CS294 / CS194 - Homework Assignment 4
Method of Local Corrections Vortex Method
Revised 11/27/11
Due date: Dec 6, 2011

November 27, 2011

1 Problem Description

You will be implementing parts of Anderson's Method of Local Corrections algorithm for solving the vortex formulation of Euler's Equations. A lengthy description is given in Scott Baden's Thesis in Section 2 that is checked into the class resources directory. You can ignore the "workmap" material , as that relates to parallel load balancing, and you are just concerned with a serial implementation of this algorithm.

2 Part 1: Prescribed velocity field problem

Recall, 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 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 have the form

$$k = dT * F(t + t', X + x') \tag{2}$$

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 update that has an error proportional to dT^4.

You will write a simple F class for rigid body rotation about the point [0.5,0.5]. Edit the file `ParticleTest.cpp`. Most of the code has already been written for you, all except the actual update estimates function.

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

The output is an estimate for k, `a_result`. Inputs are the time you are to evaluate the function $t + t'$, the timestep to take dT, 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 x', represented by the `ParticleShift` class.

The current implementation is wrong. You are to replace it with the correct implementation of a rigid body rotation of `omega` radians per second counter clockwise about the point [0.5, 0.5].

3 Part 2 Compute order of your solution

The error in your integration is calculated for you in `ParticleTest.cpp` for several values of the timestep size (since we are computing one full revolution of rigid body rotation, the exact answer is just the starting positions).

Edit the main function to output an estimate of your schemes *order of accuracy* between each two successive solutions. The order of accuracy of a scheme p is defined as(where h is the timestep size and C is some constant independent of h)

$$\Phi_h = \Phi_{exact} + Ch^p \tag{3}$$

4 Part 3: Vortex Method Calculations

5 Specific Instructions

You are to code up and check in

TestVortex.cpp In this assignment you are to write the main( ) function. Your main function will do the following:

- Prompt the user to input the number of grid points N to use in the grid solver. For all of the problems described here, your computational domain will be the unit square $[0, 1] \times [0, 1]$, and your grid will be a nodal-point grid $\{0, \ldots, N\} \times \{0, \ldots, N\}$.
- Define and initialize the ParticleSet object that will contain the description of the particle collection that you will be evolving in time. For each of the test cases below, we will provide sufficient information for you to define the data members.
- Define the RK4 class using the ParticleSet class, the ParticleVelocity class, and the ParticleShift class, corresponding to the class template parameters `X,F,K`. The declarations of these classes are contained in the header files `ParticleShift.H, ParticleVelocity.H, and ParticleShift.H`, respectively.

- Write a time stepping loop that will integrate the particle distribution forward in time for a specified number of time steps, using a fixed time step Δt. At the end of each time step, use the `PWrite` function to output the particle locations for purposes of plotting using visit.

`ParticleSet.cpp` . The implementation of `ParticleSet` provided to you is incomplete. You will need to implement the member function `ParticleSet::rebin`, which bin sorts the particles and stores the information in the objected pointed to by the shared pointer `m_bins`.

`ParticleVelocity.cpp` . You will implement the `ParticleVelocity` class based on the method of local corrections PPPM method given in class. Your implementation will use the member functions of the classes `MLCVortexMethod, PoissonInfinite`. These are:

- `MLCVortexMethod::computeVelocities`: computes the velocities induced on a set of particles by all of the particles in the input `ParticleSet` on the input `int a_bin[DIM]`, and stores each component in the tuple `MDArray[DIM]` (note change in interface from the original).
- `PoissonInfinite::applyOperator`: computes the operator applied to the input `MDArray a_phi`, and stores it in the `MDArray a_LOfPhi`. a_LOfPhi.getBox() must be contained in `MDArray a_phi.getBox().grow(-1)`.
- `PoissonInfinite::solve` takes the input `MDArray a_rhs` as the right-hand side for the Poisson equation, solves using a Fast Sine Transform and the James-Lackner algorithm for computing the infinite domain boundary conditions, and stores the result in the `MDArray a_solution`. Both `a_rhs` and `a_solution` must be defined on the box $[0 \dots N] \times [0 \dots N]$.
- `MLCVortexMethod::MLCInterpolate` interpolates the velocities from the input for all of the particles located in the bin `a_bin` using the grid velocities stored in `MDArray<Real> a_gridVelocities[DIM]`, and stores them in the appropriate locations in the input `ParticleSet a_particles`.

Test Problems 1. A single particle, with strength 1., placed at (i) (.5,.5) , (ii) .4375,5625, (iii) .45,.55 . The number of grid points is given by N = 32, correction radius = 3, cutoff distance 1.4/N, $\Delta t = 1.$; run for 1 time step. In all of these cases, the displacement of the particle should be small, since the velocity induced by a single particle on itself should vanish - it doesn't exactly, due to numerical error in the representation of the infinite-domain boundary condition. 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 located at (.5,.5), the other with strength 0, located at (.5,.25). The number of grid points is given by N = 32, correction radius = 3, cutoff distance 1.4/N. 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: one with strength 1 located at (.5,.25), the other with strength 1, located at (.5,.75). The number of grid points is given by N = 32, correction radius = 3, cutoff distance 1.4/N. Run for 300 time steps, $\Delta t = .1$. 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 $\boldsymbol{i} \in [0 \ldots N_p]$, $N_p = 140, 280$, 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. This corresponds to a pair of patches of vorticity of constant strength. Take the grid spacing $N = 32$, the correction radius to be 2, and the cutoff distance to be $h_p^{.75}$. 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 $\Delta t = .025$. Your results should roughly match up with those in the Baden paper on p. 108, except his times are double yours, i.e. your t=2.5 corresponds to his t=5.0 .

6 Some Hints

- The member functions `copyTo , += ,...` of `MDArray<T>` have been extended to apply to methods in which the `Box` over which the right-hand side is defined is not equal to that of the object calling the function. In that case, the operation is performed on the intersection of the two `Boxes`. This will make some of the steps of MLC much easier.

- When looping over bins, perform operations only if there are particles in the bin.

- The formal definition of binsort is that a particle at position $\boldsymbol{x}_p$ is in bin $\boldsymbol{i}$ only if $\boldsymbol{x} \in [(i_0 - .5)h, (i_0 + .5h)] \times [(i_1 - .5)h, (i_1 + .5h)]$. Using the `cmath` function `floor` will make this calculation much easier.

- Shared pointers are tricky. The easiest way to initialize the shared pointer for the `Bins` is to initialize one in the main program, and then assign it to the member data in your `ParticleSet` object.

- Before running the last two problems out to the full times, make sure you have recompiled with optimization on. The larger problem should take less than an hour.