Numerical Solution of PDEs Using the Finite Element Method

Exercise 1: Creating and manipulating `Triangulations`

Jean-Paul Pelteret (`jean-paul.pelteret@fau.de`)
Luca Heltai (`luca.heltai@sissa.it`)

19 March 2018

Some useful resources

`https://www.dealii.org/8.5.1/doxygen/deal.II/step_1.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/step_49.html`

1. Using `step-1` as a base:
   (a) Compile and run this tutorial on the command line or inside a suitable IDE, and inspect at the output.
   (b) Create a helper function that takes a reference to a Triangulation and prints the following information:
      - number of levels
      - number of cells
      - number of active cells

      Test this with all of your meshes.
2. Modifying an existing meshing function
   (a) Comment out the `.set_manifold(0, ...)` line in `second_grid()`. What happens now?
   (b) Output mesh two as an svg file instead of eps. Open it in a browser to display it (`Firefox`, for example).
   (c) Go into `second_grid()` and remove the last line (`.set_manifold(0);`). The program will crash when you run it. Try to find out what is going on by debugging the program (e.g. For `Qt Creator`: "Debug" $\rightarrow$ "Start debugging") and stepping through the function `second_grid()`. You can fix this problem in a more elegant way than putting the line you removed back in. How? See the tutorial description for more info.
3. Creating a mesh from scratch
   (a) Generate a circle using `GridGenerator::hyper_ball()` in 2d (add a function `third_grid()` to `step-1`).
      i. Use a `SphericalManifold` everywhere, only on the boundary, or on all cells except the center cell and refine the mesh globally twice.
      ii. Set the output format of the previous example to `vtk` and inspect the mesh in `Paraview`.
   (b) Create an image of an L-shape domain with one global refinement.
      i. Inspect the mesh in `Paraview`.
      ii. Refine the L-shaped mesh adaptively:
         α) Refine all cells with the distance between the center of the cell and re-entrant corner is smaller than $\frac{1}{3}$.
         β) Refine exactly at the re-entrant corner (i.e. those with the corner as a vertex) several times.
4. Reading in a mesh
   (a) Take a look at `step-49` and read the included `.msh` file in your modified `step-1` program.

(b) Add two levels of refinement to the cells at the boundary of the cut-outs.

5. Additional tasks

   (a) Create a mesh that represents the surface of a torus and refine it 2 times globally. Output to `vtk` format and check the output. Note that your `Triangulation` needs to be of type `Triangulation<2,3>` (not explicitly discussed in this course).

Numerical Solution of PDEs Using the Finite Element Method

Exercise 2: Creating a DoFHandler and visualising sparsity patterns

Jean-Paul Pelteret (jean-paul.pelteret@fau.de)
Luca Heltai (luca.heltai@sissa.it)

19 March 2018

Some useful resources

https://www.dealii.org/8.5.1/doxygen/deal.II/step_2.html
https://www.dealii.org/8.5.1/doxygen/deal.II/namespaceDoFRenumbering.html

1. Using `step-2` as a base:
   (a) Compile and run this tutorial, and inspect at the output.
   (b) Look at the generated sparsity patterns (`Firefox`, for example).
2. Investigate:
   (a) How does the pattern change if you increase the polynomial degree from 1 to 2, or to 3?
   (b) How does the pattern change if you use a globally refined (say 3 times) unit square?
   (c) Are these patterns symmetric? Why/why not?
   (d) How many entries per row in the sparsity pattern do you expect for a `Q1` element (assuming four cells are around each vertex)?
      i. Check that this is true for the mesh in (b) (look for `row_length(i)` and output them for each row).
      ii. Can you construct a 2d mesh (without hanging nodes) that has a row with more entries?
   (e) How many entries per row in the sparsity pattern are there for `Q2` and `Q3` elements, again assuming four cells around each vertex?
   (f) Print all entries for row 42 for the original renumbered sparsity pattern.
   (g) Renumber the DoFs using the `boost::king_ordering` algorithm. What does the sparsity pattern look like now?
3. Additional tasks
   (a) Compute and output statistics like the number of unknowns, bandwidth of the sparsity pattern, average number of entries per row, and fill ratio.
   (b) Investigate the other appropriate DoF renumbering schemes. Which one produces the most banded structure?
   (c) Repeat the above for increasing refinement levels. Which is the most efficient scheme (lets say, in terms of bandwidth reduction versus computational time expended)? You can get an estimate of the time for this operation like this:

      ```
      $ time ./step-2
      ```

   (d) What happens if you change the mesh from 2d to 3d?
   (e) Investigate the sparsity patterns generated for other types of `FiniteElement`s of varying order.

NUMERICAL SOLUTION OF PDES USING THE FINITE ELEMENT METHOD

EXERCISE 3: SOLVING THE POISSON PROBLEM

Jean-Paul Pelteret (jean-paul.pelteret@fau.de)
Luca Heltai (luca.heltai@sissa.it)

19 March 2018

Some useful resources

https://www.dealii.org/8.5.1/doxygen/deal.II/step_3.html
https://www.dealii.org/8.5.1/doxygen/deal.II/step_4.html

---

1. Using `step-3` as a base:
   (a) Compile and run this tutorial, and inspect at the output.
   (b) Modify the code so that the problem is dimension-independent.
   (c) Switch to `vtk` output and visualize in `Paraview`. Figure out how to warp the solution by the solution variable.
   (d) Add a zero Neumann boundary condition to one edge of the domain. Assign this Neumann boundary the indicator 1.
   Tip: Look at the instructions in "Modify the type of boundary condition" in the "Possibilities for extensions" section of the tutorial.
   (e) Add a non-zero Dirichlet boundary condition to one edge of the domain.
      i. Set the value to -0.5 for the boundary with indicator 1.
      Tip: Look at the instructions in "A slight variation of the last point" in the "Possibilities for extensions" section of the tutorial.
      ii. Change the setup to have $f = 0$. Compare this result to that where f is non-zero.
2. Additional tasks
   (a) Do "Convergence of the mean". Can you see the order h^2?
   (b) Increase the polynomial order (you need to increase all orders of the quadratures in the program!) and check the convergence of the mean now.
   (c) Switch to an L-shaped domain and experiment with a combination of Dirichlet and Neumann boundary conditions. By experimentation, identify the faces adjacent to the re-entrant corner and apply Dirichlet conditions only there.
   Tip: There is more than one way to generate such a grid using the built-in functions.

Numerical Solution of PDEs Using the Finite Element Method

Exercise 4: Global and local error computation and estimation

Jean-Paul Pelteret (jean-paul.pelteret@fau.de)
Luca Heltai (luca.heltai@sissa.it)

20 March 2018

Some useful resources

https://www.dealii.org/8.5.1/doxygen/deal.II/step_6.html
https://www.dealii.org/8.5.1/doxygen/deal.II/step_7.html
https://www.dealii.org/8.5.1/doxygen/deal.II/group__numerics.html

1. Using `step-5` (or your previously modified version of `step-3`) as a base:
   (a)
   $$\begin{aligned} -\Delta u(\mathbf{x}) = f(\mathbf{x}) \quad &\text{in} \quad \Omega \in [0,1]^2, \quad \text{with} \\ u(\mathbf{x}) = \bar{u}(\mathbf{x}) \quad &\text{on} \quad \partial\Omega, \quad \text{and} \end{aligned}$$
   (b) Set the boundary conditions $\bar{u}(\mathbf{x})$ and forcing function $f(\mathbf{x})$ to get the manufactured solution
   $$u(\mathbf{x}) = \sin(\pi x_1)\ \cos(\pi x_2).$$
   Make sure the $\mathcal{L}^2$ errors are converging.
   Tip: Look at the `VectorTools::integrate_difference` function.
   (c) Implement the computation of the $\mathcal{H}^1$ error.
   Tip: For this you need to compute and implement the gradient of the manufactured solution.
   (d) Use the `KellyErrorEstimator` to predict where the regions of geometry where the solution approximation is inaccurate. Visualise this error using `Paraview`. Do you observe any correlation between the gradient of the solution and the estimated local solution error?
   Tip: Use a different quadrature rule to prevent super-convergent effects when using the `KellyErrorEstimator`.
   (e) Perform local cell marking and refinement using the cell-based estimated error. For this, the logic of the `refine_mesh` function must be modified.

2. Additional tasks
   (a) Compare the convergence rates (number of DoFs versus the solution error, best viewed in a log-log plot) when using global refinement and when using local refinement with the Kelly error estimator.
   (b) Investigate the influence of the coarsening and refinement parameters on the solution accuracy.
   (c) Investigate the effect of changing the polynomial order for the solution ansatz on the solution accuracy.
   (d) Integrate a non-constant coefficient into the governing equation, i.e. solve the heterogeneous Poisson equation
   $$-\alpha(\mathbf{x})\Delta u(\mathbf{x}) = f(\mathbf{x}) \quad \text{in} \quad \Omega$$
   where $\alpha(\mathbf{x})$ represents some material parameter. Repeat the calculation of the error using the `KellyErrorEstimator`, while taking this spatially dependent coefficient into consideration.
   Tip: Look at the documentation for the `KellyErrorEstimator` before deciding on how to implement $\alpha(\mathbf{x})$.

(e) Choose $\alpha(\mathbf{x})$ to be spatially discontinuous. Do you observe any correlation between the location of the material discontinuity and the estimated local solution error? What influence does this have on the location of the refined cells? Tip: Verify your conclusions by looking to the results of `step-6`. Further information can be found in the discussion "Playing with the regularity of the solution" in the "Possibilities for extensions" section of `step-6`.

Numerical Solution of PDEs Using the Finite Element Method

Exercise 5: Local refinement, hanging nodes and the ConstraintMatrix

Jean-Paul Pelteret (jean-paul.pelteret@fau.de)
Luca Heltai (luca.heltai@sissa.it)

20 March 2018

Some useful resources

`https://www.dealii.org/8.5.1/doxygen/deal.II/step_4.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/step_5.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/namespaceGridGenerator.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/namespaceVectorTools.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/namespaceDoFTools.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/group__constraints.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/step_49.html`

1. Using `step-5` (or your previously modified version of `step-3`) as a base:
   (a) Solve Laplace's equation
   $$\begin{aligned} -\Delta u(\mathbf{x}) &= 0 \quad \text{in} \quad \Omega, \quad \text{with} \\ u(\mathbf{x}) &= 0 \quad \text{on} \quad \partial\Omega_1, \quad \text{and} \\ u(\mathbf{x}) &= 1 \quad \text{on} \quad \partial\Omega_2 \end{aligned}$$
   on an L-shaped domain with length and width of dimension 1. Here, Ω_2 denotes one of the end-edges of the L, and $\partial\Omega_1 = \partial\Omega \backslash \partial\Omega_2$.
   (b) Starting from a coarse grid, perform successive global refinements to increase the accuracy of the solution and to locate the singularity.
   (c) Now switch from using global refinement to using local refinement in the vicinity of the singularity. To accomplish this, you'll need to build the hanging node constraints.
   (d) Build the Dirichlet boundary directly into a global `ConstraintMatrix`, along with the existing hanging node constraints. With this change, several parts of your code need to be modified:
      i. The constraints need to be built in the `setup` function.
      Tip: You can use the `VectorTools::interpolate` function here.
      ii. The `ConstraintMatrix` should be used to distribute local cell and vector contributions to their global counterparts.
      iii. The constraints must be distributed to the solution.
2. Additional tasks
   (a) Following (c), what happens if you "forget" to distribute the (hanging node) constraints after solving the linear system?
   (b) Instead of using the `VectorTools::interpolate` function, construct the Dirichlet contributions to the `ConstraintMatrix` manually.
   Tip: There are two ways to accomplish this: (1) Use the tools provided in the `DoFTools` namespace, or (2) loop over cell faces and interrogate them for global DoFs that have support there.
   (c) Experiment with some of the other grids in the `GridGenerator` namespace, such as `GridGenerator::hyper_cross` and `GridGenerator::cheese`, or create your own by using some of the grid modification tools discussed in `step-49`. In each case, find ways to efficiently increase the cell density in the location of singularities.

(d) Using your existing code, replicate the study performed in `step-5`. Note that the governing equation has changed slightly. Can you further improve the accuracy of the result in `step-5` using Manifolds?
Tip: Look to the discussion "A better mesh" in the "Possibilities for extensions" section of `step-6`.

Numerical Solution of PDEs Using the Finite Element Method

Exercise 6: Shared memory parallelisation

Jean-Paul Pelteret (`jean-paul.pelteret@fau.de`)
Luca Heltai (`luca.heltai@sissa.it`)

20 March 2018

Some useful resources

`https://www.dealii.org/8.5.1/doxygen/deal.II/step_6.html`
`https://dealii.org/8.5.1/doxygen/deal.II/group__threads.html`
`https://dealii.org/8.5.1/doxygen/deal.II/classThreads_1_1TaskGroup.html`
`https://www.dealii.org/8.5.1/doxygen/deal.II/namespaceWorkStream.html`

1. Using `step-6` or the outcome of the previous exercise as a base:
   (a) Parallelise the following parts of your code using `TaskGroup` class:
      i. The system setup function. Can you parallelise the two calls the fill the `constraints`?
      ii. The assembly loop.
      iii. The (manual) calculation of the solution $\mathcal{L}^2$ norm.
   (b) Parallelise the following parts of your code using `TBB` via the `WorkStream` class:
      i. The assembly loop.
      ii. The (manual) calculation of the solution $\mathcal{L}^2$ norm.
   (c) Investigate the possible speed-up by playing around with the number of threads set in the call to `Utilities::MPI::MPI_InitFinalize`.
   Tip: This class needs to be created in the `main` file before you create and execute your problem class.
2. Additional tasks
   (a) What influence do the `queue_length` and `chunk_size` have on the efficiency of the various parallel operations that you have implemented?
   (b) Perform some timings and compare the results:
      i. The serial version of this code.
      ii. The TBB threaded version of the code, but enforcing the use of one thread via the call to `Utilities::MPI::MPI_InitFinalize`.
      iii. The TBB threaded version of the code using the maximum number of threads.

Numerical Solution of PDEs Using the Finite Element Method

Exercise 7 & 8: MPI parallelisation: `parallel::shared::Triangulation` and `parallel::distributed::Triangulation`

Jean-Paul Pelteret (jean-paul.pelteret@fau.de)
Luca Heltai (luca.heltai@sissa.it)

21 March 2018

Some useful resources

https://www.dealii.org/8.5.1/doxygen/deal.II/step_17.html
https://www.dealii.org/8.5.1/doxygen/deal.II/step_18.html
https://www.dealii.org/8.5.1/doxygen/deal.II/step_40.html
https://www.dealii.org/8.5.1/doxygen/deal.II/group__distributed.html
https://www.dealii.org/8.5.1/doxygen/deal.II/group__TrilinosWrappers.html
https://www.dealii.org/8.5.1/doxygen/deal.II/group__PETScWrappers.html

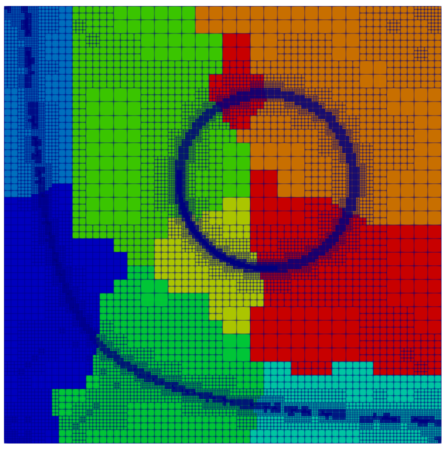
1. Using the supplied modified version of `step-6` as a base:
   (a) For the first version of this code, use `parallel::shared::Triangulation` and solve the 2d non-homogeneous Poisson equation

$$\begin{aligned} -\alpha(\mathbf{x})\Delta u(\mathbf{x}) &= f(\mathbf{x}) \quad \text{in} \quad \Omega \in [0,1]^2, \quad \text{with} \\ u(\mathbf{x}) &= 0 \quad \text{on} \quad \partial\Omega, \end{aligned}$$

   where

$$\alpha(\mathbf{x}) = \begin{cases} 5 & \text{if} \quad |\mathbf{x}-\mathbf{c}| < 0.2 \\ 1 & \text{otherwise,} \end{cases} \quad \text{and} \quad f(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad |x_1 x_2| > 0.05 \\ -10 & \text{otherwise,} \end{cases}$$

   and $\mathbf{c} = [0.6, 0.6]^T$. The result similar to the following:


(a) Mesh and partitioning

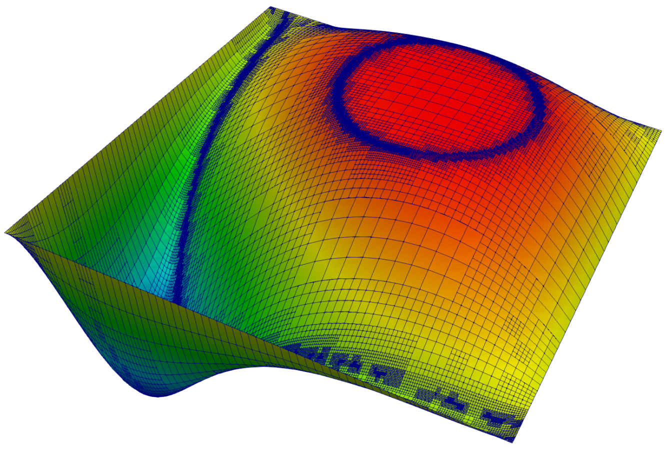

(b) Warped solution

Figure 1: Result produced from 3 initial global refinements and 8 refinement cycles, as visualised in `Paraview`.

These are the rough steps that you'll need to take to achieve this (look for the `TODO`'s listed in the minimal code):

i. Implement the functions defining the material coefficient $\alpha(\mathbf{x})$ and forcing function $f(\mathbf{x})$. Extra credit for using a `dealii::Function` for this purpose.

ii. Initialise the MPI environment correctly in the `main` function.

iii. In the `Step6` class constructor, initialise the class member variables correctly.

iv. In the `setup_system` function, initialise the sparsity pattern, system matrix, solution and RHS vectors correctly.

v. In the `assemble_system` function:

   α) Configure the range of cells over which the assembly is performed.

   β) Implement the forcing function.

   γ) Ensure synchronisation of the elements of the linear system at the end of the assembly loop.

vi. In the `solve` function, correctly choose the template parameter for the conjugate gradient solver, and select an appropriate preconditioner.

vii. In the `refine_grid` function, create the vector with entries required by the `KellyErrorEstimator`.

viii. In the `output_results` function, correctly construct the solution vector to be passed to `DataOut` for later processing and visualisation.

(b) Repeat the above using a `parallel::distributed::Triangulation`.

2. Additional tasks

(a) Compare the distribution of cells across the processes for the two implementations. Is there a difference and, if so, why?

(b) For this problem, measure the performance difference between the two implementations. What, do you think, are the primary factors affecting any differences you notice?

(c) Investigate some of the various options for solvers (direct and iterative) and preconditioners. For example, `Trilinos` offers a direct solver, and its own implementation of iterative solvers, and numerous preconditioners. Consider the properties of the linear system when deciding which options/combinations to test.

(d) Similar to the previous task, investigate the use of `PETSc` as the parallel linear algebra library.