```
/* ---------------------------------------------------------------------
 *
 * Copyright (C) 1999 - 2021 by the deal.II authors
 *
 * This file is part of the deal.II library.
 *
 * The deal.II library is free software; you can use it, redistribute
 * it, and/or modify it under the terms of the GNU Lesser General
 * Public License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 * The full text of the license can be found in the file LICENSE.md at
 * the top level directory of deal.II.
 *
 * ---------------------------------------------------------------------

 *
 * Authors: Wolfgang Bangerth, 1999,
 *          Guido Kanschat, 2011
 */



#include <deal.II/grid/tria.h>
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/grid/grid_generator.h>

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

#include <deal.II/dofs/dof_tools.h>

#include <deal.II/fe/fe_values.h>
#include <deal.II/base/quadrature_lib.h>

#include <deal.II/base/function.h>
#include <deal.II/base/function_lib.h>
#include <deal.II/base/symbolic_function.h>
#include <deal.II/base/function_spherical.h>
#include <deal.II/differentiation/sd/symengine_number_types.h>
#include <deal.II/numerics/vector_tools.h>
#include <deal.II/numerics/matrix_tools.h>

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


#include <deal.II/numerics/data_out.h>
```

```
#include <deal.II/numerics/data_out.h>
#include <fstream>
#include <iostream>

using namespace dealii;

// const dealii::Differentiation::SD::Expression x("x");
// const dealii::Differentiation::SD::Expression y("y");
//   // This is a symbolic expression, which is an expression constructed
//   // from individual symbols.
// const dealii::Differentiation::SD::Expression f = (x + y)*(x + y);

class Step3
{
public:
  Step3();

  void run();


private:
  void make_grid();
  void setup_system();
  void assemble_system();
  void solve();
  void output_results() const;

  Triangulation<2> triangulation;
  FE_Q<2>          fe;
  DoFHandler<2>    dof_handler;

  SparsityPattern      sparsity_pattern;
  SparseMatrix<double> system_matrix;

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


Step3::Step3()
  : fe(1)
  , dof_handler(triangulation)
{}



void Step3::make_grid()
{
  GridGenerator::hyper_rectangle(triangulation, {-2,-6.28},{2,6.28});
  triangulation.refine_global(5);
```

```
  std::cout << "Number of active cells: " << triangulation.n_active_cells()
            << std::endl;
}




void Step3::setup_system()
{
  dof_handler.distribute_dofs(fe);
  std::cout << "Number of degrees of freedom: " << dof_handler.n_dofs()
            << std::endl;

  DynamicSparsityPattern dsp(dof_handler.n_dofs());
  DoFTools::make_sparsity_pattern(dof_handler, dsp);
  sparsity_pattern.copy_from(dsp);

  system_matrix.reinit(sparsity_pattern);

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




void Step3::assemble_system()
{
  QGauss<2> quadrature_formula(fe.degree + 1);
  FEValues<2> fe_values(fe,
                        quadrature_formula,
                        update_values | update_gradients | update_JxW_values);

  const unsigned int dofs_per_cell = fe.n_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);

  for (const auto &cell : dof_handler.active_cell_iterators())
    {
      fe_values.reinit(cell);

      cell_matrix = 0;
      cell_rhs    = 0;

      for (const unsigned int q_index : fe_values.quadrature_point_indices())
        {
          for (const unsigned int i : fe_values.dof_indices())
```

```
          for (const unsigned int j : fe_values.dof_indices())
            cell_matrix(i, j) +=
              (fe_values.shape_grad(i, q_index) * // grad phi_i(x_q)
               fe_values.shape_grad(j, q_index) * // grad phi_j(x_q)
               fe_values.JxW(q_index));           // dx

        for (const unsigned int i : fe_values.dof_indices())
          cell_rhs(i) += (fe_values.shape_value(i, q_index) * // phi_i(x_q)
                          1. *                                // f(x_q)
                          fe_values.JxW(q_index));            // dx
      }
    cell->get_dof_indices(local_dof_indices);

    for (const unsigned int i : fe_values.dof_indices())
      for (const unsigned int j : fe_values.dof_indices())
        system_matrix.add(local_dof_indices[i],
                          local_dof_indices[j],
                          cell_matrix(i, j));

    for (const unsigned int i : fe_values.dof_indices())
      system_rhs(local_dof_indices[i]) += cell_rhs(i);
  }


  std::map<types::global_dof_index, double> boundary_values;

  SymbolicFunction<2> fun("x^2+y; t*x*y");

  VectorTools::interpolate_boundary_values(dof_handler,
                                           0,
                                           Functions::SymbolicFunction<2> ("x^2+y"),
                                           boundary_values);
  MatrixTools::apply_boundary_values(boundary_values,
                                     system_matrix,
                                     solution,
                                     system_rhs);
}



void Step3::solve()
{
  SolverControl solver_control(1000, 1e-12);
  SolverCG<Vector<double>> solver(solver_control);

  solver.solve(system_matrix, solution, system_rhs, PreconditionIdentity());
}
```

```
void Step3::output_results() const
{
  DataOut<2> data_out;
  data_out.attach_dof_handler(dof_handler);
  data_out.add_data_vector(solution, "solution");
  data_out.build_patches();

  std::ofstream output("solution.vtk");
  data_out.write_vtk(output);
}



void Step3::run()
{
  make_grid();
  setup_system();
  assemble_system();
  solve();
  output_results();
}



int main()
{
  deallog.depth_console(2);

  Step3 laplace_problem;
  laplace_problem.run();

  return 0;
}
```