UCB Math 228B, Spring 2015: Problem Set 3

Due March 5

1. Consider Euler's equations of compressible gas dynamics in two space dimensions:

$$u_t + \nabla \cdot F = 0, \quad \text{where } u = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \text{ and } F = \begin{bmatrix} \rho u & \rho v \\ \rho u^2 + p & \rho u v \\ \rho u v & \rho v^2 + p \\ u(\rho E + p) & v(\rho E + p) \end{bmatrix} \tag{1}$$

Here, ρ is the fluid density, u, v are the velocity components, and E is the total energy. For an ideal gas, the pressure p has the form $p = (\gamma - 1)\rho(E - (u^2 + v^2)/2)$, where γ is the adiabatic gas constant. We will solve these on a square domain with periodic boundary conditions, for $0 \le t \le T$. The spatial derivatives will be discretized with a fourth order compact Padé scheme, and the solution will be filtered using a sixth order compact filter. A standard RK4 scheme will be used for time integration.

a) Write a function `euler_fluxes` with

Inputs : `r, ru, rv, rE`

Outputs : `Frx, Fry, Frux, Fruy, Frvx, Frvy, FrEx, FrEy`

which returns the 8 flux functions in (1) for the 4 solution components. Assume $\gamma = 7/5$.

b) Write a function `compact_div` with

Inputs : `Fx, Fy, h`

Outputs : `divF`

which calculates the divergence of a grid function field $F = [F_x, F_y]$ using the 4th order compact Padé scheme with periodic boundary conditions and grid spacing h:

$$\alpha f'_{i-1} + f'_i + \alpha f'_{i+1} = a \frac{f_{i+1} - f_{i-1}}{2h}, \qquad \alpha = 1/4, \qquad a = \frac{2}{3}(\alpha + 2)$$

c) Write a function `compact_filter` with

Inputs : `u, alpha`

Outputs : `u`

which filters the grid solution u using the 6th order compact filter with parameter α:

$$\alpha \hat{f}_{i-1} + \hat{f}_i + \alpha \hat{f}_{i+1} = a f_i + \frac{c}{2}(f_{i+2} + f_{i-2}) + \frac{b}{2}(f_{i+1} + f_{i-1})$$

where $a = 5/8 + 3\alpha/4$, $b = \alpha + 1/2$, $c = \alpha/4 - 1/8$

d) Write a function `euler_rhs`

Inputs : `r, ru, rv, rE, h`

Outputs : `fr, fru, frv, frE`

which computes the right-hand side of the discretized $-\nabla \cdot F$ (essentially just calling `euler_fluxes` and `compact_div`).

Turn page $\longrightarrow$

e) Write a function `euler_rk4step` with

Inputs : `r, ru, rv, rE, h, k, alpha`

Outputs : `r, ru, rv, rE`

which takes one RK4 step using `euler_rhs` and filters each solution component using `compact_filter`.

f) Verify the correcness of your solver using the function `euler_vortex` (on the course web page). Use a square domain $0 \le x, y \le 10$ with grid spacings $h = 10/N$ and $N = 32, 64, 128$. Use the time step $k \le 0.3h$, adjusted so the final time $T = 5\sqrt{2}$ is a multiple of k. Use the initial solution:

```
pars = [0.5, 1.0, 0.5, np.pi/4, 2.5, 2.5]
r, ru, rv, rE = euler_vortex(x, y, 0.0, pars)
```

and compare with the exact final solution:

```
r0, ru0, rv0, rE0 = euler_vortex(x, y, 5 * np.sqrt(2), pars)
```

Calculate the errors in the infinity norm over all solution components. Plot the errors vs. h in a log-log plot, for the 3 grid spacings h and the two filter coefficients $\alpha = 0.499$, $\alpha = 0.48$. Estimate the slopes of the two curves.

g) Simulate a Kelvin-Helmholtz instability, using the unit square domain $0 \le x, y \le 1$ with grid spacing $h = 1/N$ and $N = 256$ grid points in each coordinate direction, $\alpha = 0.48$, time step $k \le 0.3h$, final time $T = 2.0$, and the initial condition:

$$
\begin{aligned}
\rho &= \begin{cases} 2 & \text{if } |y - 0.5| < (0.15 + \sin(2\pi x)/200)\,, \\ 1 & \text{otherwise.} \end{cases} \\
u &= \rho - 1, \quad v = 0, \quad p = 3
\end{aligned}
$$

Plot the final solution using a contour or color plot of the density ρ.

Turn page $\longrightarrow$

2. Write a function `pmesh` with

Inputs : `pv, hmax, nref`

Outputs : `p, t, e`

which generates an unstructured triangular mesh of the polygon with vertices `pv`, with edge lengths approximately equal to $h_{\max}/2^{n_{\text{ref}}}$, using a simplified Delaunay refinement algorithm. The outputs are the node points `p` (N-by-2), the triangle indices `t` (T-by-3), and the indices of the boundary points `e`.

(a) The 2-column matrix `pv` contains the vertices x_i, y_i of the original polygon, with the last point equal to the first (a closed polygon).

(b) First, create node points along each polygon segment, such that all new segments have lengths $\leq h_{\max}$ (but as close to $h_{\max}$ as possible). Make sure not to duplicate any nodes.

(c) Triangulate the domain using `Delaunay` in Scipy or `delaunayn` in Octave.

(d) Remove the triangles outside the domain (see for example the `contains_point` function in Matplotlib or `inpolygon` in Octave).

(e) Find the triangle with largest area A. If $A > h_{\max}^2/2$, add the circumcenter of the triangle to the list of node points.

(f) Retriangulate and remove outside triangles (steps (c)-(d)).

(g) Repeat steps (e)-(f) until no triangle area $A > h_{\max}^2/2$.

(h) Refine the mesh uniformly n_{ref} times. In each refinement, add the center of each mesh edge (no duplicates) to the list of node points, and retriangulate.

Finally, find the nodes `e` on the boundary using the `boundary_nodes` command. The example in the figures uses the arguments below, but also make sure that the function works with other polygons, $h_{\max}$, and n_{ref}.

```
pv = np.array([[0,0], [1,0], [.5,.5], [1,1], [0,1], [0,0]])
hmax = 0.2
nref = 1
```