Solving PDEs with Backward Implicit Time Steps

Matthieu Gomez *

August 8, 2021

This package EconPDEs.jl introduces a fast and robust way to solve systems of PDEs + algebraic equations (i.e. DAEs) associated with economic models. It builds on the method presented in Achdou et al. (2016), but extends it to handle non-linearities.

Consider a PDE of the form

$$0 = f_0(x, V) + f_1(x, \partial_x V)\partial_x V + f_2(x, \partial_x V)\partial_{xx} V + \partial_t V \tag{1}$$

As in Achdou et al. (2016), I first discretize the state x on a grid and I approximate derivatives $\partial_x V_t$ and $\partial_{xx} V_t$ using finite difference approximations. First order derivatives are upwinded, which helps with convergence and ensures that boundary counditions are satisfied.

Given V_{t+1}, I solve for V_t using a backward implicit time step:

$$0 = f_0(x, V_t) + f_1(x, \partial_x V_t)\partial_x V_t + f_2(x, \partial_x V_t)\partial_{xx} V_t + \frac{1}{\Delta}(V_{t+1} - V_t) \tag{2}$$

This non-linear equation is solved using a Newton-Raphson method. Julia packages ForwardDiff and SparseDiffTools are used to compute the (sparse) Jacobian of (2) automatically.

Difference with Achdou et al. (2016) Achdou et al. (2016) propose to solve the PDE (1) using a backward *semi*-implicit time steps of the form:

$$0 = f_0(x, V_t) + f_1(x, \partial_x V_{t+1})\partial_x V_t + f_2(x, \partial_x V_{t+1})\partial_{xx} V_t + \frac{1}{\Delta}(V_{t+1} - V_t) \tag{3}$$

Compared to (2), each time step corresponds to a *linear* equation in V_t. This makes it easier to solve each step, since one only needs to invert a matrix. However, this simplification makes the method less robust: one can show that the associated scheme is typically non monotonous, unless f_1 and f_2 do not depend on the value function.

*I thank Valentin Haddad, Ben Moll, and Dejanir Silva for useful discussions.

Stationary Solution In most cases, one is only interested in the stationary solution of the PDE (1), i.e.,

$$0 = f_0(x, V) + f_1(x, \partial_x V)\partial_x V + f_2(x, \partial_x V)\partial_{xx} V \tag{4}$$

In this case, I use the same method, but I adapt the time step Δ over time. More precisely, if the backward time step (2) is successful (i.e., the Newton-Raphson method converges), Δ is increased; otherwise, it is decreased. This method ensures converges since, the Newton-Raphson method always converges for Δ small enough. Yet, it does not sacrifice speed: as $\Delta \to \infty$, the method becomes equivalent to a non-linear solver for the PDE, which ensures quadratic convergence around the solution.

The idea of adapting Δ comes from the Pseudo-Transient Continuation method used in the fluid dynamics literature. Formal conditions for the convergence of the algorithm are given in Kelley and Keyes (1998).

Applications Empirically, I find the method to be fast and robust — the examples folder shows that the algorithm solves a wide range of asset pricing models.

References

Achdou, Yves, Jiequn Han, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll, "Heterogeneous Agent Models in Continuous Time," 2016. Working Paper.

Kelley, Carl Timothy and David E Keyes, "Convergence analysis of pseudo-transient continuation," *SIAM Journal on Numerical Analysis*, 1998, *35* (2), 508–523.