Astrophysical Exercises: The Stability of Saturn's Rings

Joachim Köppen Strasbourg 2009

1. Physical Background

In 1856, James Clerk Maxwell submitted an prize-winning Essay on the Stability of Saturn's Rings. In this very detailed study he first proved that that the rings could not be of a solid material, which lead him to propose that they consist of a great number of small particles which orbit that planet. He investigated very carefully under which conditions such a configuration would be stable and would thus make the rings a feature that would last long and would not show any change.

His stability analysis showed that if one has a ring of n satellites, evenly spaced around the planet, and orbiting it in the same circular orbit, this system would be stable against perturbations of the relative positions if

$$M_{\mathrm{Saturn}}/M_{\mathrm{Ring}} > 0.435n^2$$

where M_{Ring} is the total mass in the Ring, formed by n satellites of equal mass.

His study was done analytically, and long before we were able to solve this gravitational n-body problem in full detail with electronic computers ... but our computations should confirm his result. And they will!

With modern computers one can easily follow the evolution of system of a large number of particles which move around each other due to the graviational attraction, and thus this approach has become an enormously useful resreach tool. It allows to study not only planets and moons, planetary systems, stellar clusters, and clusters of galaxies, but often one applies these techniques to similar problems, such as the evolution of the clumps that make up an interstellar gas cloud. Also, one can follow the collision of interacting particles and study how galaxies collide, or stars or even subatomic particles. In this exercise, we write a simplified program that computes the movements of a large number of mass points under the influence of their gravitational attraction. So we can simulate the fate of a ring of satellites to check Maxwell's criterion, but also to generalize the model to unequal satellite masses, multiple rings etc...

2. The Equations

We consider a system of n points of mass m_i with positions $\mathbf{r}_i$ and velocity $\mathbf{v}_i$ at time t. Thus the force experienced by the i-th point is the (vector) sum of all the forces exerted by all the other particles:

$$\mathbf{Force}_i = \sum_{k=1}^{n}{}' \frac{G m_i m_k}{d_{ik}^3} \mathbf{r}_{ik} \tag{1}$$

The quote on the summation sign means that we sum over all $k \neq i$, since the mass point does not attract itself. G ist the gravitational constant, and $\mathbf{r}_{ik} = \mathbf{r}_i - \mathbf{r}_k$ is the radius vector from particle i to k, with the distance $d_{ik} = |\mathbf{r}_{ik}|$.

Each mass point experiences an acceleration $\mathbf{a}_i$ due to this force:

$$m_i \mathbf{a}_i = \mathbf{Force}_i \tag{2}$$

which changes the velocity and the position of the particle:

$$\frac{d^2}{dt^2}\mathbf{r}_i = \frac{d}{dt}\mathbf{v}_i = \mathbf{a}_i \tag{3}$$

These are all the equations that rule the movements of each particle, and thus the evolution of the whole n-body system.

2.1 How to Compute it

The equations (1) to (3) are vector equations, so we write them down for each component separately. For simplicity, we use cartesian coordinates $\mathbf{r}_i = (x_i, y_i, z_i)$. This gives for the x-component of the acceleration of the i-th particle:

$$a_{x,i} = \sum_{k=1}^{n}{}' \frac{G m_k}{d_{ik}^3} (x_i - x_k) \tag{4}$$

To do the integration of time, we take a constant time step Δt, during which we assume that the acceleration at the 'old' time t is constant in each component and for each particle. Thus the x-component of the velocity of the i-th particle will change during this time step:

$$v_{x,i}(t + \Delta t) = v_{x,i}(t) + a_{x,i} \Delta t \tag{5}$$

Of course, during this time interval, the acceleration changes, as the mass point flies to another position. So what one really needs for the acceleration is a suitable mean value for each time step. But in order to compute this, we need to know the new position, ... which is what we are just going to compute. So, such a more accurate method (which is called *implicit* since it uses information about the new position) needs an iteration for each time step and it is more complex to program. For the time

being, we shall use the simpler method, but please remember, it is less accurate. It is called an *explicit* method, as it uses only information from the old time.

To compute the new position of the mass point at the new time $t+\Delta t$ we assume that the velocity is constant. However, the same question as before is raised: which velocity shall we use? We have two: the old one $v(t)$, or the new one $v(t+\Delta t)$, and we could even take some average value. We recommend to use the *new* velocities. The advantage of doing this is the following: One can simply show that the change of the angular momentum during a time step will be zero, if we use this mixed method (Please check this by calculating analytically $\Delta\mathbf{L} = \mathbf{r}(t+\Delta t)\,\mathtt{X}\,\mathbf{v}(t+\Delta t) - \mathbf{r}(t)\,\mathtt{X}\,\mathbf{v}(t)$ from Eqs. (5) and (6), and also by observing the components of $\mathbf{L}$ in the numerical calculations). So the new positions are

$$x_i(t+\Delta t) = x_i(t) + v_{x,i}(t+\Delta t)\Delta t \tag{6}$$

Since we now use some information from the new time, the method is implicit. These computations are done for all the points, and after all the new positions are ready, one repeats the operation for the next time time step.....

2.2 Presentation of results

For our study of Maxwell's criterion, we can limit our computations to the orbital plane of the ring, in, say, the (x, y)-plane. This will reduce the number of computations necessary during ech time step, and thus will speed up the program, and will allow us to use a larger number of particles instead!

While a direct plot of the instantaneous positions of the point masses, projected into some plane – say the (x, y)-plane – is helpful to do the first checks with very simple models, and to get experiences with these systems, the plot quickly becomes quite filled with a mass of lines.

Often it is helpful to plot just the distance of each point from the centre-of-mass, whose x-coordinate is:

$$x_0 = \sum_{i=1}^{n} m_i x_i / \sum_{i=1}^{n} m_i \tag{7}$$

The other coordinates are computed analoguously.

2.3 Checks of the Method

Before one applies one's program to complex problems, one must check whether it really does what it was designed to do. The ideal way is to run it for simple problems that have an analytical solution. For example:

1. compute the orbit around the sun of a planet which has circular velocity: take two masses, a big one at the centre ($x = y = z = 0$ and $v_x = v_y = v_z = 0$) and a very small one at ($x = r$, $y = z = 0$) with initial velocity ($v_x = v_z = 0$, $v_y = v_c$). v_c is the velocity of a circular orbit with radius r. The planet should not only make a perfect circle around the sun, and come back to the initial position

(within the orbital period – that you had computed from the basic formulae!), but also keep staying on the circle for as many cycles as possible. The accuracy depends on the time step you had chosen.
2. check other initial velocities. They should give elliptic orbits, and if one exceeds the escape velocity, the planet should never come back!
3. compute the kinetic and potential energies of the whole system, as well as the modulus or the components of the angular momentum vector. Plot them as a function of time. Since we do not feed in any energy or angular momentum, they must stay constant. Do they ???? How well does your program conserve energy and momentum? Dependence on time step.

3. General Remarks, Hints, and Kinks (I know you won't read this!!!)

0. Before actually writing the program, read the whole dossier and try to get the global idea. It may be very usefull to make a flow chart diagram in order to understand the sequence of what is to be computed; a diagram of the program structure and the data structure to find out, how the loops and iterations are nested, which data from earlier parts you need at each section, which kind of vectors and arrays you are going to need. This may seem bureaucratic, boring or even old fashioned, but **don't start typing anything, before you are absolutely clear about what you plan to do**. Otherwise you may really end up wasting much time in trying to find the logical errors, loopholes, and cul-de-sacs of your hasty programming. Save yourself the frustration, disappointment, and anger!
1. General Program Planning: It is a good idea to lay out the program as general as possible. This makes it easier to include other effects, or to try out other situations. Expanding the program to other force laws, e.g. repelling, as for the a cluster of protons instaed of stars, or completely different dependence on distance.
2. Modular Construction: It is also a good idea to break up the program into mathematically or physically sensible units. This allows a better testing of these individual modules — and most of the time is spent in tracing an error — a more flexible use of them for other purposes, and their exchange against improved or alternative methods, improved data, other physical processes, etc. For example, if the time integration is contained in a separate unit, one simply exchanges this against a more sophisticated method, if need arises, but without the trouble of having to change the program at a dozen places. For testing, a simple main program has to be written which supplies the necessary input data to this unit. When adapting the program to a different problem, one merely re-arranges the modules. The main program may then be just a control program to call the subprograms in the particular order, and gets and supplies data from and to each part.
3. Check Everything by Hand: Often, we underestimate our ingenuity to make small logical mistakes or simple typing errors, which may cause faulty results.

The worst kind of mistakes are those which produce results that look as one would expect them to be. Do take the trouble of check everything the program does, until you are sure it does only what you want it to do. In programs about physical things, basic physics must be obeyed: conservation of particles, energy, etc. Also, all the simple and limiting cases which we do understand, must be reproduced accurately.

4. Provide Error Messages and Tracing: Especially during testing, you will print out everything that is useful to judge on what the program does and what decisions it makes. For example, have a print out of the energies and angular momentum of the system. It is recommendable to define one or more variables that can be used to switch on these print-outs. In this way, one can always check every part of the program, even after it has been considered finished. If you later want to try out some modification, and the results are either complete rubbish (because you made some error) or you just want to understand how the solution behaves, this option is very useful. Provide written messages if something goes not normal, e.g. if the automatic stepwidth goes below or above specified values. This become more important as the program grows in size and complexity.
5. Take Time For Comments: Don't be lazy with putting comments into the program. Not only for the general description what each subprogram does, what data it needs, and what variables it changes. But also if you change just a line for a test. Often one forgets after a few days about it, and is quite surprised about the results. Save yourself the panic! Plenty of comments are vital, if you find some time later that you might use it for something, but you can't remember about its inner workings. Don't wait for them after "the program is finished". It never happens, or you won't have the time.
6. Be highly skeptic of anything the program produces.
7. When you are making tests, and later running the program for various situations and parameters, try to keep a careful written record of what you do, noting input parameters and results. This will make it easier for you later to compare results with earlier ones, in case you have to hunt for an error that has crept in yesterday when you "just changed a few things, almost nothing — but the program doesn't work any more".

4. Suggestions

a after verification that the program works as it should and that the orbit of a single satellite around a massive planet remains stable for as many orbits as you can do, build up a system of satellites. Maxwell investigated 36 moons of equal mass, equally spaced in a circle and moving with circular speed. For the beginning, it may be better to start with a smaller number ... in any case: for a given mass ratio of Saturn and its Ring, what happens if you increase the number of particles? Maxwell's criterion tells us that the ring will become unstable, if the number of particles is larger than a certain limit. Let's test that! (at

http://astro.u-strasbg.fr/~ koppen/maxwell/ you will find a JAVA applet which does the same simulation)

b what happens if we deviate from our assumptions? For example, we could place the moons on a slightly elliptical orbits. What happens? Does the criterion still hold? What happens if we make this orbit even more elliptical? Does an increase of ellipticity lead to a higher or lower stability?

c If your program is sufficiently fast, let's do it fully in three dimensions, and let us place the satellites in circular orbits, but each one with a slightly different inclination angle with respect to the plane of the ring. What does that do?

d etcetcetc