

# Particle Methods for Vortex Problems

March 27, 2025

You will be implementing parts of a particle-in-cell (PIC) method for vortex dynamics, described below. This is primarily an exercise in more elaborate template programming. Generally speaking, you are integrating an ODE of the form

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

In this problem set our forcing functions will all be independent of time, so you can ignore the `a_time` argument, but it is good to have this form available to you when you use RK4 in other projects. 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`.

The stages of RK4 all require the calculation of quantities of the form

$$k := \Delta t * F(t + \Delta t, X + k) \quad (2)$$

Your  $F$  operator is an evaluation of everything on the right of the equal sign. RK4 is built up by various estimates of what the update to  $X$  should be, then recombined to cancel out low order error terms to create a stable method with an error in the solution that is  $O(\Delta t)^4$ .

Specifically, you will implement the class `ParticlesVelocities`, that has the single member function

```
void operator()(ParticleShift& a_result,  
               double a_time,  
               double a_deltat,  
               const ParticleSet& a_X)
```

The input is the current estimate for  $k$ , `a_result`, and the output new estimate for  $k$  is returned in `a_result`:

$$k := \Delta t F(t + \Delta t, X + k)$$

Inputs are the time you are to evaluate the function  $t + \Delta t$ , the timestep to take  $\Delta t$ , the state at the start of the timestep  $X$  in this case `ParticleSet`, and the shift to use to this state in this evaluation of  $F$   $k$ , represented by the `ParticleShift` class. In the case of our particle method,  $F$  has no explicit dependence on the first time argument, but we still have implement our class as if it does, in order to conform to the general RK4 interface.

## Specific Instructions

You are to implement in the directory `/src/Particles` `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  $k$ . In addition, you are to implement a driver program that performs the following calculations.

1. A single particle, with strength  $1/h^2$ , placed at (i)  $(.5,.5)$  , (ii)  $.4375, .5625$ , (iii)  $.45, .55$  . The number of grid points is given by  $N = 32$ ,  $\Delta t = 1$ .; run for 100 time step. In all of these cases, the displacement of the particle should be roundoff, since the velocity induced by a single particle on itself should vanish. In the case of the initial position of  $(.5,.5)$  the displacement should be comparable to roundoff. Output: position of the particle after one step
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  $N = 32$ . Run for 300 time steps,  $\Delta t = .1$  . 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  $.25$ . Output: graph of the time history of the radius and angle.
3. Two particles: located at  $(.5,.25)$  and  $(.5,.75)$  both with strength  $1/h^2$ . The number of grid points is given by  $N = 32$ . Both particles should move at a constant angular velocity on a circle centered at  $(.5,.5)$  of radius  $.25$ . Output: graph of the time history of the radius and angle for both particles.
4. Two-patch problem. For each point  $i \in [0 \dots N_p]$ ,  $N_p = 128, 256$ , place a particle at the point  $ih_p$ ,  $h_p = \frac{1}{N_p}$  provided that

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

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. Take the grid spacing  $N = 64$ . Integrate the solution to time  $T = 15$ , plotting the result at least every 1.25 units of time (to make a nifty movie, plot every time step). Set  $\Delta t = .025$ . We will provide a reference solution against which you can compare yours.

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 for the two-patch case.

## Description of Algorithm for Computing the Velocity Field

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

$$\omega_i^g = \sum_k \omega^k \Psi(ih - x^k)$$

where the  $\mathbf{x}^k$ 's are the positions of the particles in `a.state` displaced by the input `a.k`'s.

$$\omega^g \equiv 0$$

$$\begin{aligned} \mathbf{i}^k &= \left\lfloor \frac{\mathbf{x}^k}{h} \right\rfloor \\ \mathbf{s}^k &= \frac{\mathbf{x}^k - \mathbf{i}^k h}{h} \\ \omega_{\mathbf{i}^k}^g + &= \omega^k (1 - s_0^k) (1 - s_1^k) \\ \omega_{\mathbf{i}^k + (1,0)}^g + &= \omega^k s_0^k (1 - s_1^k) \\ \omega_{\mathbf{i}^k + (0,1)}^g + &= \omega^k (1 - s_0^k) s_1^k \\ \omega_{\mathbf{i}^k + (1,1)}^g + &= \omega^k s_0^k s_1^k \end{aligned}$$

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

$$\psi_{\mathbf{i}} = \sum_{j \in \mathbb{Z}^2} G(\mathbf{i} - \mathbf{j}) \omega_j^g$$

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

$$\vec{U}_{\mathbf{i}}^g = \left( \frac{\psi_{\mathbf{i}+(0,1)} - \psi_{\mathbf{i}-(0,1)}}{2h}, -\frac{\psi_{\mathbf{i}+(1,0)} - \psi_{\mathbf{i}-(1,0)}}{2h} \right)$$

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

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

$$\begin{aligned} \mathbf{i}^k &= \left\lfloor \frac{\mathbf{x}^k}{h} \right\rfloor \\ \mathbf{s}^k &= \frac{\mathbf{x}^k - \mathbf{i}^k h}{h} \end{aligned}$$

$$\begin{aligned} \vec{U}^k &= \vec{U}_{\mathbf{i}^k}^g (1 - s_0^k) (1 - s_1^k) \\ &+ \vec{U}_{\mathbf{i}^k + (1,0)}^g s_0^k (1 - s_1^k) \\ &+ \vec{U}_{\mathbf{i}^k + (0,1)}^g (1 - s_0^k) s_1^k \\ &+ \vec{U}_{\mathbf{i}^k + (1,1)}^g s_0^k s_1^k \end{aligned}$$

Note that the operator `ParticleVelocities::operator()` requires you to return in `a.k` the quantities  $\Delta t \vec{U}^k$ .