Show and tell with PETSc and Firedrake

Examples from my book. I wrote a book called *PETSc for Partial Differential Equations* which was published by SIAM Press in 2021. The C and Python codes for the book's examples are at `github.com/bueler/p4pdes`. In this demonstration I'll show examples from Chapters 5, 11, and 14.

What is PETSc? The *Portable, Extensible Toolkit for Scientific computing* is an open and free C library of numerical software, especially linear algebra, mesh management, and ODE IVP solvers, from Argonne National Laboratory. Starting in about 1990, PETSc co-evolved with the *Message Passing Interface* (MPI), also from Argonne, as the fundamental infrastructure for doing science and engineering simulations and computations on supercomputers, the largest of which have more than a million processors (cores). MPI and PETSc are essential "software stack" for most large-scale *parallel* computations.

Documentation and download is at `petsc.org`.

Example 1. Chapter 5 solves a pair of *coupled diffusion-reaction equations* on $(x,y) \in (0,2.5) \times (0,2.5)$ and $t > 0$:

$$u_t = D_u \nabla^2 u - uv^2 + \phi(1-u)$$
$$v_t = D_v \nabla^2 v + uv^2 - (\phi+\kappa)v$$

where D_u, D_v, ϕ, κ are constants and $u(t,x,y), v(t,x,y)$ are chemical concentrations. This is a model for pattern generation, for instance as an explanation of how animal skins can end up spotted.

The C code `pattern.c` calls the PETSc library for time-stepping, parallel grid management, and parallel, iterative solvers for linear systems. The spatial derivatives are approximated with a 9-point-stencil centered finite difference scheme for ∇^2, which generates an MOL system. Default time-stepping (`ts_`) is by an adaptive method which is implicit for the stiff diffusion part and explicit for the non-stiff, nonlinear reaction terms. Other methods can be chosen at run-time, for instance by `-ts_type beuler`, and so on.

Here is how to build it, and run it in parallel (4 cores) on a 96×96 spatial grid:

```
$ cd c/ch5/ && make pattern
$ mpiexec -n 4 ./pattern -da_refine 5 -ts_max_time 5000 -ts_monitor \
    -ts_monitor_solution draw
```

Example 2. C code `advect.c` in Chapter 11 takes a finite volume approach, to generate the MOL ODE system, for solving a scalar *advection equation* in 2D, on $(x,y) \in (-1,1) \times (-1,1)$, with periodic boundary conditions, for $t > 0$:

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

The velocity field in this example is rotational: $\mathbf{a}(x,y) = \langle y, -x \rangle$. The spatial derivatives are approximated with finite differences and a flux-limited higher-order upwind scheme. The time-stepping is by a 3rd-order adaptive Runge-Kutta method, quite suitable for such hyperbolic problems if the fluxes are discretized appropriately.

Here is an example run (4 cores, 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 $u(x, y, 0)$ would look like a cone and a square tower. The time-dependent solution rotates the initial picture back to the same configuration at $t_f = 2\pi$, but numerical diffusion does cause the sharp edges to smooth out.

Example 3. The last example uses a Python code in Chapter 14. It solves a *Stokes problem* for a steady flow of a 2D viscous, incompressible fluid with velocity $\mathbf{u} = \langle u, v \rangle$, pressure p, and constant viscosity $\mu > 0$. The 3 coupled scalar equations are

$$-\nabla \cdot (2\mu\, D\mathbf{u}) + \nabla p = \mathbf{0},$$
$$\nabla \cdot \mathbf{u} = 0.$$

The boundary value problem has zero velocity $\mathbf{u} = \mathbf{0}$ on the bottom and sides of the unit square, but along the top we impose right-ward motion; this is called a *lid-driven cavity*.

Firedrake (`firedrakeproject.org`) is a Python finite element library which allows us to express a PDE problem directly, independent of a choice of particular (spatial) discretization. More precisely we express the *weak form* of the PDE in the *Unified Form Language*. Then Firedrake will apply the finite element (FE) method and generate a linear or nonlinear system of algebraic equations, which is then solved by PETSc solvers. For the Stokes problem we write the following core description as part of `stokes.py`:

```
V = VectorFunctionSpace(mesh, 'CG', degree=2)
W = FunctionSpace(mesh, 'CG', degree=1)
Z = V * W
up = Function(Z)
u,p = split(up)
v,q = TestFunctions(Z)
Du = 0.5 * (grad(u) + grad(u).T)
Dv = 0.5 * (grad(v) + grad(v).T)
F = (2.0 * args.mu * inner(Du,Dv) - p * div(v) - div(u) * q) * dx
```

The discretized problem uses a completely arbitrary triangular mesh, generated here by a custom Python script (to specify the domain) and then *Gmsh*. One generates and views the mesh, solves the problem (Schur-complement multigrid over 4 cores), and visualizes the solution using *Paraview* as follows:

```
$ cd python/ch14/
$ ./lidbox.py mesh.geo
$ gmsh -2 mesh.geo
$ gmsh mesh.msh
$ mpiexec -n 4 ./stokes.py -mesh mesh.msh -refine 5 -s_ksp_type gmres \
    -schurgmg lower -schurpre selfp -s_ksp_rtol 1.0e-12 \
    -showinfo -o solution.pvd
$ paraview solution.pvd
```

The sparse linear system here has 3 million degrees of freedom, on a mesh of 8×10^5 triangular elements.

A similar Stokes model for glacier ice, a fluid with nonlinear viscosity, is at

`github.com/bueler/stokes-ice-tutorial`