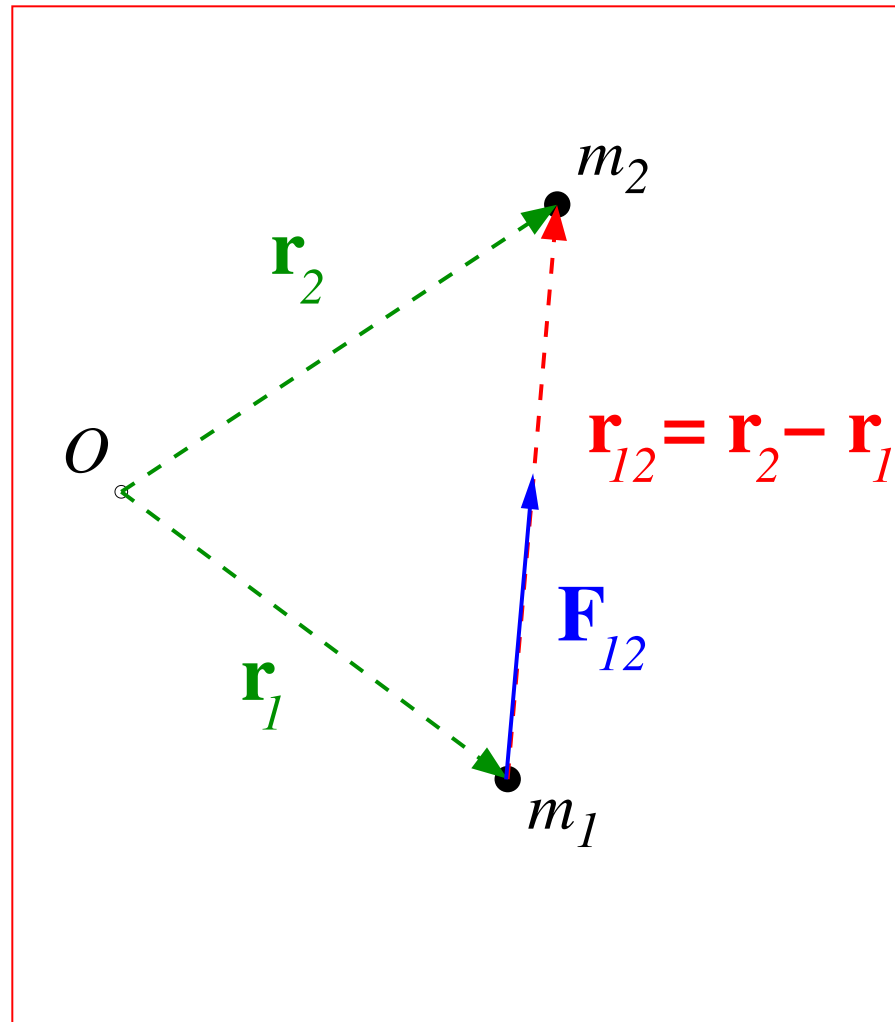
PHYSICS 21

SOLUTION OF N-BODY PROBLEMS USING FDAS

1. PHYSICAL & MATHEMATICAL FORMULATION



1.1 Derivation of the equations of motion

- 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

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

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

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

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)

$$\hat{\mathbf{r}}_{ij} \equiv \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}} \tag{3}$$

in (1) to get

$$m_i \, \mathbf{a}_i = G \sum_j \frac{m_i \, m_j}{r_{ij}^3} \, \mathbf{r}_{ij} \tag{4}$$

where we note that

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

- 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

1.2 Non-dimensionalization

- We again consider the process of non-dimensionalization as we did for the case of the nonlinear pendulum, but in this case in slightly more generality, and slightly more formally

- The key observation is 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 $\gamma = -2$).

- There are thus many (infinitely many) different ways that we can choose a system of units in which the Newton constant satisfies, $G = 1$, and we will now assume that we are working in such a system

- It is important to note that although non-dimensionalizing simplifies actual calculations (we don't have to keep track of parameters that are essentially irrelevant for the mathematics), it 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 $G = 1$.

- The issue of non-dimensionalization also applies to the *masses* of the particles

- Rather than thinking about the masses being given in some specific unit, kilogram, mass of the sun etc., it is more convenient to view the numbers assigned to the m_i as being relative to the mass of some *particular* particle

- For example, we can choose one of the particles, with label I, say, and then work in units such that $m_I = 1$ (while still keeping $G = 1$, since we can make a total of *three* transformations to fix the units of M, L and T)

- Then, for example, if $m_i = 2$, this means that, physically, the i-th particle is twice as massive as the I-th

- Note that with $G = 1$, and typical masses specified as values close to unity, the characteristic distances (separations) at which the bodies will strongly interact will be of order unity as well, and their speeds will also tend to be of order unity

- This is simply a reflection that non-dimensionalization is equivalent to the use of units that are *natural* for the problem; i.e. units in which the dynamical unknowns have values that are "close to" 1; i.e./e.g. not 10^{12} or 10^{-13} as they might be if some "non-natural" system were used

- You should also use this type of non-dimensionalization in your projects rather than, for example, trying to model our solar system (which isn't particularly recommended due to the simplicity and essential linearity of the dynamics) using distances in meters and masses in kilograms

Derivation of EOM (continued)

- Getting back to the equations of motion, the accelerations of the particles are

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

(recall that $i = 1, 2, \ldots, N$ is implied) so with $G = 1$, equation (4) becomes

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

and then dividing both sides of the above equation by m_i, we have the second-order differential equations of motion for the vector quantities $\mathbf{r}_i$ that govern the system of N particles

$$\frac{d^2\mathbf{r}_i}{dt^2} = \sum_j \frac{m_j}{r_{ij}^3}\,\mathbf{r}_{ij} \qquad (5)$$

- Note that the particle mass m_i has "dropped" out of the equations, since it appears in both the "$m_i \mathbf{a}_i$" on the left hand side of the second law, and in all of the forces acting on particle i on the right hand side

- Among other things, this means that there is no difficulty in computing the motion of "test particles"; i.e. particles with (effectively) zero-mass, but which nonetheless are subject to gravitational forces

- In particular, this is the case for the particles representing stars in the Toomre model of galaxy collisions

- As a side note, we also observe that the fact that m_i appears in all terms of Newton's second law for the problem—i.e. on both the left and right hand sides—and that it thus "cancels", is ultimately a deep observation that underlies Einstein's theory of general relativity (equivalence principle)

- Now, in order to compute a *specific* solution of the differential equations of motion (which is the only thing that we can do/model with a computer), we must supply initial conditions, which in this case are the initial positions and initial velocities of the particles, i.e.

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

$$\mathbf{v}_i(0) \equiv \frac{d\mathbf{r}}{dt}(0) = \mathbf{v}_{0i} \qquad i = 1, 2, \ldots, N \tag{7}$$

where $\mathbf{r}_{0i}$ and $\mathbf{v}_{0i}$ are *specified* vectors (3 components each, so 6 scalar values for each particle, for a total of $6N$ numbers)

- **IMPORTANT:** Remember that we are working in a Cartesian coordinate system $(x, y, z,)$ so that the time-dependent position vector of the i-th particle $\mathbf{r}_i$ is written in terms of its 3 time-dependent coordinates, $x_i(t)$, $y_i(t)$, and $z_i(t)$:

$$\mathbf{r}_i(t) \equiv [x_i(t),\, y_i(t),\, z_i(t)]$$

  In what follows, I am going to reexpress the *vector* equations of motion in component form, i.e. so that there will separate, equations for x_i, y_i and z_i, which due to the "democratic nature" of Cartesian coordinates (in contrast, for example, to spherical coordinates), are all identical in form.

- I do this primarily to emphasize that we *are* working in Cartesian coordinates, and when we get to details about the finite differencing, I will show the development only for the $x_i(t)$-coordinate, since the treatment of $y_i(t)$ and $z_i(t)$) is identical.

- However, because the equations for the three coordinate functions have identical forms we can *directly* discretize the *vector* form of the equations and then implement the FDA for the vector $\mathbf{r}_i$ in a straightforward way in Matlab (or any other programming language that supports multi-dimensional arrays)

- This type of implementation *does* add another dimension (index) to the arrays that we must deal with, which enumerates the three coordinate values, and this may take a little time to get used to, but in the end you will find that it results in the most natural and easy-to-understand form of the code.

- I will supply more details of the implementation later; for the time being just keep in mind that although I will discuss the FDA that we will use in terms of one of the coordinates, $x_(t)$ we will ultimately implement the algorithm using $\mathbf{r}_i(t)$

- So we proceed to express (5), (6) and (7) in Cartesian form by taking their x, y and z components:

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

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

$$\frac{d^2 z_i(t)}{dt^2} = \sum_j \frac{m_j}{r_{ij}^3} \, (z_j - z_i) \tag{10}$$

  and we again note that $i = 1, 2, \ldots, N$ is implicit in the above and following equations, i.e. the equations hold for each and every particle

- The component-form of the initial conditions take the form

$$
\begin{aligned}
x_i(0) &= x_{i0} && (11)\\
y_i(0) &= y_{i0} && (12)\\
z_i(0) &= z_{i0} && (13)\\
v_{x_i}(0) &= v_{xi0} && (14)\\
v_{y_i}(0) &= v_{yi0} && (15)\\
v_{z_i}(0) &= v_{zi0} && (16)
\end{aligned}
$$

where the x_{i0}, y_{i0}, z_{i0}, v_{xi0}, x_{yi0} and v_{zi0} are values that we must specify in order to get the simulation going (again, 6 values per particle)

2. SOLUTION VIA FINITE DIFFERENCE APPROXIMATION

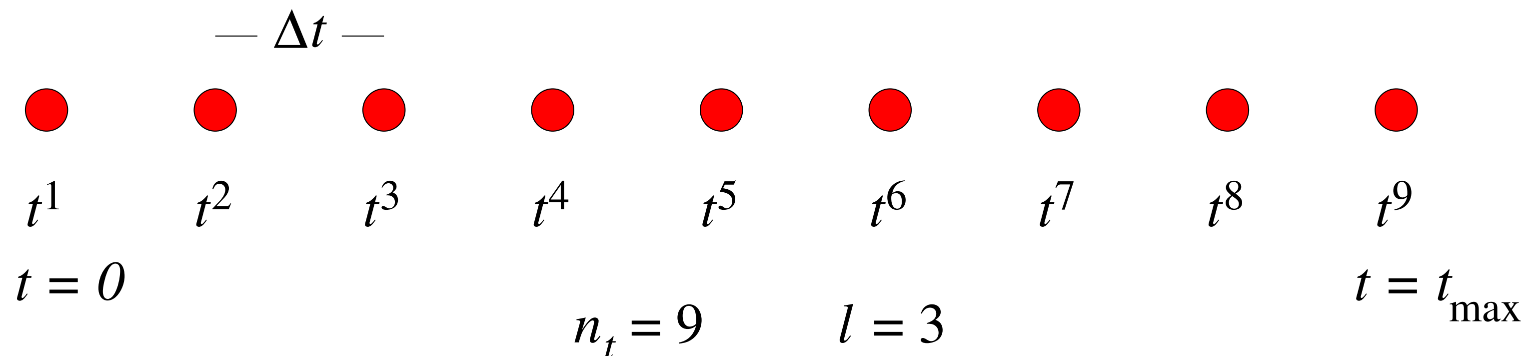
2.1 Discretization: Step 1—Finite Difference Grid

- The 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 (however, it is important to know that for general N-body calculations this will *not* be a good assumption, particularly if particles start "clumping")

- We can thus use the same type of grid that we did for the pendulum problem



where, as before, the superscript labels the discrete time step, $(t \to t^n)$, and is not to be confused with a power

- As before we can specify the number of time steps and the mesh spacing via the integer-valued *level* parameter, ℓ

$$n_t = 2^\ell + 1$$

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

- For your term projects, fixing $t_{\max}$ and using ℓ to determine the mesh parameters will be convenient for development, debugging, and testing—and especially for convergence tests

- However, when you start to do actual numerical experiments—modeling of specific scenarios—you will generally have to "play around" with `tmax`, so that the simulation time is long enough to study the behaviour of interest.

- You should then choose ℓ so that your results are acceptably accurate, subject to limits on the amount of simulation time (wall-clock time) that are available to complete the modeling before the project due date!

2.2 Discretization: Steps 2 & 3
Derivation & Solution of FDAs

- We now want to convert the continuum equations to a discrete form and we illustrate the procedure for $x_i(t)$: $y_i(t)$ and $z_i(t)$ can be treated in identical fashion (and $i = 1, 2, \ldots, N$ is still implicit in the following)

- Also remember that although the discussion here is in terms of one scalar coordinate function, $x_i(t)$, ultimately we will difference and solve the equations for the vector $\mathbf{r}_i(t)$

- We use the finite difference notation, $x_i^n = x_i(t^n)$, where superscript n labels the time step number as usual

- We need an approximation for the second time derivative and we use the same second-order formula we used for the pendulum simulation:

$$\left.\frac{d^2 x_i(t)}{dt^2}\right|_{t=t^n} \to \frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2} \quad (17)$$

- Substituting in (8), we have

$$\frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2} = \sum_j \frac{m_j}{\left(r_{ij}^n\right)^3} \left(x_j^n - x_i^n\right) \qquad n+1 = 3, 4, \ldots, n_t \tag{18}$$

- 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

$$x_i^{n+1} = 2x_i^n - x_i^{n-1} + \Delta t^2 \sum_j \frac{m_j}{\left(r_{ij}^n\right)^3} \left(x_j^n - x_i^n\right) \qquad n+1 = 3, 4, \ldots, n_t \tag{19}$$

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

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

- The values for x_i^1 follow immediately from the initial conditions

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

but by the same reasoning we appealed to for the pendulum simulation, we need to use Taylor series expansion to determine the values x_i^2 to $O(\Delta t^2)$ accuracy (i.e. *including* the $O(\Delta t^2)$ term)

$$x_i^2 = x_i(\Delta t) = x_i(0) + \Delta t \frac{dx_i}{dt}(0) + \frac{1}{2}\Delta t^2 \frac{d^2 x_i}{dt^2}(0) + O(\Delta t^4) \qquad (20)$$

- We then substitute the initial conditions for the particle positions and velocities (x components), and, importantly, use the equation of motion (8) to eliminate the second time derivative (again, this is the same procedure that we followed for the pendulum case), to get

$$x_i^2 = x_{i0} + \Delta t\, v_{xi0} + \frac{1}{2}\Delta t^2 \sum_j \frac{m_j}{r_{ij0}^3}\left(x_{j0} - x_{i0}\right) \tag{21}$$

where

$$r_{ij0}^3 = \left[\left(x_{j0} - x_{i0}\right)^2 + \left(y_{j0} - y_{i0}\right)^2 + \left(z_{j0} - z_{i0}\right)^2\right]^{3/2} \tag{22}$$

- 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

2.3 Convergence Testing

- We can apply the same techniques of convergence testing discussed in the nonlinear pendulum notes to the FDA solution of the gravitational N-body problem

- For example, denoting the solution for the x coordinate of some specific particle computed at level ℓ as $x_\ell(t^n)$ (i.e. we have suppressed the particle index to minimize notational confusion) and assuming that the FDA equations have been implemented properly (including those needed to initialize the scheme, $x_\ell(0)$ and $x_\ell(\Delta t_\ell)$), we can expect

$$x^\star(t^n) - x_\ell(t^n) = {\Delta t_\ell}^2 \, e_2(t^n) + \ldots$$

where e_2 is the leading order (dominant) error function

- We can then, for example, fix initial conditions, run calculations at discretization levels ℓ, $\ell+1$, $\ell+2$, $\ell+3$... (as many levels as we want and/or as are feasible) and verify that plots of

$$x_\ell(t^n) - x_{\ell+1}(t^n)$$

$$4 \times \left(x_{\ell+1}\left(t^n\right) - x_{\ell+2}\left(t^n\right)\right)$$

$$16 \times \left(x_{\ell+2}\left(t^n\right) - x_{\ell+3}\left(t^n\right)\right)$$

  etc., approach coincidence as ℓ increases

- The convergence test can be applied to the discrete solutions for x, y and z, and for all particles

- *Note:* It is best to perform these tests for cases with a few particles (small N), and for relatively short integration times

- For many particles, long integration times or when particles get close to one another, you are not likely to see good convergence

- This is especially the case for large values of N where the problem becomes very sensitive to initial conditions (i.e. “chaotic” behaviour ensues)

- However, the central use of convergence testing is to *validate* your implementation, i.e. to demonstrate explicitly and *intrinsically*—without resort to comparison with known or previously computed solutions—that there aren’t any bugs/defects in your code

- The point is—and this is again typical of numerical simulations—once you have established your program’s correctness for a few “suitably general” initial conditions (perhaps even one set of initial values), there is no reason to expect that it’s not correct for *any* input

- At that juncture, there’s nothing much more you can do with the code *per se*—what you get is what you get, relative to the method that has been adopted—if you’re not satisfied with the results, you will need to improve the simulation approach

2.4 Energy Quantities and Energy Conservation

- The total kinetic energy for the collection of N particles is

$$T(t) = \sum_{i=1}^{N} \frac{1}{2} m_i \, {v_i}^2 = \sum_{i=1}^{N} \frac{1}{2} m_i \, \left(v_{x_i}^2 + v_{y_i}^2 + v_{z_i}^2 \right) \tag{23}$$

- The total potential energy is

$$V(t) = - \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{m_i \, m_j}{r_{ij}} \tag{24}$$

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

  If we summed over all values of j—i.e. so that the lower limit of the second sum was 1—we would "double count" the potential energy contributions (think, e.g., of the two-particle case where there is only one contribution)

- The total conserved energy is

$$E(t) = T(t) + V(t) \tag{25}$$

- Again, paralleling the pendulum calculations, we can compute discrete versions of these quantities, and especially for small numbers of particles, we can test for *convergence* of

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

as one way of establishing code correctness (refer to the pendulum notes for more details)

- **Important note for the Toomre problem**: Only the total energy associated with the *massive* particles (i.e. those representing the galactic cores) is conserved in this case

- The sum of the KE and PE for any/all of the test particles will *not* be conserved in general, so bear this in mind in your testing

3. IMPLEMENTATION IN MATLAB

- Use *multi-dimensional arrays* to the store the particle positions

- Ideally store entire solution (i.e. all time steps) in one array, as we did in the pendulum example (if you run into memory limitations using very large numbers of particles and/or time steps, ask for further implementation suggestions as necessary)

- For example, create and (for efficiency) "zero" a 3-dimensional array `r` via

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

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

- In particular, this specific representation, with the particle, spatial coordinate, and time dimension in the precise order given above, is necessary if you want to use the instructor-supplied functions to output the results for subsequent visualization with the `xfpp3d` utility that will be demoed in the lab

- Then we will have the following correspondences

$$\texttt{r}(\texttt{i},1,\texttt{n}) \equiv x_i^n$$

$$\texttt{r}(\texttt{i},2,\texttt{n}) \equiv y_i^n$$

$$\texttt{r}(\texttt{i},3,\texttt{n}) \equiv z_i^n$$

and although the translation of the difference equations to `Matlab` will be more involved than it was for the pendulum example, it should be a relatively straightforward process if you proceed carefully

- It is important to emphasize that we are using the second dimension of the array `r` to enumerate the three coordinates x, y and z. We can thus use `Matlab's` "colon" notation for specifying all elements along that dimension to simplify/shorten code, e.g.

$$\mathtt{r}(\mathtt{i},:,\mathtt{n})$$

for specific values of `i` and `n` is a 3-element vector containing the 3 Cartesian components of particle i's position at time t^n, so that

$$\mathtt{r}(\mathtt{i},:,\mathtt{n}+\mathtt{1})-\mathtt{r}(\mathtt{i},:,\mathtt{n})$$

(again, for example) is the change from one time step to the next of all 3 components of the particle's position

- Note that the computation of the contributions to the acceleration on any particle due to the forces from all of the other particles, as given—for the case of the x-positions—by the right hand side of equation (18), is used in two places—the basic update equation (19) and the initialization of the values at the second time step t^2, i.e at time $t = \Delta t$, equation (21).

- You can thus simplify the overall program, including development/debugging, by "factoring" the code and writing a separate acceleration-computing function with a 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
```

- You can expect to use nested `for` loops to compute all elements of a

- For the Toomre model, storage of all of the particle positions—for the cores as well as all of the stars—in a *single* multidimensional array is still advocated, since it will ultimately lead to cleaner code, and will facilitate visualization using instructor-supplied routines

- In this case you will have to work a little harder with "bookkeeping" to come up with a scheme for enumerating the various types of particles so that they can still be referenced with a single array index: ask if you need assistance with this

3.1 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 the particles be separated by a distance d. Let the initial position and velocity vectors be

$$\begin{aligned} \mathbf{r}_1(0) &= (d_1, 0, 0) \\ \mathbf{r}_2(0) &= (-d_2, 0, 0) \\ \mathbf{v}_1(0) &= (0, v_1, 0) \\ \mathbf{v}_2(0) &= (0, -v_2, 0) \end{aligned}$$

where d_1, d_2, v_1 and v_2 are all positive quantities, so that the distance between the particles is $d = d_1 + d_2$.

- Show that if

$$
\begin{aligned}
d_1 &= \frac{m_2}{m} d \\
d_2 &= \frac{m_1}{m} d \\
v_1 &= \frac{\sqrt{m_2 d_1}}{d} \\
v_2 &= \frac{\sqrt{m_1 d_2}}{d}
\end{aligned}
$$

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.

4. ELECTROSTATIC N-BODY PROBLEM

(Charges-on-a-sphere problem (COSP) or similar)

4.1 Derivation of the equations of motion

- 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:

$$m_i \, \mathbf{a}_i = m_i \frac{d^2 \mathbf{r_i}}{dt^2} = -k_e \sum_{j=1,\, j \neq i}^{N} \frac{q_i \, q_j}{r_{ij}^2} \, \hat{\mathbf{r}}_{ij} \ , \qquad i = 1, 2, \ \ldots \ N \ , \qquad 0 \leq t \leq t_{\max} \tag{26}$$

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 convenient to non-dimensionalize (and, once more, 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 (26) then becomes simply

$$\mathbf{a}_i = -\sum_j \frac{\hat{\mathbf{r}}_{ij}}{r_{ij}^2} = -\sum_j \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}^3} \tag{27}$$

where, as before, we are now using

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

with $i = 1, 2, \ldots, N$ and $0 \le t \le t_{\max}$ 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. (27) becomes

$$\mathbf{a}_i = -\sum_j \frac{\mathbf{r}_j - \mathbf{r}_i}{r_{ij}^3} - \gamma \mathbf{v}_i \tag{28}$$

  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 at first glance it might not seem entirely natural for this problem, it is still best to use Cartesian components of (28) in simulations.

- If you don't believe me, ask yourself whether you would rather compute the distance between Vancouver BC and Austin TX given

  1. Their x, y, x coordinates in a Cartesian system with origin at the center of the Earth (or anywhere else for that matter), or
  2. Their latitude and longitude, which is a "natural" coordinate system to use on the surface of a sphere

- So, as with the gravitational case, I will write out the component-form of the equations, and then focus on the equation for x_i when I describe the FDA, but once again, in practice the FDA can be applied directly to the position *vectors*, $\mathbf{r}_i$, and it is best to do so.

- Doing this gives us the final form of the equations of motion:

$$\frac{d^2 x_i(t)}{dt^2} = -\sum_j \frac{(x_j - x_i)}{r_{ij}^3} - \gamma \frac{dx_i}{dt} \tag{29}$$

$$\frac{d^2 y_i(t)}{dt^2} = -\sum_j \frac{(y_j - y_i)}{r_{ij}^3} - \gamma \frac{dy_i}{dt} \tag{30}$$

$$\frac{d^2 z_i(t)}{dt^2} = -\sum_j \frac{(z_j - z_i)}{r_{ij}^3} - \gamma \frac{dz_i}{dt} \tag{31}$$

4.2 Discretization: FDAs

- We can now discretize equations (29-31) 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.

$$\left.\frac{dx_i(t)}{dt}\right|_{t=t^n} \approx \frac{x_i^{n+1} - x_i^{n-1}}{2\Delta t} \tag{32}$$

- Thus our discretization of the equation of motion in the x-direction (29) is

$$\frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2} = -\sum_j \frac{\left(x_j^n - x_i^n\right)}{\left(r_{ij}^n\right)^3} - \gamma \frac{x_i^{n+1} - x_i^{n-1}}{2\Delta t} \qquad n+1 = 3, 4, \ldots, n_t \tag{33}$$

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

- I will leave it to you to solve (33) (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, it will suffice to set $x_i^2 = x_i^1$, $y_i^2 = y_i^1$ and $z_i^2 = z_i^1$,

4.3 Constraining the charges to the sphere

- **Important:** At all time steps, you must ensure that the charges *are* on the unit sphere. When you use (33) (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 significant differences from the gravitational problem
  - Energy is *not* conserved here, so there is no point in checking for its conservation
  - The details of the COSP *dynamics* are not of much concern—it is the final states that are important—so it is not crucial that the time evolution be simulated very accurately
  - Largely because of this, convergence tests will be of limited significance and do not need to 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
  - A key challenge will be determining *when* equilibrium has been attained and that the final states your simulation yields really *are* equilibria

- The implementation hints for the gravitational case also apply here: in particular, I recommend that you store the positions of the particles in a three dimensional array with dimensions `ncharges` $\times$ 3 $\times$ `nt`.

- You may also find it useful to compute and monitor the total potential energy, $V(t)$ (use a formula analogous to (24)) during the simulations