CS294 - Homework Assignment 4
Particle-in-Cell Vortex Method
Due date: 11:59 PM, November 1, 2017

October 23, 2017

You will be implementing parts of a particle-in-cell (PIC) method for vorticity dynamics in 2D. This is primarily an exercise in more elaborate template programming, with a little sorting thrown in. Generally speaking, you are integrating an ODE of the form

$$\frac{dX}{dt} = F(t, X) \tag{1}$$

In this problem set our forcing functions will all be independent of time, so you can ignore the `a_time` argument, but the `RK4` template class implements the more general form, so you need to implement your code to conform to the interface. We will be using the 4th-order explicit Runge-Kutta integration technique to evolved this system of ODEs. In this case X is the class `ParticleSet`. To see what the RK4 interface is doing, we re-write the Runge-Kutta algorithm as follows.

$$\begin{aligned}
k_1 =& \Delta t F(t^n, X^n) \\
k_2 =& \Delta t F(t^n + \frac{1}{2}\Delta t, X^n + \frac{1}{2}k_1) \\
k_3 =& \Delta t F(t^n + \frac{1}{2}\Delta t, X^n + \frac{1}{2}k_2) \\
k_4 =& \Delta t F(t^n + \Delta t, X^n + k_3)
\end{aligned}$$

$$X^{n+1} = X^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

In pseudocode, this is given by

```
K := 0; delta := 0;
K:= dt*F(t,X+K); delta+=K; K*= 1/2;
K:= dt*F(t+dt/2,X+K); delta+=2*K; K*= 1/2;
K:= dt*F(t+dt/2,X+K); delta+=2*K;
K:= dt*F(t+dt,X+K); delta += K;
```

Thus the stages of RK4 all require the calculations of the form

$$k := \Delta t F(t, X + k) \tag{2}$$

where the k appearing on the right-hand side is scaled by $\frac{1}{2}$ or 1 as appropriate prior to performing the calculation.

The `F` template class implements an operator that evaluates everything on the right of the equal sign, given the inputs `a_dX, a_time, a_dt, a_X`. It first computes a temporary `a_X + a_k`, then evaluates the right-hand side and stores in in `a_k` and scales the result by `a_dt`.

In the particular case of our particle method, the `F` template class contains no data, and the single member function

```
void ParticleVelocities::operator()(ParticleShift& a_k,
                                    const double& a_time,
                                    const double& a_dt,
                                    ParticleSet& a_X)
```

The input is the current estimate for the increment in the state `a_k`; and on return it contains the output new estimate for k; the time to evaluate the function (ignored in this case); the timestep to take Δt, the state at the start of the timestep `a_X`.

Specific Instructions

You are to implement the `ParticleSet, ParticleShift` classes conforming to the declarations in `ParticleSet.H`. You will also implement in `ParticleVelocities.cpp` the single member function `ParticleVelocities::operator()(ParticleShift& a_k, const double& a_time, const double& dt, ParticleSet& a_state)` : computes the $k's$ induced on a set of particles by all of the particles in the input `ParticleSet` displaced by the input k. The `main` in `VortexMethod2D.cpp` implements the following four tests, all defined on the unit square.

1. (`test = 1`) A single particle, with strength $1./h^2$, placed at (.49,.24). The number of grid points is given by $M = 6, 2^M = N = 64$; run to time $t = 10$. The displacement of the particle should be roundoff, since the velocity induced by a single particle on itself should vanish.

2. (`test = 2`) Two particles: one with strength $1/h^2$ located at (.5,.5), the other with strength 0, located at (.5,.25). The number of grid points is given by $M = 6, 2^M = N = 64$. Run to time $t = 10$. The strength 1 particle should not move, while the zero-strength particle should move at constant angular velocity on a circle centered at (.5,.5) of radius .25.

3. (`test = 3`) Two particles: located at (.5,.25) and (.5,.75) both with strength $1/h^2$. The number of grid points is given by $M = 6, 2^M = N = 64$. Run to time $t = 10$. Both particles should move at a constant angular velocity on a circle centered at (.5,.5) of radius .25.

4. (`test =` something other than 1, 2, or 3) Two-patch problem. For each point $\boldsymbol{i} \in [0 \dots N_p]$, $N_p = 256$, we place a particle at the point $\boldsymbol{i}h_p$, $h_p = \frac{1}{N_p}$ provided that

$$||\boldsymbol{i}h_p - (.5, .375)|| \leq .12 \text{ or } ||\boldsymbol{i}h_p - (.5, .625)|| \leq .12.$$

The strength of each of the particles should be h_p^2/h^2. This corresponds to a pair of patches of vorticity of constant strength. The grid $M = 7, 2^M = N = 128$, particle refinement factor = 2 (i.e. 4 particles / grid point). Integrate the solution to time T = 12.5.

In the first three cases, the first particle in the exact solution has a fixed radial location relative to the center of the square, (0.,.25,.25, respectively) and we output the computed radius to compare. For the final calculation, we output plot files for the particles and the vorticity on the mesh. They should look similar to the results in Lecture 17. By setting `ANIMATION = TRUE` in your makefile, you can produce a pair of plotfiles every time step (particle locations, vorticity field on the grid). The default is to produce a pair of plotfiles at the end of the calculation, and only for the two-patch case.

Description of Algorithm for Computing the Velocity Field

1. Depositing the charges in the particles on the grid.

$$\omega^g_{\boldsymbol{i}} = \sum_k \omega^k \Psi(\boldsymbol{i}h - \boldsymbol{x}^k)$$

where the $\boldsymbol{x}^k$'s are the positions of the particles in `a_state` displaced by the input `a_k`. This is givn as follows. Initialize

$$\omega^g \equiv 0.$$

Then, for each particle,

$$\boldsymbol{i}^k = \left\lfloor \frac{\boldsymbol{x}^k}{h} \right\rfloor,$$
$$\boldsymbol{s}^k = \frac{\boldsymbol{x}^k - \boldsymbol{i}^k h}{h},$$

$$\begin{aligned}
\omega^g_{\boldsymbol{i}^k} +&= \omega^k (1 - s_0^k)(1 - s_1^k), \\
\omega^g_{\boldsymbol{i}^k + (1,0)} +&= \omega^k s_0^k (1 - s_1^k), \\
\omega^g_{\boldsymbol{i}^k + (0,1)} +&= \omega^k (1 - s_0^k) s_1^k, \\
\omega^g_{\boldsymbol{i}^k + (1,1)} +&= \omega^k s_0^k s_1^k.
\end{aligned}$$

2. Convolution with the Green's function to obtain the potential on the grid, using Hockney's algorithm. The Hockney class should be constructed and maintained in `ParticleSet` - all you have to do is call the member function `Hockney::convolve` at the appropriate time to compute

$$\psi_{\boldsymbol{i}} = \sum_{\boldsymbol{j} \in \mathbb{Z}^2} G(\boldsymbol{i} - \boldsymbol{j}) \omega^g_{\boldsymbol{j}}.$$

3. Compute the fields on the grid using finite differences.

$$\vec{U}^g_{\boldsymbol{i}} = \Big(\frac{\psi_{\boldsymbol{i}+(0,1)} - \psi_{\boldsymbol{i}-(0,1)}}{2h}, -\frac{\psi_{\boldsymbol{i}+(1,0)} - \psi_{\boldsymbol{i}-(1,0)}}{2h}\Big).$$

4. Interpolate the fields from the grid to the particles.

$$\vec{U}^k = \sum_{\boldsymbol{i}\in\mathbb{Z}^2} \vec{U}_{\boldsymbol{i}} \Psi(\boldsymbol{x}^k - \boldsymbol{i}h),$$

$$\boldsymbol{i}^k = \Big\lfloor \frac{\boldsymbol{x}^k}{h} \Big\rfloor,$$
$$\boldsymbol{s}^k = \frac{\boldsymbol{x}^k - \boldsymbol{i}^k h}{h},$$

$$\begin{aligned}\vec{U}^k =& \vec{U}^g_{\boldsymbol{i}}(1-s^k_0)(1-s^k_1),\\ &+\vec{U}^g_{\boldsymbol{i}+(1,0)} s^k_0 (1-s^k_1),\\ &+\vec{U}^g_{\boldsymbol{i}+(0,1)} (1-s^k_0) s^k_1,\\ &+\vec{U}^g_{\boldsymbol{i}+(1,1)} s^k_0 s^k_1.\end{aligned}$$

Note that the operator`ParticleVelocities::operator()` requires you to return in `a_k` the quantities $\{\Delta t \vec{U}^k\}_{k=1,\dots,N_p}$.