LINMA2710 – Scientific Computing – 2021-2022

Project 2

Deadline: Friday 1th April 2022 18:15

Statement

Propose an implementation of the explicit Euler scheme[1] for the diffusion problem

$$\frac{\partial u}{\partial t} = D(x,y)\Big(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\Big) + f(x,y)$$

where $D(x,y)$ is the *diffusion coefficient*. Note that we consider $D(x,y)\nabla \cdot \nabla u$ instead of $\nabla \cdot (D(x,y)\nabla u)$ to simplify the problem.

You are given `DiffusionProblem.hpp` and `EulerDiffusionSolver.hpp`. On this basis, you should implement the classes `DiffusionProblem` representing a diffusion problem, and `EulerDiffusionProblem` representing a solver based on the Euler scheme. To implement the solver use your implementation of `SparseMatrix` from the first project. The descriptions of the methods to implement are available in the parent class header files.

Two other files are furnished: `test.cpp` to do a basic test of your implementation; and `main.cpp` to create your own diffusion problem.

We consider a diffusion problem in two dimensions x, y. The domain is a rectangle $[0, L_x] \times [0, L_y]$ where L_x and L_y are provided by the `DiffusionProblem`. The initial value of the function u being integrated is given at $t = 0$ in `DiffusionProblem` and it should be integrated until L_t. You should use a grid of n_t time intervals and n_x (resp. n_y) intervals in the dimension x (resp. y), the dimension of the state space is thus $(n_x + 1, n_y + 1)$. There are 4 boundary conditions: at $x = 0$, at $x = L_x$, at $y = 0$ and at $y = L_y$. All boundary conditions are Dirichlet. The boundary conditions may or may not be satisfied at $t = 0$ by the function provided in `DiffusionProblem`, you only need to enforce it for $t > 0$. For each of the 4 vertices $((0,0),(L_x,0),(0,L_y),(L_x,L_y))$, you can assume that they agree when writing `EulerDiffusionSolver` and you should make sure that they agree when writing your `DiffusionProblem`.

Consider the vectorization of an $(n_x + 1) \times (n_y + 1)$ matrix obtained by stacking the columns of the matrix on top of one another[2]. That is the entry at row r and column c of the matrix is the $((c-1)(n_x+1)+r)$th entry of the vectorization. Using the explicit Euler scheme, `EulerDiffusionSolver` computes a vectorized approximation $\tilde{u}_k \approx u(kL_t/n_t)$ using the following affine discrete system:

$$\tilde{u}_{k+1} = A\tilde{u}_k + b \tag{1}$$

where A is a `SparseMatrix` and $\tilde{u}_k$ and b are `Vectors`.

The matrix A and vector b must be computed in the constructor of the `EulerDiffusionSolver` class. You have to implement the method `solve(sol)` of `EulerDiffusionSolver` that solve the problem using (1) and A and b computed in the constructor.

Investigate the behavior of your algorithm on `MyDiffusionProblem` for different values of n_t, n_x, n_y, and n.

Instructions

1. **Fraud:** As always for this course, you must do all the writing (report, codes) individually. Never share your production. However, you are allowed, and even encouraged, to exchange ideas on how to address the assignment.
2. **Plagiarism:** As always, you must cite all your sources.

[1] see Section 7.4 of Tveito et al. for a discussion of the explicit Euler scheme for the diffusion problem in one dimension (http://link.springer.com/book/10.1007%2F978-3-642-11299-7).

[2] See https://en.wikipedia.org/wiki/Vectorization_(mathematics).

3. **Submission:** Using the Moodle assignment activity, submit your report in a file called `Report_Projet 2_FirstName LastName.pdf`. The report should be short (maximum 4 pages) and should include

   - the result of a diffusion problem of your choice at different time steps;
   - an experimental evaluation of the rate of convergence to the solution w.r.t. n_t, n_x and n_y.

   On Inginious[3], submit your files `main.cpp`, `DiffusionProblem.cpp`, `EulerDiffusionSolver.hpp`, `EulerDiffusionSolver.cpp`, `SparseMatrix.hpp`, `SparseMatrix.cpp`, `SparseVector.hpp` and `SparseVector.cpp` containing the implementation of the corresponding classes.

   You are allowed to make as many submissions as you need, only the last submission will be taken into account. You are advised to verify that your submission passes the tests in Inginious early before the deadline. On Inginious, all the files available on Moodle are compiled together with the classes submitted and the tests included on `test.cpp` are executed.

   Note that, even if submitting the code on Inginious is mandatory, the Inginious automatic grading has no influence to the final grading. As mentionned, the tests on Inginious are exactly those included in `test.cpp`. Since these tests are minimalist, passing them is a necessary but not sufficient condition for having a correct code. You are strongly advised to add more tests to `test.cpp` to verify the correctness of your implementation.

4. **Language:** Reports in French are accepted without penalty. However, English is strongly encouraged. The quality of the English will not impact the grade, provided that the text is intelligible.

Plotting your simulation

You can use the tool of your choice to plot the results. Note that `main.cpp` writes your solution of your diffusion problem in a `sol.txt` file to facilitate this process. We also furnish a Julia code (`plot_sol.jl`) that plot the solution for you based on `sol.txt`.

These two files are there to help you; you can change them freely.

[3] https://inginious.info.ucl.ac.be/course/LINMA2710/