Fundamentals of Simulation Methods

Final Exam with Answers

Janosh Riebesell

February 8th, 2016

Lecturer: Volker Springel

Note The provided answers earned a 1,0 in the actual exam.

1 Short questions (24 pts)

State whether the following statements are right or wrong, and discuss in 1 to 2 sentences why this is the case. Note: there might be any number of correct statements.

a) Solvers for fluid dynamical problems:
   1) Finite-volume solvers on a fixed Cartesian grid are Galilean invariant.
   2) Smoothed particle hydrodynamics has a higher spatial resolution in high density regions.
   3) The spatial order of a discontinuous Galerkin scheme can be increased by increasing the order of the basis polynomials.
   4) Roe's Riemann solver provides an exact solution of the piecewise constant initial value problem.

a) Fluid dynamics
   1) False. Smoothed particle hydrodynamics are fully Galilean invariant. Finite-volume solvers are not.
   2) Correct. Due to its Lagrangian character, the local resolution of SPH follows the mass flow automatically, a property that is convenient in representing the large density contrasts often encountered in astrophysical simulations.
   3) Correct. This so-called p-refinement can even be remarkably more efficient than an h-refinement, i.e. an increase in the grid resolution.
   4) False. As the full name Roe's *approximate* Riemann solver suggests, it yields an approximate solution.

b) You want to solve Poisson's equation for a given density field on a 3D periodic grid of 1000 cells per dimension using iterative solvers. First, you employ a Gauss-Seidel iteration scheme and find the system to converge quite slowly. Which of the following methods could be used to speed up the convergence rate?
   1) Jacobi iteration scheme instead of Gauss-Seidel.

2) Red-black sweeps.
3) Multigrid methods.
4) Clouds-in-cell (CIC) assignment.

b) Iterative solvers

1) False. Gauss-Seidel's immediate use of updated Φ-values allows it to attain a faster (usually close to twice as fast) rate of convergence than experienced with Jacobi iterations.
2) False. Red-black sweeping is a way to (partially) overcome Gauss-Seidel's poor parallelizability and the dependence of the final result on which grid element is taken first when starting the iteration. It does not improve the convergence rate.
3) Correct. Multigrid methods converge *substantially* faster than Gauss-Seidel. That is because mutligrid methods operate on much coarser and hence computationally cheaper grids for much of the time. Converged results from the coarse grid levels are used to improve the initial guess on finer levels, reducing the amount of work that has to be done at full resolution.
4) False. CIC is a mass (or charge) assignment kernel used with the particle mesh technique. It has nothing at all to do with iterative solvers.

c) When simulating a collisionless system, gravitational softening is needed to

1) prevent self-forces on particles.
2) reduce two-body interactions with sizable deflections.
3) prevent the formation of bound particle pairs.
4) allow the use of the Verlet integrator instead of a 2nd-order Runge-Kutta.

c) Gravitational softening

1) False. Self-forces are unrelated to gravitational softening. Vanishing self-forces are ensured by employing the same assignment kernel W to calculate forces at the particle positions as was used in the initial density construction on the fiducial mesh.
2) Correct. In a singular potential, close-range interactions result in large-angle scatterings. Adding a softening parameter ϵ to the gravitational potential effectively introduces a smallest resolved length scale into the system which prohibits such close contact.
3) Correct. If $\langle v^2 \rangle \gg \frac{Gm}{\epsilon}$, a softening length ϵ prevents highly correlated bound states which would violate the system's supposed collisionlessness.
4) False. The Verlet method is a symplectic integrator particularly popular in molecular dynamics simulations. A gravitational softening is in no way related to the use of a specific integrator.

2 Runge-Kutta methods (24 pts)

An explicit Runge-Kutta method to solve the equation

$$\frac{\mathrm{d}}{\mathrm{d}t} y = f(t) \tag{1}$$

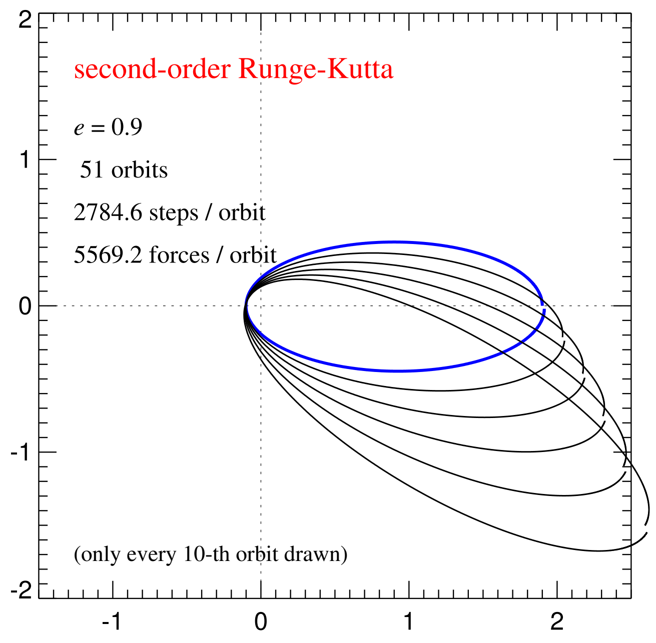

Figure 1: Kepler orbit

needs two function evaluations and can be generally written as

$$y(t+\Delta t) = y(t) + [b_1 f(t) + b_2 f(t+\alpha\Delta t)]\Delta t. \tag{2}$$

a) Show that for the construction of the second-order accurate scheme, b_1, b_2, and α are not uniquely determined. Which relations do b_1, b_2, and α have to obey?

b) Figure 1 shows an orbit (i.e. a planet orbiting around a star) integration carried out with a Runge-Kutta method. How can you recognize that the result is inaccurate? Explain how the accuracy could be improved.

a) The relations that b_1, b_2 and α need to fulfill in order for eq. (2) to constitute a second-order accurate integrator follow from two straightforward Taylor expansions. For the exact solution $y_{\text{ex}}(t)$, we can write

$$\begin{aligned} y_{\text{ex}}(t) &= y(t) + \dot{y}(t)\Delta t + \frac{1}{2}\ddot{y}(t)\Delta t^2 + \mathcal{O}_s(\Delta t^3) \\ &= y(t) + f(t)\Delta t + \frac{1}{2}\dot{f}(t)\Delta t^2 + \mathcal{O}_s(\Delta t^3), \end{aligned} \tag{3}$$

where we used the ODE (1) in the second step. $f(t+\alpha\Delta t)$ on the other hand, we may expand into

$$f(t+\alpha\Delta t) = f(t) + \alpha\dot{f}(t)\Delta t + \mathcal{O}_s(\Delta t^2). \tag{4}$$

The difference between the exact solution $y_{\text{ex}}(t)$ and the approximate method $y(t+\Delta t)$ is then

$$\begin{aligned} y_{\text{ex}}(t) - y(t+\Delta t) = y(t) &+ f(t)\Delta t + \frac{1}{2}\dot{f}(t)\Delta t^2 + \mathcal{O}_s(\Delta t^3) \\ &- \left\{ y(t) + \left[b_1 f(t) + b_2 \left(f(t) + \alpha\dot{f}(t)\Delta t + \mathcal{O}_s(\Delta t^2) \right) \right] \Delta t \right\} \\ &= (1-b_1-b_2)f(t)\Delta t + (\tfrac{1}{2} - \alpha b_2)\dot{f}(t)\Delta t^2 + \mathcal{O}_s(\Delta t^3). \end{aligned} \tag{5}$$

Thus, eq. (2) is second-order accurate if

$$1 - b_1 - b_2 = 0 \quad \text{and} \quad \tfrac{1}{2} - \alpha b_2 = 0. \tag{6}$$

These are just two constraints for the three variables b_1, b_2, and α. They are hence not uniquely determined by requiring second-order accuracy alone.

b) The simulation's inaccuracy manifests itself in the rapidly changing orbit and its increasing eccentricity. Both suggest a violation of phase-space volume conservation (as mandated by the Liouville theorem) and a secular drift of the total energy.

The simulation could be carried out with higher accuracy by resorting to a symplectic integrator. When applied to Hamiltonian systems such as a Kepler orbit, their use of structure-oreserving maps award them with excellent conservation properties as far as the above-mentioned energy and phase-space volume are concerned.

3 Pitfalls of floating point math (24 pts)

a) Assume you have a single-precision IEEE floating point variable x (which has a mantissa of 24 bit). You use it as a counter in a loop that carries out 10^9 iterations, and in each iteration you add 1 to x (starting with an initial value $x = 0$). Once the loop has finished, what do you get for x? Why?

b) Now repeat the experiment from above, but you add 4 to x in each iteration of the loop. What will be the final result now? Why?

c) Suppose you have a long data set with N measurement values x_i, and you want to determine the variance

$$\sigma^2 = \langle x^2 \rangle - \langle x \rangle^2 \tag{7}$$

of them. Let's consider two possible approaches to implement this on a computer:

1) We first write a loop to build the sum of the x_i, namely

$$S = \sum_{i=1}^{N} x_i, \tag{8}$$

and the sum of the x_i^2 as

$$T = \sum_{i=1}^{N} x_i^2, \tag{9}$$

then we calculate σ^2 as

$$\sigma^2 = \frac{T}{N} - \frac{S^2}{N^2}. \tag{10}$$

2) Alternatively, we do not calculate T but write a second loop in which we evaluate

$$Q = \sum_{i=1}^{N} \left(x_i - \frac{S}{N} \right)^2, \tag{11}$$

and then set

$$\sigma^2 = \frac{Q}{N}. \tag{12}$$

Discuss advantages and disadvantages of these solutions.

a) One of the most common issues encountered in floating point calculations are round-off errors. Operations such as $x+y = x$ can occur if y lies below the employed data type's machine precision ϵ_m, i.e. if $|y| < \epsilon_\mathrm{m}|x|$. In this case, we have $y = 1$, a precision of $p = 24$, and $\epsilon_\mathrm{m} = \frac{1}{2^p} = \frac{1}{2^{24}}$, which yields for x a final value of

$$x_\mathrm{f} = \frac{1}{\epsilon_\mathrm{m}} = 2^{24} \approx 1.68 \times 10^7. \tag{13}$$

b) If we add 4 with each iteration, the criterion for round-off to occur simply becomes $4 < \epsilon_\mathrm{m}|x|$, and hence x_f now reads

$$x_\mathrm{f} = \frac{4}{\epsilon_\mathrm{m}} = 4 \times 2^{24} \approx 6.71 \times 10^7. \tag{14}$$

c) Method 1) is called a *one-pass* formula, because S and T can both be calculated simultaneously and independent of each other. On average, $x_i^2 > (x_i - \frac{S}{N})^2$. Therefore, Method 1) is more prone to suffer from *overflow*.

Method 2) is a *two-pass* formula because it first requires calculating S before it is able to determine Q. For large deviations x_i from the mean value $\frac{S}{N}$ of the distribution, Q may assume a large value while the summation is still in progress. Repeated small contributions that might add up to significance when put all together will then all be lost to *round-off errors*.

Both methods are numerically stable only for small data sets and small x_i. For large N, they depend inordinately on the ordering of the data.

4 Code example (24 pts)

The following two functions are part of a code which adds particles to a discrete 1D density field (first function), solves Poisson's equation in some way to calculate a discretized acceleration field (not shown here) and maps the force field back to the particles (second function).

a) Assuming that all parts of the code which are not shown work perfectly fine, what problem will occur when running the code with the two functions shown below?

b) Modify the second function such that the above problem is solved. It's sufficient if you write pseudo code.

c) Do the functions work for periodic boundary conditions? Why?

```c
void add_particle_to_density_field(double rho_field[N], //the density
   discretized on a grid of length N
    int N,
    double cellsize, //size of a single cell of the grid
    double particle_pos, // the particle position
    double particle_mass) // the particle mass
{
    double xx = particle_pos / cellsize;
    int i = floor(xx); //floor(x) truncates all decimal digits of the
       floating point number x, similar to (int) x in C (or: it does a
       strict rounding to the next lower integer number; floor(5.9) = 5

    double u = xx - i;

    int ii = i + 1;
    if(ii >= N)
        ii = 0;

    rho_field[i] += (1 - u) * particle_mass / cellsize;
    rho_field[ii] += u * particle_mass / cellsize;
}
```

```
double interpolate_force_field_to_particle_position(double acc_field[N],
   // the acceleration field discretized on a grid of length N
    int N,
    double cellsize, //size of a single cell fo the grid
    double particle_pos, // the particle position
    double particle_mass) // the particle mass
{
    double xx = particle_pos / cellsize;

    int i = floor(xx + 0.5);

    double acceleration = acc_field[i];

    return acceleration * particle_mass;
}
```

a) A particle mesh simulation carried out with the above functions will suffer from nonvanishing self-forces and non-antisymmetric pairwise forces, the reason being that the force interpolation function uses a different assignment kernel than what was used in the initial density construction performed by `add_particle_to_density_field`.

b) The problem may be remedied by modifying the force interpolation as follows:

```
double interpolate_force_field_to_particle_position(double acc_field[N],
    int N, double cellsize, double particle_pos, double particle_mass)
{
    double xx = particle_pos / cellsize;

    int i = floor(xx);

    double u = xx - i;

    int ii = i + 1;
    if(ii >= N)
        ii = 0;

    double acc = (1 - u) * acc_field[i] + u * acc_field[ii];

    return acc * particle_mass;
}
```

c) Yes, the two functions are compatible with periodic boundary conditions since `ii` is mapped back to 0 if `ii >= 0`.