PHYS 210: Introduction to Computational Physics
Finite Difference Solution of N-Body Problems

Note: This document is subject to update, but the current version will always be available via the course notes page

1. THE GRAVITATIONAL N-BODY PROBLEM

1.1 Physical & Mathematical Formulation

- Consider N point particles, labelled by an index i, with masses m_i

$$m_i \qquad i = 1, 2, \ldots N$$

and position vectors, $\mathbf{r}_i(t)$

$$\mathbf{r}_i(t) \equiv [x_i(t),\, y_i(t),\, z_i(t)] \qquad i = 1, 2, \ldots N$$

where we have established a standard set of Cartesian coordinates (x, y, z) with some arbitrarily chosen origin (In practice, however, it may be most convenient to choose the origin at the center of mass of the system).

- We wish to study the dynamics of the system due to the (attractive) Newtonian gravitational force exerted by each particle on every other particle.

- Combining Newton's second law, as well as his law of gravitation, we have the basic equations of motion in vector form

$$m_i\, \mathbf{a}_i = G \sum_{j=1,\, j\neq i}^{N} \frac{m_i\, m_j}{r_{ij}^2}\, \hat{\mathbf{r}}_{ij} \ , \qquad i = 1, 2, \ldots N \ , \qquad 0 \leq t \leq t_{\max} \tag{1}$$

where

  - $\mathbf{a}_i = \mathbf{a_i}(\mathbf{t})$ is the acceleration of the i-th particle
  - G is Newton's gravitational constant
  - r_{ij} is the magnitude of the separation vector $\mathbf{r}_{ij}$ between particles i and j:

$$\mathbf{r}_{ij} \equiv \mathbf{r}_j - \mathbf{r}_i$$

$$r_{ij} \equiv |\mathbf{r}_j - \mathbf{r}_i|$$

and we recall that the magnitude of any vector, $\mathbf{w} = [w_x,\, w_y,\, w_z]$ is given by:

$$w \equiv |\mathbf{w}| = \sqrt{w_x^2 + w_y^2 + w_z^2}$$

  - $\hat{\mathbf{r}}_{ij}$ is the unit vector in the direction from particle i to particle j (i.e. in the direction of the separation vector:)

$$\hat{\mathbf{r}}_{ij} \equiv \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}} \tag{2}$$

– From now on, for brevity of notation we will use

$$\sum_{j=1,\, j\neq i}^{N} \rightarrow \sum_{j}$$

and $i = 1, 2, \ldots, N$ and $0 \leq t \leq t_{\max}$ will be implied.

- For the purposes of computation, it turns out to be more convenient to use (2) in (1) to get

$$m_i \, \mathbf{a}_i = G \sum_j \frac{m_i \, m_j}{r_{ij}^3} \, \mathbf{r}_{ij} \tag{3}$$

where we note that

$$r_{ij}^3 = \Big[(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2 \Big]^{3/2}$$

- It is also convenient to "non-dimensionalize" the system of equations, which in this case means choosing units in which $G = 1$, which we will hereafter do

- **Begin Aside: Non-dimensionalizing**

- Note that if we have a problem in mechanics that involves up to three parameters p_k, $k = 1, 2, 3$ which have *distinct* dimensions

$$M^{\alpha_k} \, L^{\beta_k} \, T^{\gamma_k}$$

where L, M and T denote length, mass and time respectively, and the α_k, β_k and γ_k are generally integers, then we can *always* choose a system of units in which $p_1 = p_2 = p_3 = 1$.

- Note that in the current case we have a single dimension-ful parameter, G

$$[G] = M^{-1} L^3 T^{-2}$$

(so $\alpha = -1$, $\beta = 3$ and $\gamma = -2$).

- Non-dimensionalizing simplifies actual calculations, but has the drawback that one must generally convert back-and-forth between the non-dimensional set of units, and the desired set (*MKS* for example) to make contact with specific physical setups (e.g. dynamics of the solar system).

- For the purposes of your term projects, you *should* work with $G = 1$.

- **End Aside:**

- Continuing, we have

$$\mathbf{a}_i(t) = \frac{d^2 \mathbf{r}(\mathbf{t})}{dt^2}$$

so (3) becomes (with $G = 1$)

$$m_i \, \mathbf{a}_i = m_i \, \frac{d^2 \mathbf{r}_i}{dt^2} = \sum_j \frac{m_i m_j}{r_{ij}^3} \, \mathbf{r}_{ij}$$

and then dividing both sides of the above equation by m_i, we have

$$\frac{d^2 \mathbf{r}_i}{dt^2} = \sum_j \frac{m_j}{r_{ij}^3} \, \mathbf{r}_{ij} \tag{4}$$

- (4) is a system of second-order-in time differential equations for the *vector* quantities, $\mathbf{r}_i(t)$

- In order to compute a specific solution, we must supply initial conditions, which in this case are the initial positions and initial velocities of the particles, i.e.

$$\mathbf{r}_i(0) = \mathbf{r}_{0i} \qquad i = 1, 2, \ldots, N \tag{5}$$

$$\mathbf{v}_i(0) \equiv \frac{d\mathbf{r}}{dt}(0) = \mathbf{v}_{0i} \qquad i = 1, 2, \ldots, N \tag{6}$$

where $\mathbf{r}_i(0)$ and $\mathbf{v}_{0i}$ are specified vectors (total of $6N$ numbers)

- Now, express (4), (5) and (6) in "component form" (i.e. take x, y and z components), then ($i = 1, 2, \ldots, N$ is implicit in the following equations)

$$\frac{d^2x_i(t)}{dt^2} = \sum_j \frac{m_j}{r_{ij}^3}\,(x_j - x_i) \tag{7}$$

$$\frac{d^2y_i(t)}{dt^2} = \sum_j \frac{m_j}{r_{ij}^3}\,(y_j - y_i) \tag{8}$$

$$\frac{d^2z_i(t)}{dt^2} = \sum_j \frac{m_j}{r_{ij}^3}\,(z_j - z_i) \tag{9}$$

and the initial conditions take the form

$$x_i(0) = x_{i0} \tag{10}$$
$$y_i(0) = y_{i0}$$
$$z_i(0) = z_{i0}$$
$$v_{x_i}(0) = v_{xi0} \tag{11}$$
$$v_{y_i}(0) = v_{yi0}$$
$$v_{z_i}(0) = v_{zi0}$$

where the x_{i0}, y_{i0}, z_{i0}, v_{xi0}, x_{yi0} and v_{zi0} are given values (again, 6 values per particle)

1.2 Solution via Finite Difference Approximation

1.2.1 Discretization: Step 1—Finite Difference Grid

- Continuum domain is

$$0 \le t \le t_{\max}$$

- We will assume that we can proceed using a uniform time mesh (i.e. constant time step) as usual: may *not* be a good assumption, particularly if particles start "clumping"

- Specify mesh via *level* parameter, ℓ

$$n_t = 2^\ell + 1$$
$$\Delta t = \frac{t_{\max}}{n_t - 1} = 2^{-\ell} t_{\max}$$
$$t^n = (n-1)\Delta t, \quad n = 1, 2, \ldots, n_t$$

1.2.2 Discretization: Step 2—Derivation of FDAs

- Continuum equations → discrete equations

- Will illustrate procedure for $x_i(t)$: $y_i(t)$ and $z_i(t)$ can be treated in identical fashion ($i = 1, 2, \ldots, N$ still implicit in the following)

- FD notation

$$x_i^n = x_i(t^n)$$

  where we use a superscript, rather than subscript, n, since we are using a superscript to enumerate the particles.

- Need approximation for second time derivative, use usual second order centred formula

$$\left.\frac{d^2 x_i(t)}{dt^2}\right|_{t=t^n} \approx \frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2}$$

- Substituting in (7), we have

$$\frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2} = \sum_j \frac{m_j}{\left(r_{ij}^n\right)^3}\left(x_j^n - x_i^n\right) \qquad n+1 = 3, 4, \ldots, n_t \tag{12}$$

- Again, we view this as an equation for the advanced-time values x_i^{n+1}, assuming that the values x_i^n and x_i^{n-1} are known

- Solving explicitly for x_i^{n+1}, we have

$$x_i^{n+1} = 2x_i^n - x_i^{n-1} + \Delta t^2 \sum_j \frac{m_j}{\left(r_{ij}^n\right)^3}\left(x_j^n - x_i^n\right) \qquad n+1 = 3, 4, \ldots, n_t \tag{13}$$

- This last equation (plus the corresponding equations for y_i^{n+1} and x_i^{n+1}) are our basic finite difference equations for the N-body problem.

- As usual for a problem in dynamics, we need to deal with the initial conditions, and, again since we are using a three-time-level scheme, we thus need to determine values for $x_i^1 = x_i(0)$ and $x_i^2 = x_i(\Delta t)$

- As usual, the x_i^1 follow immediately from the initial conditions

$$x_i^1 = x_{i0}$$

  but we need to use Taylor series expansion to determine the values x_i^2 to $O(\Delta t^2)$

$$x_i^2 = x_i(\Delta t) = x_i(0) + \Delta t \frac{dx_i(t)}{dt}(0) + \frac{1}{2}\Delta t^2 \frac{d^2 x_i(t)}{dt^2}(0) + O(\Delta t^2) \tag{14}$$

- Using the initial conditions for the particle positions and velocities (x components), (7) to eliminate the second time derivative, and neglecting higher order terms we have

$$x_i^2 = x_{i0} + \Delta t\, v_{xi0} + \frac{1}{2}\Delta t^2 \sum_j \frac{m_j}{r_{ij0}^3}\left(x_{j0} - x_{i0}\right) \tag{15}$$

  where

$$r_{ij0}^3 = \left[\left(x_{j0} - x_{i0}\right)^2 + \left(y_{j0} - y_{i0}\right)^2 + \left(z_{j0} - z_{i0}\right)^2\right]^{3/2} \tag{16}$$

- Keep in mind that we have two other sets of equations and initial values for the y_i^n and z_i^n

- If we wish to consider 2D motion, we can simply set all of the z_{i0} and v_{zi0} to 0

1.3. Energy Quantities and Energy Conservation

- Total kinetic energy

$$T(t) = \sum_{i=1}^{N} \frac{1}{2} m_i \, {v_i}^2 \tag{17}$$

- Total potential energy

$$V(t) = -\sum_{i=1}^{N} \sum_{j=1,\, j<i}^{N} \frac{m_i \, m_j}{r_{ij}} \tag{18}$$

- **Important:** Note the the second summation in the above is limited to values of j that are strictly *less* than i.

  If we summed over all values of j—i.e. so that the upper limit of the sum was N—we would "double count" the potential energy contributions (think, e.g., of the two-particle case where there is only one contribution)

- Total conserved energy

$$E(t) = T(t) + V(t) \tag{19}$$

- Can compute discrete versions of these quantities, and especially for small numbers of particles, test for convergence of

$$dE(t) = E(t) - E(0)$$

  as one way of establishing code correctness

1.4. Octave Implementation Suggestions

- Use multi-dimensional arrays to store discrete positions
- Ideally store entire solution (i.e. all time steps) as we did with pendulum example, and as you are asked to do in Homework 4.
- For example, create and "zero" 3-dimensional array `r` via

```
r = zeros(N, 3, nt);

   N:   number of particles
   nt:  total number of time steps
```

- Then would have the following

$$\texttt{r(i,1,n)} \equiv x_i^n$$
$$\texttt{r(i,2,n)} \equiv y_i^n$$
$$\texttt{r(i,3,n)} \equiv z_i^n$$

- Consider writing an acceleration-computing routine with the header

```
function [a] = nbodyaccn(m, r)

m:  Vector of length N containing the particle masses
r:  N x 3 array containing the particle positions
a:  N x 3 array containing the computed particle accelerations
```

  Note that the particle accelerations are given by the right hand side of equation (12): once they have been computed, they can be used in equation (13) to update the particle positions.

- Begin with a code that uses loops to compute the accelerations and, if necessary to, implement the basic difference equations. Once that is working, save it/rename it, and try to write a version that uses whole array operations as much as possible
- We will discuss an application, `xfpp3d`, that you can use to visualize the output from N-body simulations in the lab.

1.5. Suggested test case

- A good, non-trivial configuration that you can use to develop and test your implementation describes two particles with arbitrary masses in mutual circular orbit about their center of mass, and in the x-y plane.
- **EXERCISE:** Let the particle masses be m_1 and m_2, respectively, and let them be separated by a distance r. Let the initial position and velocity vectors be

$$\begin{aligned} \mathbf{r}_1(0) &= (r_1, 0, 0) \\ \mathbf{r}_2(0) &= (-r_2, 0, 0) \\ \mathbf{v}_1(0) &= (0, v_1, 0) \\ \mathbf{v}_2(0) &= (0, -v_2, 0) \end{aligned}$$

  where r_1, r_2, v_1 and v_2 are all positive quantities, so that the separation is given by $r = r_1 + r_2$. Show that if

$$\begin{aligned} r_1 &= \frac{m_2}{m} r \\ r_2 &= \frac{m_1}{m} r \\ v_1 &= \frac{\sqrt{m_2 r_1}}{r} \\ v_2 &= \frac{\sqrt{m_1 r_2}}{r} \end{aligned}$$

  where $m = m_1 + m_2$ is the total mass of the system, then the particles *will* execute circular orbits about the center of mass.
- **NOTE:** If you *do* use this configuration to develop/test your code, I expect that you will include the verification (or derivation) of the above results in your writeup.

2. ELECTROSTATIC N-BODY PROBLEM

(For those working on charges-on-a-sphere problem (COSP) or similar)

- Let us now consider a collection of N point charges, with charges q_i and masses m_i:

$$m_i \ , \ i = 1, 2, \ldots N$$

$$q_i \ , \ i = 1, 2, \ldots N$$

- Defining a coordinate system and position vectors precisely as we did for the gravitational case, we have the electrostatic equations of motion:

$$m_i \, \mathbf{a}_i = m_i \frac{d^2 \mathbf{r_i}}{dt^2} = -k_e \sum_{j=1,\, j \neq i}^{N} \frac{q_i \, q_j}{r_{ij}^2} \, \hat{\mathbf{r}}_{ij} \ , \qquad i = 1, 2, \ \ldots \ N \ , \qquad 0 \leq t \leq t_{\max} \tag{30}$$

  where k_e is Coulomb's constant (note the $-$ sign relative to (1), since there is a *repulsive* force between charges of the same sign).

- However, in this case *we demand that the charges remain on the surface of a sphere.* Let us assume that the radius of the sphere is R and that it is centred at the origin of our coordinate system, $(0, 0, 0)$.

  We observe that the choice of R is essentially arbitrary, and will have no effect on the equilibrium positioning of the N charges. Thus we set $R = 1$.

  Then we must have

$$r_i \equiv |\mathbf{r}_i| \equiv \sqrt{x_i^2 + y_i^2 + z_i^2} = 1 \, , \quad i = 1, 2, \ldots N$$

- The simplest form of COSP has identical charges, i.e. equal masses and equal charges (of the same sign). This is the version that you should implement in your term projects, at least to begin with.

  In this case, it is especially convenient to non-dimensionalize (and again, this is always possible), so that

$$m_i = 1 \ , \ i = 1, 2, \ldots N$$

$$q_i = 1 \ , \ i = 1, 2, \ldots N$$

$$k_e = 1$$

- Eqn (30) then becomes simply

$$\mathbf{a}_i = -\sum \frac{\hat{\mathbf{r}}_{ij}}{r_{ij}^2} \tag{31}$$

  where, as before, we are now using

$$\sum_{j=1,\, j \neq i}^{N} \rightarrow \sum_{j}$$

  with $i = 1, 2, \ldots, N$ and $0 \leq t \leq t_{\max}$ implied.

- Now, the basic idea behind COSP is to start the N charges at some arbitrary positions on the sphere and, most conveniently, with no velocity. We then use dynamical evolution to find the (an?) equilibrium configuration. In order for the charges to "settle down" to that configuration, we must add some dissipation (friction) to the system.

- A straightforward way to do this is to add a term to the equation of motion that is proportional to the velocity, and which retards the motion, i.e. (31) becomes

$$\mathbf{a}_i = -\sum \frac{\hat{\mathbf{r}}_{ij}}{r_{ij}^2} - \gamma \mathbf{v}_i \tag{32}$$

  where γ is an adjustable parameter which controls the amount of dissipation (and which is something that you will need to experiment with in your implementation)

- Although it might not seem entirely natural for this problem, it is still best to use Cartesian components of (32) in simulations. This yields

$$\frac{d^2x_i(t)}{dt^2} = -\sum_j \frac{(x_j - x_i)}{r_{ij}^3} - \gamma \frac{dx_i}{dt} \tag{33}$$

$$\frac{d^2y_i(t)}{dt^2} = -\sum_j \frac{(y_j - y_i)}{r_{ij}^3} - \gamma \frac{dy_i}{dt} \tag{34}$$

$$\frac{d^2z_i(t)}{dt^2} = -\sum_j \frac{(z_j - z_i)}{r_{ij}^3} - \gamma \frac{dz_i}{dt} \tag{35}$$

- We can now discretize these equations exactly as we did for the gravitational case, except that now we need to handle the velocity (friction) term, for which we use the $O(\Delta t^2)$ centred approximation for the first derivative, e.g.

$$\left.\frac{dx_i(t)}{dt}\right|_{t=t^n} \approx \frac{x_i^{n+1} - x_i^{n-1}}{2\Delta t}$$

- Thus our discretization of the equation of motion in the x-direction (33) is

$$\frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2} = -\sum_j \frac{\left(x_j^n - x_i^n\right)}{\left(r_{ij}^n\right)^3} - \gamma \frac{x_i^{n+1} - x_i^{n-1}}{2\Delta t} \qquad n+1 = 3, 4, \ldots, n_t \tag{36}$$

  and the y and z equations have precisely the same form.

- I will leave it to you to solve (36) (as well as the y and z eqns.) for the advanced-time unknowns, x_i^{n+1}

- As mentioned above, you can initialize the particle positions arbitrarily (just don't put two or more in the same place!) and the easiest thing to do is to set the initial velocities to 0.

- Unlike the gravitational N-body case, we are only interested in the final, equilibrium configuration of the charges here, not in the details of the dynamics.

- This means that there is no need to use the Taylor series technique to "properly" initialize the values x_i^2, y_i^2, z_i^2. Assuming that the initial velocities *are* 0, then it will suffice to set $x_i^2 = x_i^1$, $y_i^2 = y_i^1$ and $z_i^2 = z_i^1$,

- **Important:** At all time steps, you must ensure that the charges *are* on the unit sphere. When you use (36) (and the y and z equations) to advance the system by Δt, the charges will generally move off the sphere. One easy way to get them back on the surface is to simply "project" them along their position vectors (from the origin).

- I.e. assuming that $\tilde{x}_i^{n+1}$, $\tilde{y}_i^{n+1}$ and $\tilde{z}_i^{n+1}$ are the provisional values of the charge coordinates after the time step, then setting

$$x_i^{n+1} = \frac{\tilde{x}_i^{n+1}}{\tilde{r}_i} \qquad y_i^{n+1} = \frac{\tilde{y}_i^{n+1}}{\tilde{r}_i} \qquad z_i^{n+1} = \frac{\tilde{z}_i^{n+1}}{\tilde{r}_i}$$

  where

$$\tilde{r}_i = \sqrt{(\tilde{x}_i^{n+1})^2 + (\tilde{y}_i^{n+1})^2 + (\tilde{z}_i^{n+1})^2}$$

  will place the charges back on the sphere.

- You should convince yourself that this works!

- Note the other differences from the gravitational problem

  - Energy is *not* conserved here, so there is no point in checking for its conservation
  - Convergence tests will also be of little/no use, and should not be included as part of your report
  - You have the advantage that you know where the charges should be—on the surface of the sphere—and you will/should know when you are computing the equilibria correctly

- The implementation hints for the gravitational case also apply here.

- Feel free to seek help from me should you have any questions about this project.