Verification and Validation in Scientific Computing - Homework 3

Steven Roberts

March 18, 2020

1 Problem overview

The Allen–Cahn PDE is a reaction-diffusion PDE governed by equation

$$\begin{aligned} \frac{\partial u(t,x,y)}{\partial t} &= \alpha \nabla^2 u(t,x,y) + \beta \left(u(t,x,y) - u(t,x,y)^3\right) + s(t,x,y), \\ t,x,y &\in [0,1] \end{aligned} \tag{1}$$

with zero Neumann boundary conditions on all four boundary edges. The parameters α and β are both taken to be 1, and $s(t,x,y)$ is a source term.

This PDE is discretized in space on an $N \times N$, uniform grid using second order finite differences. The spatial discretization parameter is denoted as Δx. In time, a second order Rosenbrock method is used with a fixed timestep of Δt.

2 Constructing a manufactured solution

In order to have an exact, analytic solution of eq. (1) for the numerical tests, I elected to use the method of manufactured solutions. I chose the exact solution as

$$u_{\text{ex}}(t,x,y) = e^{-t}\left(\cos(2\pi x) + \cos^2(4\pi y)\right),$$

which is already compatible with the boundary conditions. Using Mathematica, I found the resulting source term to be

$$s(t,x,y) = e^{-3t}\left(e^{2t}\left(\left(4\pi^2 - 2\right)\cos(2\pi x) + \left(32\pi^2 - 1\right)\cos(8\pi y) - 1\right) + \left(\cos(2\pi x) + \cos^2(4\pi y)\right)^3\right).$$

3 Results for experiments

In the numerical experiments, I used three different norms to measure the error between the exact PDE solution and the numeric solution:

$$\|x\|_1 = \frac{1}{N^2}\sum_{i=1}^{N}|x_i|$$
$$\|x\|_2 = \frac{1}{N}\sqrt{\sum_{i=1}^{N}x_i^2}$$
$$\|x\|_\infty = \max_i |x_i|.$$

Figure 1 plots these errors as Δx and Δt are simultaneously refined. We can see all measures of the error converge with a slope close to that of the second order reference slope. This is more clear in fig. 2, as the numerical order is plotted. It is a bit higher than 2 at around 2.2, but that is still consistent with the theoretical order.

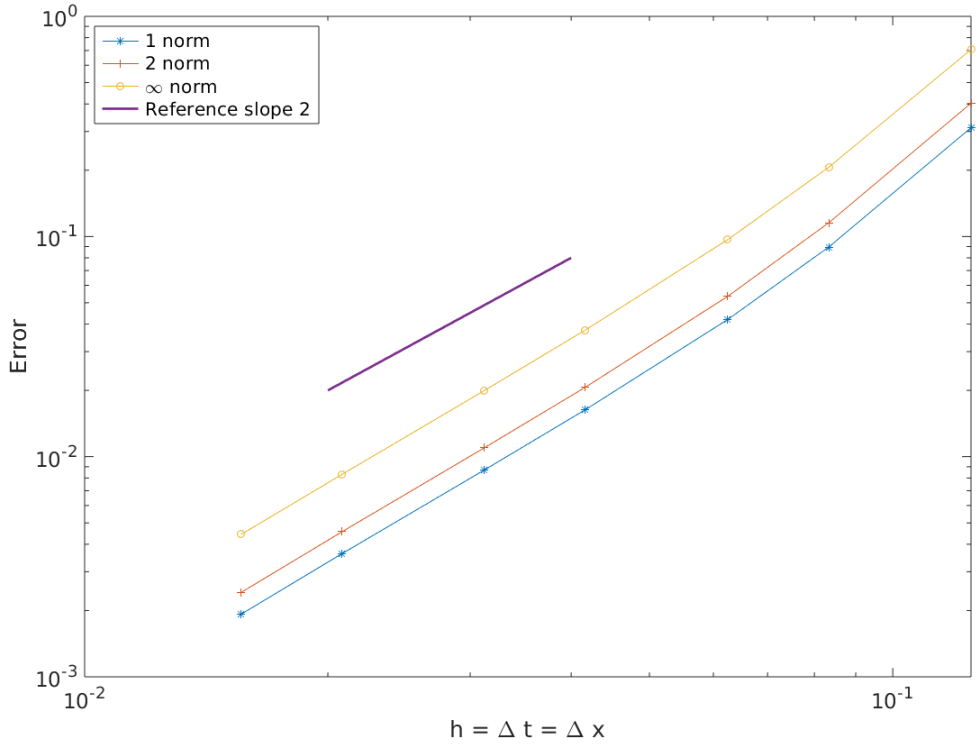

Figure 1: Error as the spatial and temporal meshes are simultaneously refined.

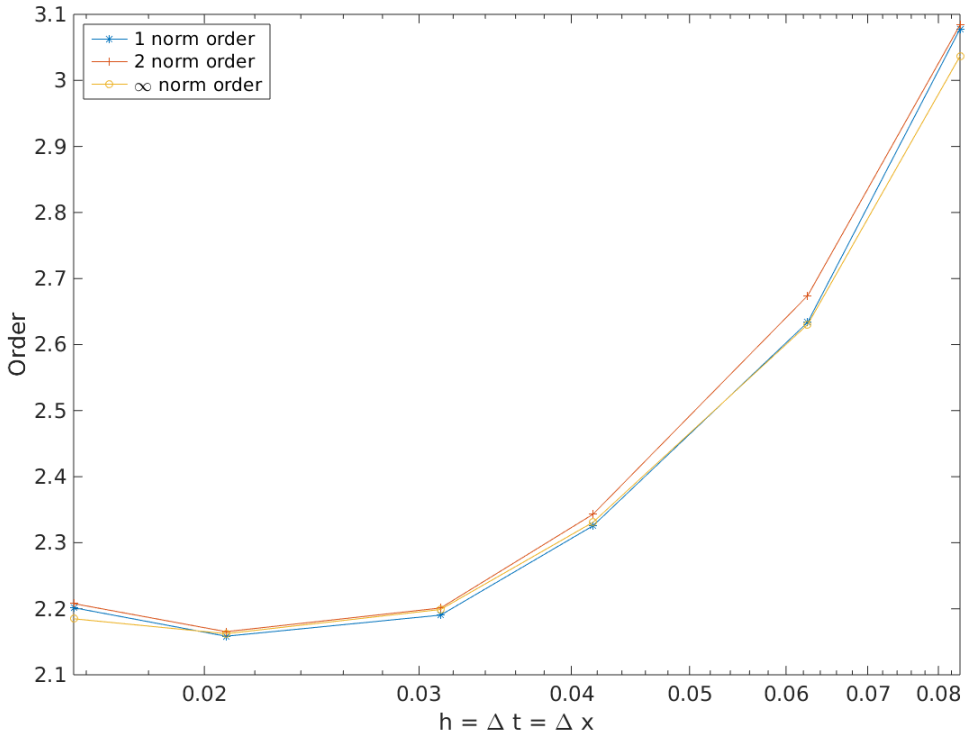

Figure 2: Convergence order as the spatial and temporal meshes are simultaneously refined.

Unfortunately, I was unable to estimate round-off errors as the code is not set up to run using single precision arithmetic. I would speculate the biggest source of these errors would come from the direct linear solves involving the Jacobian found in each Rosenbrock stage. Considering the errors in fig. 1 are on the order of 10^{-2} and the solution values are on the order of 10^{0}, I would expect round-off errors to be negligible in the experiments presented. Only for significantly finer spatial and temporal meshes would I expect round-off errors to dominate.

4 Code coverage

Using uniform timesteps is the most sensible way to perform the convergence test, but it means the adaptive timestepping part of the integrator in the model is not tested. Moreover, we only considered the case when $\alpha = \beta = 1$. A more thorough and convincing test could include manufactured solutions for several parameter values. Otherwise, the manufactured solution test provides full verification of the Allen–Cahn code.

5 Experience using Git for project

The source code for this homework is available in the GitHub repository `https://github.com/Steven-Roberts/VVSC-Project`. I found this version control system simple and straightforward to use. Any time I got a major piece of the homework working or reached a stopping point, I made a commit. The follow example shows the code difference in the latest commit and the commit before that:

```
>> git diff HEAD^ HEAD HW3/solve_allen_cahn.m

diff --git a/HW3/solve_allen_cahn.m b/HW3/solve_allen_cahn.m
index b2026bb..8543aba 100644
--- a/HW3/solve_allen_cahn.m
+++ b/HW3/solve_allen_cahn.m
@@ -1,4 +1,4 @@
-function sol = solve_allen_cahn(timesteps, spatialGridPts, doublePrecision)
+function sol = solve_allen_cahn(timesteps, spatialGridPts)

 exactSol = @(t, x, y) exp(-t) * (cos(2 * pi * x) + cos(4 * pi * y).^2);

@@ -9,10 +9,6 @@ x = x(:);
 y = y(:);
 y0 = exactSol(tspan(1), x, y);

-if nargin > 2 && ~doublePrecision
-    y0 = single(y0);
-end
-
 params.alpha = 1;
 params.beta = 1;
 params.n = spatialGridPts;
```