LINMA2710 – Scientific Computing – 2022-2023

Project 2 : Finite Volumes

Deadline: Monday 17th of April 2023 18:15

Statement

You will implement a finite volume solver for the diffusion problem

$$-\nabla \cdot (a(\mathbf{x})\nabla u) = f \quad \text{in } \Omega$$

with given scalar functions a and f, the domain Ω, and the scalar solution function u. Furthermore, the solution u should satisfy a Dirichlet condition on the boundary of the domain

$$u = g \quad \text{on } \partial\Omega.$$

Note that we look for a steady-state solution, we do not ask for the dynamics in time given an initial condition. Also, we will focus on two-dimensional domains.

This project is based upon the book [1], the Figures and equations in the statement come from this reference. Do not hesitate to consult it for further explanation.

The different parts of the codes are: the definition of the problem and the points defining the volumes, the decomposition into finite volumes from the given points through numerical geometry, writing a linear system of equations defining u, solving it, and obtaining a visualization of the solution.

Defining the problem

You have several lambda functions to define in `main.cpp`: a, f, g whose role is described above.

You must also give 2-dimensional points in `v_points`. These points define the finite volumes used to approximate the solution. Also, a part of the points should exactly be on the boundary of the domain. They will define the domain Ω.

An example is given in the code furnished.

Decomposing the domain into finite volumes

One of the strength of finite volumes methods over finite difference methods is the flexibility to choose a non-trivial discretization (not necessarily a grid) and the facility to handle non-rectangular domains. You will implement a solver that can leverage this flexibility.

However, this flexibility comes with a cost on the complexity of the code. Numerical geometry is necessary to partition the domain in volumes. Since this is not a numerical geometry course, the decomposition into finite volumes is implemented for you.

We will rely on the library `Fade2D`[1], look at the documentation and examples to install it in your workspace and understand the classes we use in the code.

Here is a fast explanation of how the decomposition works and what are its limits. Given a set of points with a part of them exactly on the border, we compute Voronoi cells for these points. The Voronoi cell of a point is a volume such that any element in the volume is closer to the point than from another one. Formally, given a set of points $\{x_i\} \in \Omega$ (the points in `v_points`) the Voronoi cell of each point is

$$V_i = \{x \in \Omega : \|x - x_i\| \leq \|x - x_j\|, \forall j \neq i\}.$$

This is represented on Figure 1.

To handle the boundary of the domain and its Dirichlet conditions, we ask to put points exactly on the boundary in such a way that no finite volume in the interior of the domain touch directly the boundary. In other words, there should be enough points on the boundary of the domain to ensure that no interior Voronoi cell goes out of the domain.

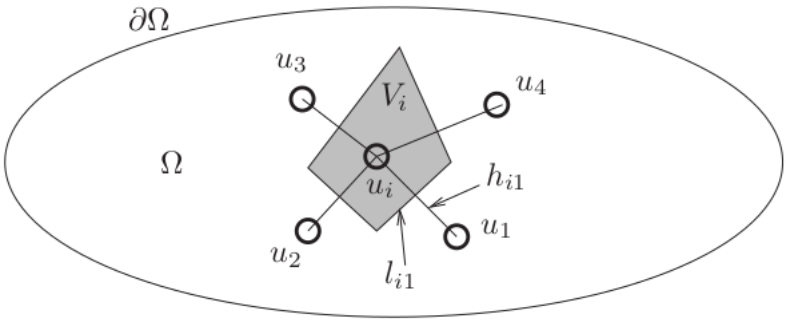

Figure 1: The Voronoi cell V_i of a point u_i [1].

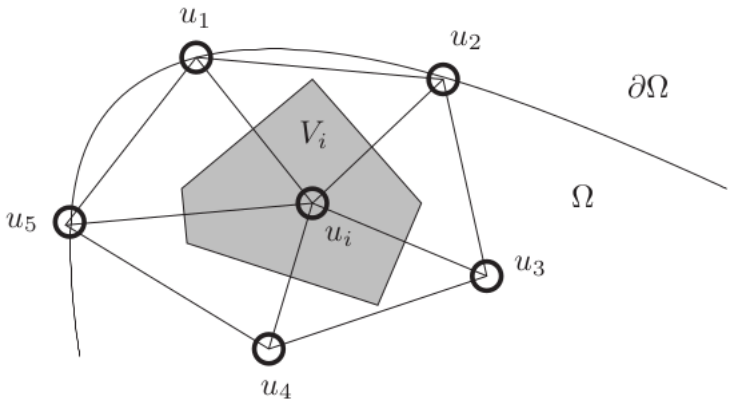

Figure 2: Voronoi cell close to the boundary with Dirichlet condition [1].

A Voronoi cell close to the border is represented on Figure 2.

We identified several requirements on the points in input to the geometrical part, note that this list may not be exhaustive :

- there should be enough points on the border in order to avoid having volumes (with center point on the interior of the domain) that go out of the domain;
- we only support convex domains;
- all the points should be distinct;
- there is a limit on the number of points that can be used with the `Fade2D` library under the free license.

Writing the linear system of equations

Your job is to implement the linear system of equations and solve it.

For example, the equation for the volume i on the interior of the domain is, following Figure 1,

$$a_{i3}\frac{u_3 - u_i}{h_{i3}}l_{i3} + a_{i4}\frac{u_4 - u_i}{h_{i4}}l_{i4} + a_{i1}\frac{u_1 - u_i}{h_{i1}}l_{i1} + a_{i2}\frac{u_2 - u_i}{h_{i2}}l_{i2} + a_{i5}\frac{u_5 - u_i}{hi5}l_{i5} = \text{Vol}(V_i)f_i.$$

Where:

- h_{ij} is the distance between x_i and x_j;
- a_{ij} is the value of $a(x)$ at the midpoint between x_i and x_j;

[1] https://www.geom.at/fade2d/html/

- l_{ij} is the length of the boundary between V_i and V_j.

To help you, the numerical geometry part already implements the computation of these quantities. See directly in the furnished code how to access the relevant information.

For the points on the border, they can be forced to satisfy the Dirichlet condition through a linear equation.

Solving the linear equation

To solve these equations, you will rely on an existing solver for sparse linear systems. We will use the `armadillo` library[2]. We refer to their documentation for more information.

Visualization of the solution

We furnish a `Python` code that reads the *sol.txt* file generated by the `C++` code and then visualizes it using `matplotlib`.

Installation and Makefile

You will need to install two libraries, `Fade2D` and `armadillo`. We refer to the websites of these packages and the search engine of your choice on how to install them.

We describe a solution for Ubuntu/WSL users, we did not test other OS.

For `Fade2D` you can follow the steps on this page[3] (download, change the Makefile, test on their examples) and then put your code in their example directory. Alternatively, you can copy-paste the *include_fade2d* directory and the `libfade2d.so` file corresponding to your OS version in your working directory with the Makefile we furnish.

For `armadillo`, you can install it by first installing the pre-required package then armadillo itself with `sudo apt-get install` (it does not give you the latest version of `armadillo` but it's sufficient for our use). If you use the Makefile we furnish we already put the `-larmadillo` option for the compiler, otherwise you should add it.

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.
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 3 pages) and should include

   - the result of a diffusion problem of your choice;
   - an experimental evaluation of the rate of convergence to the solution w.r.t. the maximal distance between two points that define the finite volumes.

   On Inginious[4], submit your files `main.cpp` with the definition of your problem, and `solver.cpp`, `solver.hpp` with the implementation of your solver.

   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 is correct, Inginious does not test anything.
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.

[2] https://arma.sourceforge.net/
[3] https://www.geom.at/fade-delaunay-triangulation/
[4] https://inginious.info.ucl.ac.be/course/LINMA2710/

References

[1] Martin J Gander and Felix Kwok. *Numerical analysis of partial differential equations using maple and MATLAB*. SIAM, 2018.