Lab 06

Convection-diffusion problems. deal.II step-9.

Advanced Topic in Scientific Computing - SISSA, UniTS, 2024-2025

Pasquale Claudio Africa

25 Oct 2024

Assignment 1

In this exercise, you will extend the problem in `step-6` by adding a convection term to the partial differential equation (PDE) being solved. Convection-dominated problems are often prone to numerical instabilities when solved using standard finite element methods without stabilization techniques.

Step-by-step instructions (1/2)

1. **Review the current formulation**:
   The current problem involves solving the following Laplace equation:
   $$-\nabla \cdot (a(\mathbf{x})\nabla u) = f \quad \text{in } \Omega,$$
   where a is the diffusion coefficient.
2. **Modify the equation**:
   Add a convection term to the equation, resulting in:
   $$-\nabla \cdot (a(\mathbf{x})\nabla u) + \mathbf{b} \cdot \nabla u = f \quad \text{in } \Omega,$$
   where **b** is the convection vector. The new term $\mathbf{b} \cdot \nabla u$ models the convection.
3. **Update the weak form**:
   Modify the weak form of the equation to account for the convection term. Implement this weak form in your finite element code by updating the relevant matrix assembly functions.

Step-by-step instructions (2/2)

4. **Set up the diffusion coefficient and the convection vector $\mathbf{b}$**:
   For simplicity, you can use, for example:
   $$a(\mathbf{x}) = 1, \quad \mathbf{b} = [1000, 1000]^T.$$
   Update the source code to include this convection vector in the matrix assembly.

   ⚠️ **Warning**: with convection terms, the resulting matrix is not symmetric anymore! Choose a proper linear solver and preconditioner.

5. **Run the code and analyze the results**:
   Compile and run the modified problem. Visualize the numerical solution. In convection-dominated scenarios, you should observe oscillations or numerical instabilities in regions where convection dominates diffusion. This instability is due to the fact that the standard Galerkin method does not provide sufficient stability for convection-dominated problems.

Questions to explore

- What happens to the solution as the convection term becomes stronger?
- How does the mesh refinement affect the solution stability?
- What types of numerical instabilities do you observe, and where do they occur in the domain?

Bonus challenge

Once you observe the instabilities, try implementing a simple stabilization technique, such as artificial diffusion or SUPG, as discussed in `step-9`, to improve the numerical stability of the solution. Compare the results with and without stabilization.

Convergence test

Consider the exact problem defined in `ad_exact_solution.pdf` over the domain $\Omega = (0,1)^2$, for some $\varepsilon \ll 1$. Compute the L^2 and H^1 errors and verify convergence estimates w.r.to the mesh size h.

`step-9`

This tutorial describes the parallelization of a finite element solver for an advection problem using shared memory parallelism via the `WorkStream` concept in deal.II. The focus is on parallelizing the assembly of the global system matrix and right-hand side vector, improving computational efficiency.

The key part of this tutorial is the assembly of the system matrix and right-hand side vector using shared memory parallelism.

Parallel assembly of the system

Two data structures are used to store data for each cell in a thread-safe manner:

```
struct AssemblyScratchData // Temporary storage for each task.
{
    FEValues<dim> fe_values;
    std::vector<double> rhs_values;
    AdvectionField<dim> advection_field;
};

struct AssemblyCopyData // Thread-safe result storage.
{
    FullMatrix<double> cell_matrix;
    Vector<double> cell_rhs;
    std::vector<types::global_dof_index> local_dof_indices;
};

WorkStream::run(begin, end, worker_function, copier_function, ScratchData(), CopyData());
```

`WorkStream`: parallel assembly

`WorkStream` is used to break down the assembly into tasks that can be processed independently by multiple threads.

```
void assemble_system()
{
    WorkStream::run(dof_handler.begin_active(),
                    dof_handler.end(),
                    *this,
                    &AdvectionProblem::local_assemble_system,
                    &AdvectionProblem::copy_local_to_global,
                    AssemblyScratchData(fe),
                    AssemblyCopyData());
}
```

`WorkStream::run` divides the cells into separate tasks for parallel processing. The `local_assemble_system` function computes element-wise contributions, and `copy_local_to_global` accumulates these local results into the global matrix.

Local assembly example

```
void local_assemble_system(const typename DoFHandler<dim>::active_cell_iterator &cell,
                           AssemblyScratchData &scratch,
                           AssemblyCopyData &copy_data)
{
    scratch.fe_values.reinit(cell);
    cell->get_dof_indices(copy_data.local_dof_indices);

    // Assembly loop.
}
```

```
template <int dim>
void AdvectionProblem<dim>::copy_local_to_global(const AssemblyCopyData &copy_data)
{
    hanging_node_constraints.distribute_local_to_global(
      copy_data.cell_matrix,
      copy_data.cell_rhs,
      copy_data.local_dof_indices,
      system_matrix,
      system_rhs);
}
```

Assignment 2

1. Compute the speedup of the execution time, i.e. investigating thread scalability with WorkStream by varying the number of threads and analyzing the impact on performance.
2. Generalize the code to 3D, making sure to account for changes in mesh generation, element degrees of freedom, and boundary conditions.

Bonus challenge

Parallelize the local residual computation from `lab05` using `WorkStream`.