Nonlinear Conservation Laws
a finite element approach

Finn Joel Bjervig*

1TD050 - Advanced Numerical Methods,
Department of Information Technology,
Uppsala university, Sweden

August 31, 2022

*Electronic address: `jobj8920@student.uu.se`

Problem description

consider the KPP (Kolmogorov–Petrovsky–Piskunov) rotating wave problem

$$\begin{aligned} \partial_t u + \nabla \cdot f(u) = 0 \qquad & (\mathbf{x}, t) \in \Omega \times (0, T] \\ u(\mathbf{x}, t) = u_0(\mathbf{x}) \qquad & \qquad \mathbf{x} \in \Omega \end{aligned}$$

The initial condition is defined as

$$u_0(\mathbf{x}) = \begin{cases} \frac{14\pi}{4} & \text{if } \sqrt{x^2 + y^2} \\ \frac{\pi}{4} & \text{otherwise} \end{cases}$$

A natural first step is to consider the GFEM formulation, for which the weak formulation is first needed. It reads:
Find $u \in H^1$ such that

$$(\partial_t u, v) + (f'(u)\nabla u, v) = 0 \qquad \forall v \in H_0^1 \tag{1}$$

The advection term has been expanded. The solution u can be approximated by $u_h \in V_h \subset H_1$, $dim(v_h) < \infty$. This culminates into the GFEM formulation which reads:
Find $u_h \in V_h$ such that

$$(\partial_t u_h, v) + (f'(u_h)\nabla u_h, v) = 0 \qquad \forall v \in v_{h,0} \subset H_0^1 \tag{2}$$

$$\begin{aligned} V_{h,0} = \{v(x,t) : v(x,t) \in C^0(\tau_h), \forall t \in (0,T], v|_K \in P_1(K),\ \forall K \in \tau_h \\ v = 0 \text{ on } \partial\Omega\} \end{aligned} \tag{3}$$

K are the elements on the mesh τ that approximates the region Ω

$\tau_h = \{K\}$, K -elements, $h_k = diam(K)$ - meshsize, $\{N_h\}$- set of all nodes on τ_h

$\{N_j\}$ denotes the set of all internal nodes and $\{N_b\}$ the set of all boundary nodes. Since $u_h \in V_h$, $\exists\{U\}_{N_j \in N_h}$ such that

$$u_h(x,t) = \sum_{N_j \in N_h} U_j(t)\varphi_j(x) \tag{4}$$

where $\varphi_j(x)$ are the appropriate hat functions.

The GFEM formulation now reads: Find $U_j \in N_j$ such that

$$\sum_{N_j \in N_h} (\partial_t U_j \varphi_j, \varphi_i) + \sum_{N_j \in N_h} (f'(U_j) \nabla \varphi_j U_j, \varphi_i) = 0 \qquad \forall \varphi_i \in v_{h,0} \tag{5}$$

recognize that this yields a system of equations consisting of the Mass matrix M and the Convection matrix $C(U_j)$

$$M \partial \mathbf{U} + C(\mathbf{U})\mathbf{U} = 0 \tag{6}$$

where $C(\mathbf{U})_{ij} = f'(\mathbf{U}) \nabla \varphi_j \varphi_i$. Crank Nicholson discretization approximates the time derivative and we get

$$M \frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} + \frac{1}{2} \left(C(\mathbf{U}^{n+1})\mathbf{U}^{n+1} + C(\mathbf{U}^n)\mathbf{U}^n \right) = 0 \tag{7}$$

solve for $\mathbf{U}^{n+1}$

$$\left(\frac{M}{\Delta t} + \frac{C(\mathbf{U}^{n+1})}{2} \right) \mathbf{U}^{n+1} = \left(\frac{M}{\Delta t} - \frac{C(\mathbf{U}^n)}{2} \right) \mathbf{U}^n \tag{8}$$

It is apparent that the nonlinear term on the left hand side $A(\mathbf{U}^{n+1})\mathbf{U}^{n+1}$ needs to be linearized. There are many ways to do this, and for simplicity (not for accuracy) sake, Piccard's iteration will be utilized. A better alternative would be to use an optimizing algorithm out of the large family of newton based methods.

Piccard iteration works by linearizing the term $A(\mathbf{U})\mathbf{U}$ by shifting one of the variables index as $A(\mathbf{U}^k)\mathbf{U}^{k+1}$ in order to solve for the next solution $\mathbf{U}^{k+1}$ until sufficient convergence is reached $\left\| \mathbf{U}^{k+1} - \mathbf{U}^k \right\| < TOL$ for some tolerance TOL. The algorithm for the KPP problem is

Algorithm 1: Piccard iteration for KPP problem

Set a tolerance TOL $= 10^{-6}$, error $= \infty$ and an initial solution $\mathbf{U}^k = \mathbf{U}^0$

while $error < TOL$ **do**

  $bx = \partial_x f(\mathbf{U}^k)$

  $by = \partial_y f(\mathbf{U}^k)$

  Assemble convection matrix $C(\mathbf{U}^k, bx, by)$

  $\mathbf{U}^{k+1} = \left(\frac{M}{\Delta t} + \frac{C(\mathbf{U}^k)}{2}\right) / \left(\frac{M}{\Delta t} - \frac{C(\mathbf{U}^k)}{2}\right) \mathbf{U}^k$

  $\mathbf{U}^k = \mathbf{U}^{k+1}$

  $error = \left\|\mathbf{U}^{k+1} - \mathbf{U}^k\right\|$

end

$\mathbf{U}^{n+1} = \mathbf{U}^k$

Implementation