2D advection show-and-tell with PETSc

I wrote a book called *PETSc for Partial Differential Equations* which was published by SIAM Press in 2021. The C and Python example codes from the book are at

`github.com/bueler/p4pdes`

Here I will demonstrate a C example which solves a 2D advection problem. To do this yourself, download and install PETSc (`petsc.org`) and clone the above repository.

What is PETSc? The Portable, Extensible Toolkit for Scientific computing is an open and free C numerical software library from Argonne National Laboratory. It contains iterative linear algebra, nonlinear solvers, mesh management and mesh-aware solvers, and ODE solvers, all implemented in parallel. It is targeted at high-resolution PDE and multiphysics simulations. Started before 1995, PETSc co-evolved with the Message Passing Interface (MPI) as fundamental infrastructure for doing simulations on supercomputers. The Python finite element library Firedrake and the Parallel Ice Sheet Model are two out of many examples of scientific computing libraries and applications which are built on top of PETSc.

Documentation and download for PETSc is at `petsc.org`, but it can be installed using a package manager or by installing a higher-level library (e.g. Firedrake).

Example. The C code `advect.c` from Chapter 11 takes a finite difference/volume approach for solving a scalar advection equation in 2D, on $(x,y) \in (-1,1)^2$, with periodic boundary conditions, for $t > 0$:

$$u_t + \nabla \cdot (\mathbf{a} u) = 0$$

The velocity field in this example is a rotational vector field around the origin: $\mathbf{a}(x,y) = \langle y, -x \rangle$. We use an equally-spaced periodic mesh (grid) in x and y to generate the MOL ODE system. The spatial derivatives, which are needed to compute the flux on the edge of each finite volume cell (dashed in left figure), are approximated with finite differences and a nonlinear, flux-limited, and higher-order upwind scheme; see the figure. The time-stepping is by a 3rd-order adaptive Runge-Kutta method, suitable for such hyperbolic problems if the fluxes are discretized appropriately.

Here is a parallel example run with visualization, over 4 cores, on a 160×160 spatial grid:

```
$ cd c/ch11/ && make advect
$ mpiexec -n 4 ./advect -da_refine 5 -adv_problem rotation \
    -ts_max_time 6.283185 -ts_monitor -ts_monitor_solution draw
```

A surface plot of the initial condition $u(x,y,0)$ looks like a cone and a square tower (center figure). The time-dependent solution rotates the initial picture back to the same configuration at $t_f = 2\pi$, but numerical diffusion causes some smoothing of the sharp edges (right figure). Note the apparent absence of "ringing" which we would have gotten from Lax-Wendroff or leap-frog.