Notes for CS542G

Robert Bridson

October 23, 2007

1 Gravity

One particular set of second-order ODEs of interest is the "n-body problem", where the motion of a set of point masses acting under Newton's law of gravitation is determined. Suppose the i'th point mass ($i = 1, \ldots, n$) has mass m_i, position $\vec{x}_i$, velocity $\vec{v}_i = \frac{d\vec{x}_i}{dt}$ and acceleration $\vec{a}_i = \frac{d\vec{v}_i}{dt}$. The force on i due to gravitational attraction from j is:

$$\vec{f}_{ij} = -G\frac{m_i m_j}{\|\vec{x}_i - \vec{x}_j\|^3}(\vec{x}_i - \vec{x}_j)$$

where G is the gravitational constant, a surprisingly hard number to measure ($G \approx 6.67428 \times 10^{-11} Nm^2/kg^2$ in SI units). The total force on i is the sum over all other points:

$$\vec{F}_i = \sum_{j \neq i} \vec{f}_{ij}$$

Even exploiting the symmetry $\vec{f}_{ij} = -\vec{f}_{ji}$ leads to $O(n^2)$ work just to evaluate the forces. This ends up being the critical bottleneck for large calculations, such as attempts to simulate galaxy formation.

2 Clustering

One approach to making this computation more tractable is to exploit clustering. In fact, this idea underlies the point mass model: the Earth is certainly not a single particle of enormous mass, but its radius is so small (and, more technically, it further possesses near symmetry) compared to the distance to other celestial bodies that as a good approximation it can be taken to be a single point. We'll exploit that in accelerating the force computation: if a cluster of many points is small enough compared to how far away it is, it can be well approximated with a single point.

Before getting to the question of determining these clusters, we need to understand how accurate this approximation is. In general it will not be perfect, and in fact we will end up with a trade-off between speed and accuracy. However, since we're also making numerical errors in time integration, and even just in the floating point representations, as long as we can control this error we needn't be overly concerned. Later we'll take a peek at the Fast Multipole Method, which in principle can reduce the errors in the force computation to the level of floating point round-off (while still giving an asymptotically faster algorithm), but there too the critical part is quantifying and controlling the error.

Suppose we aim to replace the cluster with a single point $\vec{x}_C$, where all cluster points are within distance r of $\vec{x}_C$. Let the distance from $\vec{x}_C$ to $\vec{x}_i$ (at which we are approximating the gravitational attraction of the cluster) be D: $\|\vec{x}_C - \vec{x}_i\| = D$. Let's take a look at $\vec{f}_{ij}$ for a j inside the cluster:

$$\begin{aligned} \vec{f}_{ij} &= -G\frac{m_i m_j}{\|\vec{x}_i - \vec{x}_j\|^3}(\vec{x}_i - \vec{x}_j) \\ &= Gm_i m_j \vec{h}(\Delta\vec{x}_j) \end{aligned}$$

where $\Delta\vec{x}_j = \vec{x}_j - \vec{x}_C$ and the new function $\vec{h}$ is defined as:

$$\vec{h}(\Delta\vec{x}) = \frac{\vec{x}_D + \Delta\vec{x}}{\|\vec{x}_D + \Delta\vec{x}\|^3}$$

with $\vec{x}_D = \vec{x}_C - \vec{x}_i$, a length D vector.

We'll rewrite $\vec{h}$ to be a little more convenient:

$$\begin{aligned}\vec{h}(\Delta\vec{x}) &= [(\vec{x}_D + \Delta\vec{x}) \cdot (\vec{x}_D + \Delta\vec{x})]^{-\frac{3}{2}} (\vec{x}_D + \Delta\vec{x}) \\ &= \left[\|\vec{x}_D\|^2 + 2\vec{x}_D \cdot \Delta\vec{x} + \|\Delta\vec{x}\|^2\right]^{-\frac{3}{2}} (\vec{x}_D + \Delta\vec{x})\end{aligned}$$

And now, we do our usual trick of expanding this in a Taylor series: how much different is $\vec{h}(\Delta\vec{x}_j)$, the attraction evaluated at point $\vec{x}_j$ within the cluster, from $\vec{h}(0)$, the attraction computed at the centre of the cluster?

$$\vec{h}(\Delta\vec{x}) = \vec{h}(0) + \nabla\vec{h}(0)\Delta\vec{x} + O(\partial^2\vec{h}\Delta x^2)$$

I've left the error term a little vague, which involves products of the second derivatives of $\vec{h}$ with the components of $\Delta\vec{x}$; with a little more effort we could bound this rigourously, but for the purposes of this course we'll stick with the $O()$ notation hiding a very reasonable constant. Let's compute the gradient of $\vec{h}$, using the vector version of the product rule:

$$\nabla\vec{h}(\Delta\vec{x}) = -\frac{3}{2}\left[\|\vec{x}_D\|^2 + 2\vec{x}_D \cdot \Delta\vec{x} + \|\Delta\vec{x}\|^2\right]^{-\frac{5}{2}} [2\vec{x}_D + 2\Delta\vec{x}] \otimes (\vec{x}_D + \Delta\vec{x}) + \left[\|\vec{x}_D\|^2 + 2\vec{x}_D \cdot \Delta\vec{x} + \|\Delta\vec{x}\|^2\right]^{-\frac{3}{2}} I$$

Here the $\otimes$ symbol represents the outer-product (i.e. the matrix formed from multiplying two vectors), and I is the identity matrix. Evaluating this at 0 and simplifying gives:

$$\begin{aligned}\nabla\vec{h}(0) &= -3\frac{\vec{x}_D \otimes \vec{x}_D}{\|vecx_D\|^5} + \frac{1}{\|\vec{x}_D\|^3} I \\ &= -3\frac{\vec{x}_D \otimes \vec{x}_D}{D^5} + \frac{1}{D^3} I\end{aligned}$$

It's similarly not hard to see that all the second derivatives of $\vec{h}$, evaluated near 0, will be $O(1/D^4)$. Thus our Taylor series approximation is:

$$\begin{aligned}\vec{h}(\Delta\vec{x}) &= \vec{h}(0) + \nabla\vec{h}(0)\Delta\vec{x} + O\left(\frac{\Delta x^2}{D^4}\right) \\ &= \frac{\vec{x}_D}{D^3} - 3\frac{\vec{x}_D(\vec{x}_D \cdot \Delta\vec{x})}{D^5} + \frac{\Delta\vec{x}}{D^3} + O\left(\frac{\Delta x^2}{D^4}\right)\end{aligned}$$

In the last line I rewrote the outer product times a vector in terms of the dot product (you can easily verify this step).

Now, let's plug this approximation in for the force from all points in the cluster

$$\begin{aligned}\vec{F}_{iC} &= Gm_i \sum_j m_j \vec{h}(\Delta x_j) \\ &= Gm_i \sum_j m_j \frac{\vec{x}_D}{D^3} - 3Gm_i \sum_j m_j \frac{\vec{x}_D(\vec{x}_D \cdot \Delta\vec{x}_j)}{D^5} + Gm_i \sum_j m_j \frac{\Delta\vec{x}_j}{D^3} + O\left(Gm_i \frac{r^2}{D^4}\right) \\ &= Gm_i \left(\sum_j m_j\right) \frac{\vec{x}_D}{D^3} - \frac{3Gm_i\vec{x}_D}{D^5}\vec{x}_D \cdot \left(\sum_j m_j \Delta\vec{x}_j\right) + \frac{Gm_i}{D^3}\left(\sum_j m_j \Delta\vec{x}_j\right) + O\left(Gm_i \frac{r^2}{D^4}\right)\end{aligned}$$

We haven't yet exactly determined the centre of the cluster. One natural choice is to use the centre of mass of the cluster:

$$\vec{x}_C = \frac{\sum_j m_j \vec{x}_j}{\sum_j m_j}$$

which then happily implies that:

$$\sum_j m_j \Delta\vec{x}_j = 0$$

Thus the linear terms in the Taylor series expansion cancel out, and we're left with:

$$\vec{F}_{iC} = -Gm_i m_C \frac{\vec{x}_i - \vec{x}_C}{\|\vec{x}_i - \vec{x}_C\|} + O\left(Gm_i \frac{r^2}{D^4}\right)$$

where $m_C = \sum_j m_j$ is the total mass of the cluster. Since the first term has magnitude $O(Gm_i/D^2)$, we see that replacing the cluster with its centre of mass incurs an $O(r^2/D^2)$ *relative* error for computing the force on i.

3 The Barnes-Hut Tree Algorithm

This is the central idea of the Barnes-Hut algorithm for efficiently approximating the forces in an n-body problem: replace distant clusters with their centres of mass, when the ratio r/D is small enough to keep the relative error below a user-supplied tolerance. The tricky part is determining those clusters: our data is just a scattered set of points.

This is where trees enter. As a prelude to computing forces, a tree (more specifically a "bounding volume hierarchy") is constructed around the points. For example, an octree could be used: a cube containing all the points is the root of the tree; each non-leaf node of the tree is split into eight children (in a $2 \times 2 \times 2$ arrangement, halving the length along each axis); the leaves are nodes containing one or zero points. Another popular alternative is the kd-tree, where instead of splitting a node eight ways, each node is just split in half along the longest axis. Either tree can be made more efficient if every node is shrunk to only just contain the points inside.

The total mass and centre of mass of each node (or rather, of all the points contained within a node of the tree) can be computed efficiently from the leaves up: a node's total mass is the sum of its children's masses, and its centre of mass is the centre of mass of its children's centres. Added to this data we include at each node a bounding radius r, an upper bound on the maximum distance of any of its points to the node's centre of mass.

Then, when it comes time to evaluate the force on a particular i, the tree is traversed from the root down. If the ratio r/D of the node's bounding radius to the distance from $\vec{x}_i$ to the node's centre of mass is small enough to satisfy an error tolerance, we use the approximation; otherwise, we recursively examine the children of the node. Obviously if we hit a leaf, we can use the exact formula (or ignore it if the leaf happens to be the point i itself).

Constructing the tree takes $O(n \log n)$ time and $O(n)$ space; with a constant that depends on the distribution of the points and the error tolerance, the time to compute each force is expected to take $O(\log n)$ time. Thus Barnes-Hut is an $O(n \log n)$ algorithm, a huge improvement over the naïve $O(n^2)$ exact approach.

4 Generalization

The first generalization we will point out is that although it was set up explicitly for gravity problems, the idea behind the method is not tied to this particular case. Electrostatic problems, where mass is replaced by charge (which can be both positive or negative), have the same functional form. More generally, other radial functions can be used—in particular, any of the radial basis functions we discussed earlier in the course! Taking the Taylor series approximation to reduce a cluster to a single point still may work, though the formulas may be different.

5 Symmetrization

We can in fact speed up the force computation further. While the algorithm above is extremely efficient for computing the force on a single particle i, you might imagine that two nearby particles will recompute nearly the same force for distant clusters, and there should be a way to reuse that work. The other thing that might bother you is that depending on the clustering, the forces computed so far aren't necessarily symmetric: the force on i due to j might not be perfectly balanced by the force on j due to i if clusters get involved.

This motivates a different approach, approximating the forces on all the particles simultaneously. More precisely, we'll look at the acceleration of every particle: the acceleration of two nearby points due to a distant cluster is going to be approximately the same (independent of how massive the two points are)—this is the same reason underlying the approximately constant *acceleration* (not force) due to gravity on the surface of the Earth.

Consider two clusters, both tightly bounded relative to the distance between them. If the centres of mass are $\vec{x}_{C1}$ and $\vec{x}_{C2}$ and the masses are M_1 and M_2, then the acceleration of any point i inside the first cluster due to the second cluster is approximately:

$$\vec{a}_{i,C2} \approx -G\frac{M_2}{\|\vec{x}_{C1} - \vec{x}_{C2}\|^3}(\vec{x}_{C1} - \vec{x}_{C2})$$

which follows from applying the previous approximation twice, with an error depending on the ratio of bounding radii to distance between the clusters. Underlying this is an approximation of the force between any two points i from the first cluster and j from the second cluster:

$$\vec{f}_{ij} \approx -G\frac{m_i m_j}{\|\vec{x}_{C1} - \vec{x}_{C2}\|^3}(\vec{x}_{C1} - \vec{x}_{C2})$$

and this gives rise to opposite and equal forces: $\vec{f}_{ij} = -\vec{f}_{ji}$.

We thus can approximate accelerations of all the points as follows, with two recursive functions:

- `add_internal_accelerations` (which takes one node of the tree and adds all accelerations between points inside that node),
- and `add_external_accelerations` (which takes two distinct nodes, and adds all accelerations between pairs of points from the nodes).

We begin with a call to `add_internal_accelerations` on the root node.

If `add_internal_accelerations` is called on a leaf (containing one or zero points) there is nothing to do. Otherwise it recursively calls itself on its children nodes, and calls `add_external_accelerations` on each pair of children. (Nothing is actually computed inside this function: it simply guides the work done in the other function.)

The `add_external_accelerations` function checks if its two nodes satisfy the error bound (the bounding radii are small enough compared to the distance between the centres—where for leaf nodes the bounding radius is 0): if so it computes the approximate acceleration and stores it at those nodes. Otherwise, it calls itself on the pairs with the smaller of the two nodes and the children of the larger node.

This recursive computation can be expected to take $O(n)$ time. To see this, assume that at any level of the tree, if there are k nodes at that level, each node will only need to interact with its $O(1)$ nearest neighbours in space (since more distant nodes will have been taken care of at higher levels with the cluster approximation). Thus the work is linear in the number of nodes in the tree, which is $O(n)$.

When these functions finish, they will have left accelerations computed at the highest level nodes where it was accurate enough—but yet not accumulated at the points. One final traversal of the tree starting at the root adds up the accelerations along each path down the tree to get the acceleration of each point in $O(n)$ time.

At the end of this, asymptotically the most expensive step may be constructing the tree in the first place, which naïvely takes $O(n \log n)$ time. However, various techniques may be applied to speed this up as well.

6 The Matrix Perspective

Another enlightening perspective on the problem can be seen as trying to quickly compute the dense matrix-vector product $A = PM$, where A is the vector of accelerations, M is the vector of masses, and P is matrix filled with terms like $P_{ij} = -(\vec{x}_i - \vec{x}_j)/\|\vec{x}_i - \vec{x}_j\|^3$. Barnes-Hut and variants essentially replace large blocks of P (corresponding to clusters) with rank-1 outer-product approximations which can be multiplied in linear rather than quadratic time.

7 The Fast Multipole Method

Whereas higher accuracy is achieved in Barnes-Hut by tightening the maximum allowed ratio between cluster size and distance, the Fast Multipole Method (FMM) takes a different tack. Instead more terms in the Taylor series are used. In this specific case the terms are called "poles"—the first term is the monopole, the second term a dipole, the third term a quadrapole, and all terms generically multipoles, , following the usage in complex analysis. If enough terms are taken, the error can be driven to below round-off for any two clusters with some finite separation between them. From the matrix perspective, this amounts to using higher rank approximations to blocks—similar to how the SVD can be used to approximate a given matrix (though here the basis vectors involved in the low-rank approximation are determined analytically, not with an eigenproblem).

The FMM further changes things around by getting rid of the tree structure and instead laying down a uniform grid, and using the centre of each grid cell to base the expansion instead of the centre of mass of the points inside. Assuming $O(1)$ points per grid cell, the forces due to points internal to a grid cell or in neighbouring grid cells take $O(n)$ time. The remaining $O(n^2)$ pairs (interactions between separated grid cells) can be accelerated in a different way, exploiting the regularity of the grid and calls to the $O(n \log n)$ Fast Fourier Transform—but fully working this out is beyond the scope of this course.