Development of a flexible, reusable and efficient transient advection-diffusion PDE solver in FEniCS

Rishi Jumani
M.S - Mathematics - II
New Mexico Tech

March 7th, 2018

Contents

1 Abstract

This document goes over the equations and the ideas behind developing a flexible, reusable and efficient advection diffusion PDE solver, using the finite element method in FEniCS. As described on the FEniCS website, *"The FEniCS Project is developed and maintained as a freely available, open-source project by a global community of scientists and software developers"*, led by Professors and Scientists at Simula Research Laboratory, Norway; CHALMERS University of Technology, Sweden and University of Cambridge, England. More details about the software can be found on the FEniCS website. The equations and the methods in this document are based on the concepts and examples in Langtangen and Logg [2016] and Langtangen and Logg [in prep.].

The solver has been coded with the help of Python classes. This allows the user to reuse the code to solve problems with different domains (2D or 3D), different combinations of boundary conditions (Dirichlet, Neumann or Robin) and different parameter values, thus helping avoid the need for frequent changes to the code. This approach goes beyond refactoring functions, and helps build a more flexible solver.

The solver developed can be used to solve any general advection-diffusion equation - change in concentration of a drug injected in the cerebro-spinal fluid, solute transport equation in groundwater modeling and even the heat conduction-convection equation.

2 Description of the PDE and its Variational form

The PDE that is to be solved is a variable coefficient, transient advection diffusion equation, with any combination of Dirichlet, Neumann or Robin boundary conditions. The spatial domain of the problem is Ω, whose boundary consists of Dirichlet, Neumann and Robin conditions $\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma_R$. The PDE is

$$\rho \frac{\partial u}{\partial t} = \nabla \cdot (\kappa \nabla u) - w \cdot \nabla u + f(\mathbf{x}, t) \text{ in } \Omega \times (0, T] \tag{2.1}$$

$$u(\mathbf{x}, 0) = IC \text{ on } \Omega \tag{2.2}$$

$$u = u_D(t) \text{ on } \Gamma_D \tag{2.3}$$

$$-\kappa \frac{\partial u}{\partial n} = g \text{ on } \Gamma_N \tag{2.4}$$

$$-\kappa \frac{\partial u}{\partial n} = r(u - U_s) \text{ on } \Gamma_R \tag{2.5}$$

where ρ is a general coefficient (for example - density in heat conduction-convection equation), u is the primary unknown, t stands for time, κ is the Diffusion coefficient, w is the advective velocity, f is the source term. The coefficients ρ and κ may vary in space, the functions f and g may vary in space and time, and the coefficients r and U_s may vary in time.

Following the approach in Langtangen and Logg [in prep.], we discretize the PDE in time, using the general θ rule. For an evolution equation, $\partial P/\partial t = Q(t)$ to get

$$\frac{P^{n+1} - P^n}{\Delta t} = \theta Q^{n+1} + (1-\theta) Q^n \tag{2.6}$$

where $\theta \in [0, 1]$ is a weighting factor. $\theta = 1$ corresponds to the Backward Euler scheme, $\theta = 1/2$ belongs to the Crank-Nicholson scheme, and $\theta = 0$ belongs to the Forward Euler scheme.

Using the θ-rule in the advection-diffusion PDE results in

$$\rho \frac{u^{n+1} - u^n}{\Delta t} = \theta \left(\nabla \cdot (\kappa \nabla u^{n+1}) - w \cdot \nabla u^{n+1} + f(\mathbf{x}, t_{n+1}) \right) + (1-\theta) \left(\nabla \cdot (\kappa \nabla u^n) - w \cdot \nabla u^n + f(\mathbf{x}, t_n) \right) \tag{2.7}$$

Using the regular Galerkin method to solve the initial boundary value problem, requires multiplying (2.7) by a test function $v \in V$, where V is a suitable function space, and integrating over Ω by using Green's theorem on the second order derivative term, to get:

$$\begin{aligned} \int_\Omega \rho \frac{u^{n+1} - u^n}{\Delta t} v &+ \theta \left(\kappa \nabla u^{n+1} \cdot \nabla v + w \cdot \nabla u^{n+1} v - f(\mathbf{x}, t_{n+1}) v \right) \\ &+ (1-\theta) \left(\kappa \nabla u^n \cdot \nabla v + w \cdot \nabla u^n v - f(\mathbf{x}, t_n) v \right) d\mathbf{x} \\ &- \int_{\Gamma_N \cup \Gamma_R} \left(\theta \kappa \frac{\partial u^{n+1}}{\partial n} v + (1-\theta) \kappa \frac{\partial u^n}{\partial n} v \right) ds = 0 \end{aligned} \tag{2.8}$$

Inserting the boundary conditions specified on Γ_N and Γ_R leads to

$$
\begin{aligned}
F(u;v) = \int_\Omega \rho \frac{u-u^n}{\Delta t} v + \theta(\kappa\nabla u\cdot\nabla v + w\cdot\nabla u v - f(\mathbf{x},t_{n+1})v) \\
+(1-\theta)\left(\kappa\nabla u^n\cdot\nabla v + w\cdot\nabla u^n v - f(\mathbf{x},t_n)v\right) d\mathbf{x} \\
+\int_{\Gamma_N} (\theta g(\mathbf{x},t_{n+1})v + (1-\theta)g(\mathbf{x},t_n)v)\, ds \\
+\int_{\Gamma_R} (\theta r(u-U_s(t_{n+1}))v + (1-\theta)r(u-U_s(t_n))v)\, ds
\end{aligned}
\tag{2.9}
$$

where u_{n+1} has been replaced by the variable u to match the code. The variational formulation is $F(u;v) = 0\ \forall v \in V$. FEniCS does not require the separation of the bilinear and linear terms in $F(u;v)$, as it has the functions lhs and rhs to do this in the code.

3 Comments on the Code

The superclass and the solver class used to solve the variational form are included in the script AdvDiffSolver.py. The abstract problem class (superclass) in the code is called AdvDiffSuper. Different problems can be implemented as subclasses of AdvDiffSuper. The solve function applies to a general advection-diffusion PDE, and can be inherited, whereas the methods that pertain to a specific problem must be implemented in the subclass. The solver used is programmed as a AdvDiffSolver class

In the solver class, AdvDiffSolver, the input arguments and the information that is required by the solver are obtained through methods calls to an instance of the super class. The solver class has a function to assign the initial condition, a function to set up data structures for the variational formulation and to assign the boundary conditions, a step function to advance the solution in time and a solve function to run the time loop.

The finite element function space, V used consists of the standard Lagrange family of elements, of any specified degree over a given mesh. A finite element of degree 1 would be a linear triangle, with nodes at the three vertices. FEniCS also allows the use of higher degree elements and discontinuous elements.

FEniCS comes packaged with several numerical linear algebra libraries, including PETSc, which is the default choice. The solver used in this code is the default Sparse LU decomposition solver. FEniCS also allows the use of many iterative solvers, along with suitable preconditioners. The linear system for the advection-diffusion PDE can also be solved with a Krylov subspace iterative solver, GMRES, with an incomplete LU (ILU) pre-conditioner, by choosing the linear_solver='Krylov' option when calling the solver.

3.1 Examples

Different combinations of meshes and boundary conditions can be used to test the code.

3.1.1 Method of Manufactured Solutions

The first example has been implemented in the script AdvDiffProblem_1.py. The method of manufactured solutions is used to test the code. Consider the solution $u = 1 + x^2 + \alpha y^2 + \beta t$ on a uniform mesh (square or cube) with a Dirichlet boundary. The source term is then $f = \beta - 2 - 2\alpha + 2xw_x + 2\alpha y w_y$, where w_x and w_y are the advective velocities in the x and y directions.

3.1.2 Dirichlet and Neumann boundary condtions + Plotting

The next example, implemented in AdvDiffProblem_2.py tests the code on a unit square domain with the following Dirichlet conditions: $u = 1$ at $x = 0$ and $u = 0$ at $x = 1$, and homogeneous Neumann conditions on the top and bottom boundary. The code solves the following simplified test problem mentioned in Langtangen and Logg [in prep.]:

$$\frac{\partial u}{\partial t} = \nabla^2 u \text{ on } \Omega = (0,1)\times(0,1)\times(0,T]) \tag{3.1}$$

$$u(\mathbf{x},0) = 0 \text{ on } \Omega \tag{3.2}$$

$$u = 1 \text{ on } \Gamma_{D1} \tag{3.3}$$

$$u = 0 \text{ on } \Gamma_{D2} \tag{3.4}$$

$$\frac{\partial u}{\partial n} = 0 \text{ on } \Gamma_N \tag{3.5}$$

The function mark_boundary marks the boundaries of the domain, so that the subclass that implements the specific details of the problem can call the function to impose the type of boundary conditions specified in the problem. This function makes it easy to define different boundary conditions for 2D or 3D domains.

This problem is an example of the development of a thermal boundary layer. The problem is solved using the backward Euler scheme. The code plots the evolution of the solution at $y = 0.5$ to match the plot in Langtangen and Logg [in prep.], as shown in Figure 1. It is also possible to save the plot frames of the solution at different times and create an animation.

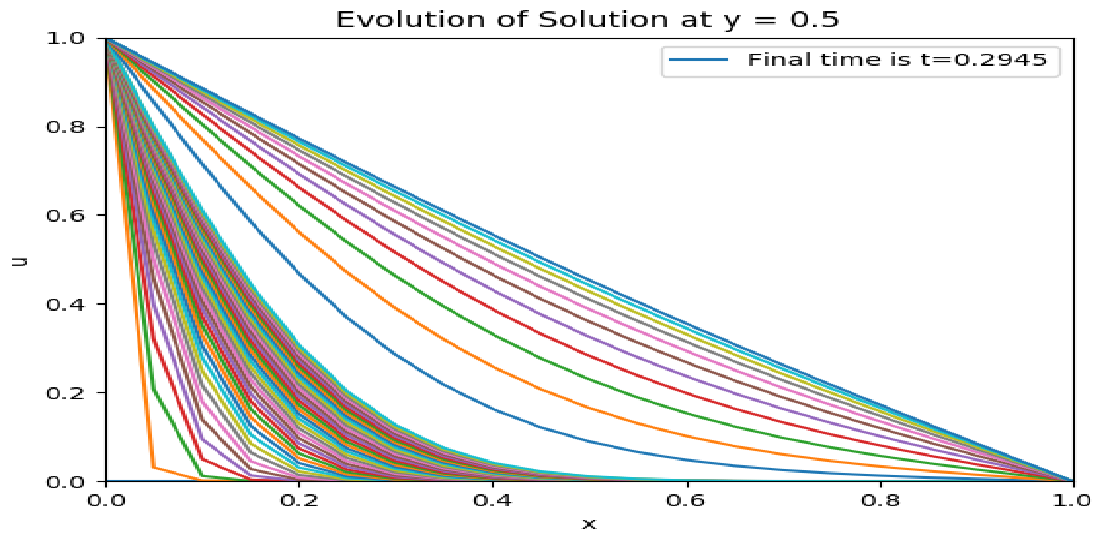

Figure 1: Development of thermal boundary layer

4 Future Work

The following examples will be added over time.
Include Post processing class (cbcpost)

4.0.1 Variable Diffusion coefficient values

4.0.2 Implement a convergence analysis module

4.0.3 Scaling a PDE example;

4.0.4 Use the DG method to solve

References

Langtangen, Hans Petter and Logg, Anders. Solving PDEs in minutes -the FEniCS tutorial Volume I. *Springer*, 2016.

Langtangen, Hans Petter and Logg, Anders. Writing more advanced FEniCS programs - the FEniCS tutorial Volume II. *Springer*, in prep.