comput Struct 2004;82(28):2437–45. http://dx.doi.org/10.1016/j.compstruc.2004.04.015, URL http://www.sciencedirect.com/science/article/pii/S0045794904002603.

[17] Kronbichler M, Kormann K. A generic interface for parallel cell-based finite element operator application. Comput & Fluids 2012;63:135–47. http://dx.doi.org/10.1016/j.compfluid.2012.04.012, URL http://www.sciencedirect.com/science/article/pii/S0045793012001429.

[18] The deal2lkit Authors. A toolkit library for deal. II. In: GitHub repository. GitHub; 2015, http://dx.doi.org/10.5281/zenodo.58013, https://github.com/mathLab/deal2lkit.