INMA2710 : Scientific Computing

Project 2

Author :
Renaud Dufays – 5919-12-00

Teacher :
P.-A. Absil

Teaching Assistant :
B. Legat

May 15, 2017

1 Introduction

In this project, we apply a finite-difference method for the diffusion equation

$$\rho(x,y)c_v(x,y)\frac{\partial u}{\partial t} = k(x,y)\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right] + f(t,x,y). \tag{1}$$

We make use of second order centered finite-difference for the discretization of the Laplacian, and we use the Explicit Euler scheme for the time integration. The scheme is thus of order $\mathcal{O}\left(\Delta t, \Delta x^2, \Delta y^2\right)$, where $\Delta t = L_t/n_t$, $\Delta x = L_x/n_x$ and $\Delta y = L_y/n_y$. With the choices stated above, the discrete version of (1) is :

$$\rho_{i,j}c_{v_{i,j}}\frac{u^{l+1}_{i,j} - u^l_{i,j}}{\Delta t} = k_{i,j}\left[\frac{u^l_{i+1,j} - 2u^l_{i,j} + u^l_{i-1,j}}{\Delta x^2} + \frac{u^l_{i,j+1} - 2u^l_{i,j} + u^l_{i,j-1}}{\Delta y^2}\right] + f_{l,i,j}, \tag{2}$$

where subscripts "i,j" and superscript "l" indicate that the quantity is evaluated at position $(i\Delta x, j\Delta y)$ and time $l\Delta t$. Since we use an explicit scheme for the time integration, some stability constraints are to be fulfilled. This is the topic of section 2. Then a short user documentation of the code is provided in section 3, along with the justification of several design decisions. A "sanity check" of the computer implementation is performed on two simple cases in section 4 and finally, the result of a particular diffusion problem at different time steps is shown in section 5, along with a discussion on the behavior of the algorithm for different values of n_t, n_x and n_y.

2 Preliminary: stability analysis

We proceed here with the so-called *Von Neumann* stability analysis. The source term $f(t,x,y)$ has no impact on the stability of the method so we can simply forget it for the sake of this section. We define $\epsilon := k/\rho c_v$, and restrict ourselves to the case where ϵ does not vary in space. Note that $\epsilon \geq 0$ as we want the parameters to be physically significant. We only present the main steps of the analysis here, but the interested reader may refer to [Win16].

The idea is to consider a class of special solutions ("Ansatz") of the form

$$U^{m,n}(t,x,y) = A_{m,n}(t)e^{i(mx+ny)} \tag{3}$$

(i is the imaginary unit $\sqrt{-1}$ in this context) which are valid in the case of periodic boundary conditions. By linearity and the superposition principle, it is sufficient to consider one single mode for the analysis. Introducing it into the diffusion equation (without source term) yields

$$A_{m,n}(t) = A_{m,n}(0)e^{-\epsilon(m^2+n^2)t}, \tag{4}$$

and thus

$$U^{m,n}(t,x,y) = A_{m,n}(0)e^{-\epsilon(m^2+n^2)t}e^{i(mx+ny)}. \tag{5}$$

For the next, we drop the "m,n" superscript to make the notations less cumbersome. Now, introducing this into the discrete diffusion equation (2) where we keep the time derivative term continuous yields (after some calculations)

$$\frac{\mathrm{d}\,U_{i,j}(t)}{\mathrm{dt}} = \underbrace{-4\epsilon\left(\frac{\sin^2(m\Delta x/2)}{\Delta x^2} + \frac{\sin^2(n\Delta y/2)}{\Delta y^2}\right)}_{:=\lambda_{m,n}} U_{i,j}. \tag{6}$$

Note that no assumption has been made on the time integration scheme until now. We could thus analyze the stability of any time integration scheme for the diffusion equation using $\lambda_{m,n}$. For the Explicit Euler scheme, we get

$$U_{i,j}^{l+1} = U_{i,j}^{l}\left[1 + \lambda_{m,n}\Delta t\right], \tag{7}$$

or equivalently

$$U_{i,j}^{l} = \left[1 + \lambda_{m,n}\Delta t\right]^l U_{i,j}^{0}. \tag{8}$$

The Explicit Euler scheme is thus stable if $|1 + \lambda_{m,n}\Delta t| \leq 1$. This condition can be rewritten as

$$-2 \leq \lambda_{m,n}\Delta t \leq 0. \tag{9}$$

The second inequality is always guaranteed. The critical mode for the first inequality is when $m\Delta x = n\Delta y = \pi$. Hence, the scheme is stable if

$$\alpha := \epsilon\Delta t\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right) \leq \frac{1}{2}. \tag{10}$$

Hence, for given values of Δx and Δy, the time step Δt must be small enough. For example, if we reduce both Δx and Δy by a factor 10, we must reduce Δt by a factor 100 ! Finally, it is important to notice that this condition is more strict than in the 1-D case : if $\Delta x = \Delta y =: h$, we must have $\epsilon\Delta t/h^2 \leq 1/4$ whereas the 1-D condition is $\epsilon\Delta t/h^2 \leq 1/2$.

3 User documentation

Two concrete classes are implemented. First, the class `MyDiffusionProblem` derives from `AbstractDiffusionProblem` and implements a particular diffusion problem that we detail in the next few lines: The initial condition is shown on figure 1. The boundary conditions are $u(t,0,y) = \sin[2\pi(y+t+1/4)]$, $u(t,x,0) = \sin[2\pi(x+t+1/4)]$, $\partial u/\partial x|_{x=1} = 0$ and $\partial u/\partial y|_{y=1} = 0$. The physical coefficients are $k = 1$ [J/m K s], $\rho = 10$ [kg/m^3] and $c_v = 1$ [J/kg K]. The spatial domain is $[0,1]\times[0,1]$ and the final integration time is $L_t = 1$. The sink term is defined as

$$f(t,x,y) = \begin{cases} -100 & \text{if } |x-0.3| \leq 0.1 \text{ and } |y-0.3| \leq 0.1 \\ 0 & \text{otherwise.} \end{cases}$$

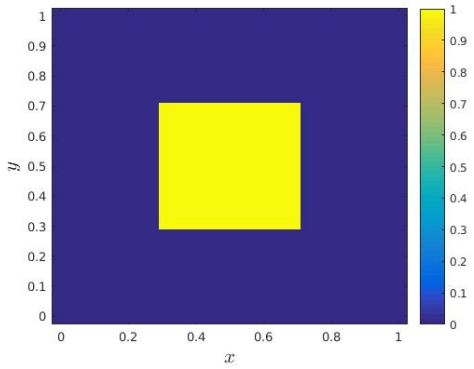

Figure 1 – Initial condition of `MyDiffusionProblem`.

The second class that we have implemented is `EulerDiffusionSolver` and derives from `AbstractDiffusionSolver`. It solves the 2-D diffusion equation using the Explicit Euler scheme. The solver implements thus an FTCS (Forward-Time Central-Space) scheme. This class is made of one single method, `solve`, which takes as argument a `Tensor` of dimensions (n_t+1, n_x+1, n_y+1) and a diffusion problem deriving from `AbstractDiffusionProblem`, for example an instance of `MyDiffusionProblem`. It also displays on the console the maximal value of $\alpha := \frac{k}{\rho c_v}\Delta t\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right)$ over the domain, which is an indicator of the stability of the scheme for the discretization used, as we have seen in section 2.

Finally, an interesting remark has to be made about the choice of using time as first dimension of the tensor : it is indeed a smart choice since for every time iteration l, some operations must be performed on every entry of the tensor corresponding to a point of the domain at time $l\Delta t$. In term of performance, we want thus the entries corresponding to neighboring points of the domain to be stored at contiguous addresses in memory for every discrete time. This is only true if we choose time as the first dimension of the tensor.

4 Verification of the computer implementation

In order to verify our computer implementation of the scheme, we construct two simple problems to which we know the solution in advance, and check if our program handles them correctly. The term "verify" must not be misunderstood : we do not prove correctness; those tests rather constitutes "sanity-checks". We briefly present them here.

The first case is when the solution is constant in space and time : choosing $C = 0.8$, the initial condition is $u(0,x,y) = C$ and we impose Dirichlet boundary conditions at the four borders of the domain: $u(t,0,y) = C$, $u(t,1,y) = C$, $u(t,x,0) = C$ and $u(t,x,1) = C$. The source term is zero everywhere. The solution is then $u(t,x,y) = C$. We can indeed verify that our program handles that case properly.

The second test problem that we construct basically amounts to the 1-D diffusion of a Jump that has been covered during the lectures. The initial condition is

$$u(0,x,y) = \begin{cases} 0 & \text{if } x \leq 0.5 \\ 1 & \text{if } x > 0.5, \end{cases}$$

and we impose the boundary conditions $u(t,0,y) = 0$, $u(t,1,y) = 1$, $\partial u/\partial y|_{y=0} = 0$ and $\partial u/\partial y|_{y=1} = 0$. The physical parameters are $k = 1$ [J/m K s], $\rho = 10$ [kg/m^3] and $c_v = 1$ [J/kg K]. With that configuration, $u(t,x,y)$ is in fact independent of y and any cut at a constant y corresponds to the solution to the 1-D case. The solution to this test problem with $n_t = 160$, $n_x = 20$ and $n_y = 20$ is shown at different times in figure 2. For each time, a 2-dimensional view of the solution illustrating the correspondence with the 1-D case is shown. As expected, the solution converges to a plane (a straight line in the 2-D view).

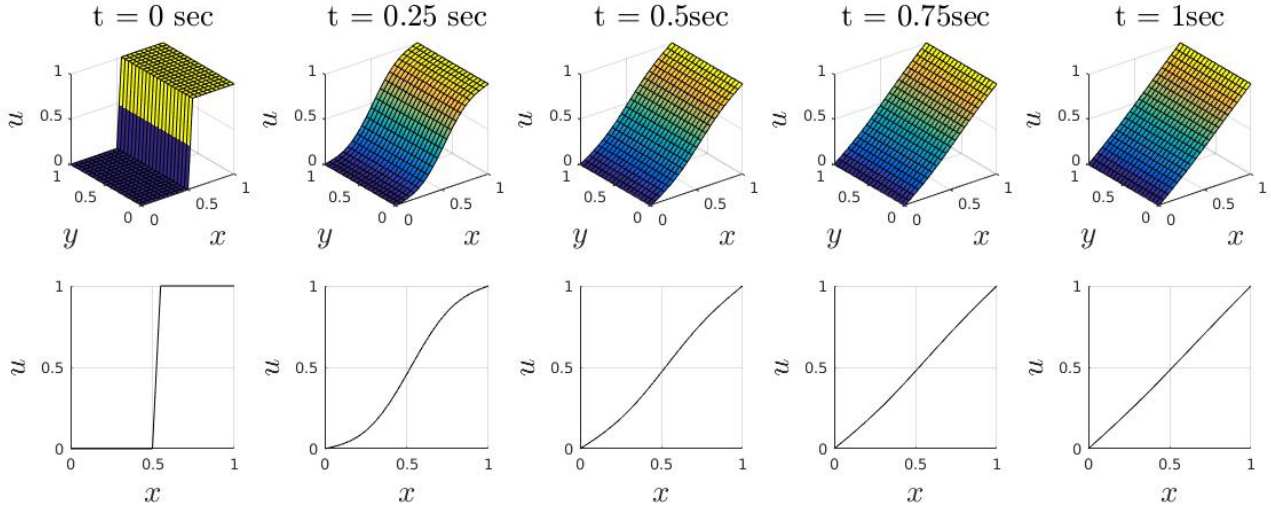

Figure 2 – Test problem : 2-D jump.

5 Results on a particular diffusion problem

Let us now investigate the behavior of our algorithm on `MyDiffusionProblem`, which has been described in section 3. Figure 3 shows the results at different time steps for $n_t = 160$, $n_x = 20$ and $n_y = 20$. For that discretization, $\alpha = 0.5$, which is exactly the limit of stability. By the Lax-Richtmyer equivalence theorem, the scheme is thus also convergent.[1] For the next, we consider this case as the reference case for comparison, as we will investigate the behavior of the algorithm for different values of n_t, n_x and n_y.

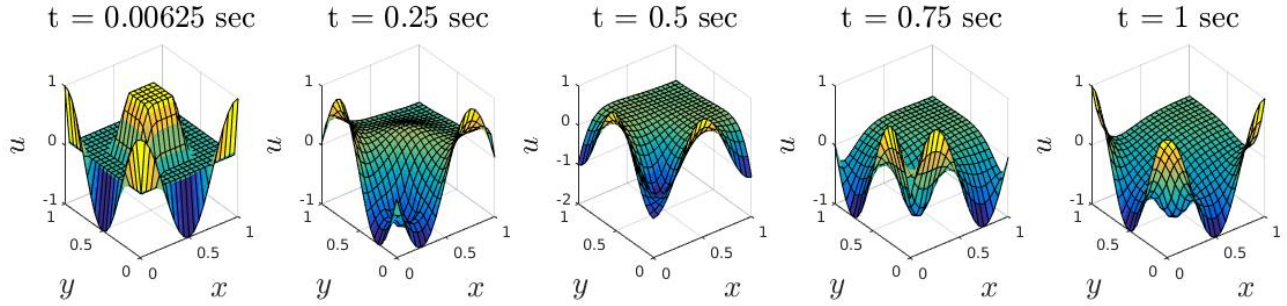

Figure 3 – Evolution of the solution to `MyDiffusionProblem`, with $n_t = 160$, $n_x = 20$ and $n_y = 20$ ($\alpha = 0.5$).

Now let us look at the behavior of the algorithm if we modify the discretization parameters. First, keeping α constant, we refine the space discretization by taking $n_x = n_y = 40$. Keeping $\alpha = 0.5$ implies that $n_t = 640$. The solution shown in 4 looks quite similar, excepted that it is now more precise. This is in agreement with the fact that the error of the scheme is $\mathcal{O}\left(\Delta t, \Delta x^2, \Delta y^2\right)$, and we are still within the limit of stability.

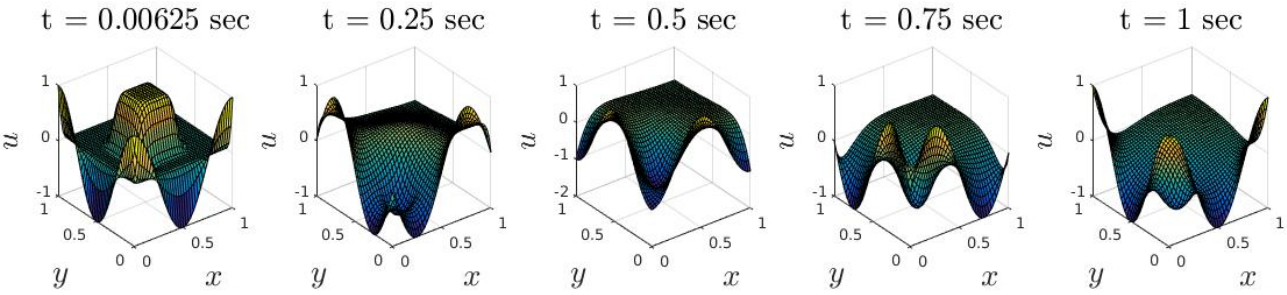

Figure 4 – Evolution of the solution to `MyDiffusionProblem`, with $n_t = 640$, $n_x = 40$ and $n_y = 40$ ($\alpha = 0.5$).

Now, what happens if we increase n_x and n_y but instead of keeping α constant, we maintain $n_t = 160$? Let us choose $n_x = n_y = 21$: we add one point in each space direction with respect to the reference case. In that case, $\alpha = 0.55125$. The evolution of the solution is shown for different times in figure 5. We observe that the solution blows up : it is unstable, in agreement with the result of section 2. The behavior of the solution if we only increase n_x (or n_y) without increasing n_y (or n_x) is qualitatively similar and is not shown here.

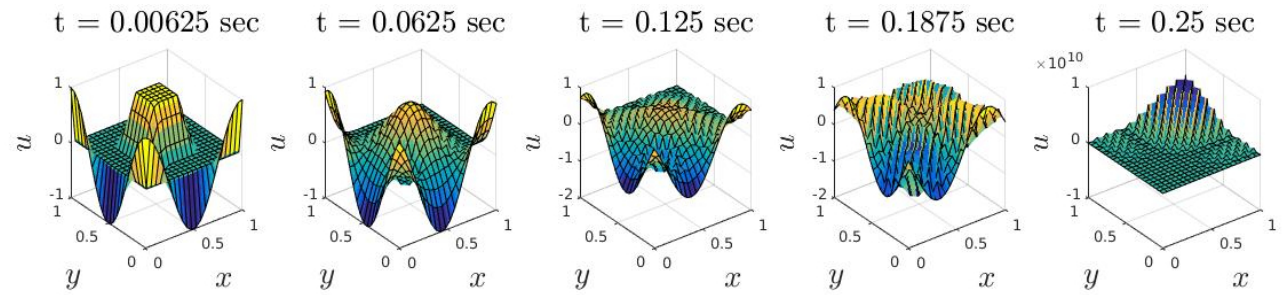

Figure 5 – Illustration of the instability of the Explicit Euler scheme ($\alpha = 0.55125$).

[1] It is easy to check that the finite difference scheme is consistent, and the initial value problem is linear and well-posed.

If instead, we maintain $n_x = n_y = 20$ and increase n_t to 320, then $\alpha = 0.25$. In that case, the scheme is obviously stable, and we have halved the error due to the time integration, as it is in $\mathcal{O}(\Delta t)$.[2] The result obtained seems to the naked eye very similar to the reference case : it is thus not shown here.

As a conclusion, increasing n_x and/or n_y enhances the precision in space, and increasing n_t enhances the precision in time. But those parameters cannot be tuned independently, since the FTCS scheme is *conditionally stable* : one must always ensure that the stability criterion is fulfilled. This criterion implies that if you want to increase the spatial precision (thus n_x and/or n_y), you have to increase the precision in time (thus n_t) accordingly. There exists *unconditionally stable* schemes for the time integration, such as the Backward-Euler scheme or the Cranck-Nicholson scheme. Those schemes are a bit trickier to implement, but they have the big advantage that they do not introduce any restriction on the time step, while maintaining a similar computational cost.

References

[PFW12] Joe Pitt-Francis and Jonathan Whiteley. *Guide to Scientific Computing in C++*. Springer-Verlag, London, 2012.

[TLNC10] Aslak Tveito, Hans Petter Langtangen, Bjørn Frederik Nielsen, and Xing Cai. *Elements of Scientific Computing*. Springer-Verlag, Berlin Heidelberg, 2010.

[Win16] Gregoire Winckelmans. *LMECA2660 : Numerical methods in fluid dynamics. Lecture notes.* Université Catholique de Louvain, 2016.

[2] In practice the double-precision arithmetic introduce floating point errors. We neglect those errors here.