Astrophysical Exercises: Precessing Orbits

Joachim Köppen Strasbourg 2011

1. Physical Background

Although the gravity is the weakest of the four fundamental forces, it rules the large scale structure of the Universe, the motions of stars and planets. Newton formulated his universal inverse-square law for the attracting force between two masses, and with his precription we have travelled to the Moon and to other planets. However, if we want to explain the orbits of stars in a spiral galaxy with Newtonian gravity, we run into a problem, because the observed mass inside the orbit does not suffice to keep the stars on their orbits. One way to account for this well-established fact is to postulate the existence of Dark Matter, some kind of matter that has so far escaped all observational efforts, except for its gravitational influence on normal matter. This model has been most successful to explanation many observational facts in the Universe, but we still have no idea what this Dark Matter is, and have no direct and firm evidence for its existence. Another explanation could be that Newton's laws do not hold for the large distances in space, so that the laws derived from experiments in Earth and its vicinity might need modifications. One of the consequences of this approach, MOND for MOdified Newtonian Dynamics, is that orbits no longer follow exactly Kepler's laws, and show a precession.

If one considers the motion of two bodies only, such as that of a planet around its parent star, the exact solution is well known: the trajectory is a conic section, and follows the three laws formulated by Kepler. But this is true only, if the gravitational force depends precisely on square of the distance. As Newton himself already realised, any deviation from the exponent 2 would cause the orbits to precess, that is, the major axis would rotate slowly around the centre of gravity of the system. This means that observations of the precession rate of orbits would allow to measure deviations from Newtonian gravity. However, there are also other efects that cause orbital precession. Apart from relativistic effects (the precession of Mercury's orbit is one of the proofs for the validity of Special Relativity) there are less exotic causes: deviations of the mass distribution in the central body from spehrical symmetry (oblateness of the Sun; the Earth's shape and its uneven mass distribution have their effects on satellite orbits), perturbations by other bodys outside but sufficiently close to the system, tidal forces from the other stars in the galaxy which the system is part of.

With our computers we can easily and accurately compute the orbits, given any prescribed force law. In this exercise, we shall write a simplified program that

simulates the flight of a planet (or a star) around a star under the influence of gravity, but also allows to determine the precession rate. Since the numerical solution of the equations will cause some numerical precession of the results, we have to ensure by choosing an appropriate method and calculational paremeters that this error remains sufficiently small. With this constraint, we shall be able to determine the precession rates for various orbital configurations, different deviations from Newtonian gravity, as well as the MOND formulation. As it is practically impossible for us human to follow a star in its orbit in a galaxy, we cannot test MOND directly. With our simulations, we can explore whether it could be possible to test MOND by high-accuracy observations of binary stars or perhaps tracking interplanetary probes. Let's find out whcih conditions and what accuracy would be required!

2. The Equations

Let us consider the orbit of a planet about a star. Since the star is much more massive than the planet, we may neglect that the planet also acts on the star, and hence both bodies perform orbits about their centre of mass. It makes life easier, if we neglect this, but we do not lose any significant aspect: It can be shown that the treatment of the proper two-body problem can be separated into the motion of the centre of mass and the relative motion of the two bodies...

The star of mass M will thus be placed in the origin of the coordinate system and it will be at rest. At at given instant of time t, the planet of mass m shall be at position $\mathbf{r} = (x, y, z)$ and moving with velocity $\mathbf{v} = (v_x, v_y, v_z)$. The force experienced by the planet is (written component-wise, marked by the indices):

$$F_i = -\frac{GMm}{r^2}\frac{r_i}{r} \tag{1}$$

where $r = |\mathbf{r}| = \sqrt{x^2 + y^2 + z^2}$ and G is the gravitational constant. The first term is Newton's law of gravity for the total force, the second term is just the projection of the force onto the direction to the star.

Here, it is useful to make another simplifying assumption: with any central force law, the motion of the planet is restricted to a plane. Thus, we may solve the equations in two dimensions only, by placing the orbital plane in the (x, y) plane. This means that the computational effort is reduced by 30 percent, and the simulation will run faster!

The planet experiences an acceleration $\mathbf{a}_i$ due to this force:

$$a_i = F_i/m = -\frac{GM}{r^2}\frac{r_i}{r} \tag{2}$$

which changes the velocity and the position of the particle:

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

These are all the equations we need to solve.

2.1 Normalized units

The first thing we notice in the above equations is that the planet's mass m has completely vanished from the final expressions. Also, that GM always appear as this product. This allows us to formulate the problem in normalized or adapted units: if we used m, kg, and s as units, we would get numerically large numbers for the astronomical dimension, masses, and times. This may cause numerical difficulties ... Instead it is better to have the numerical values of all the variables close to unity. If we chose such a unit of mass that $GM = 1$ and as unit of length that the initial distance planet-star becomes $r = 1$, and as unit of time that the speed for a circular orbit at radius 1 also becomes $v_c(r = 1) = 1$, we have a much better numerical situation. As the circumference of a circle of unity radius is 2π, this will be the value of the circular period.

There is another benefit: the general problem of a small body around a very massive one is reduced to this simple view. Irrespective of the real masses and the real dimensions, we get a circular orbit, if the small body has circular speed as its initial speed. This applies to any system: the circular speed for Earth satellites at low alitude is about 7.9 km/s, the circular speed around the Sun is 30 km/s at the place of the Earth, and about 5.5 km/s at Neptune's distance, and about 200 km/s for the Sun in the Milky Way Galaxy.

2.2 How to Compute it

At a given time t we compute the force, and thus the acceleration. 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. Thus the x-component of the velocity of the planet will change during this time step:

$$v_x(t + \Delta t) = v_x(t) + a_x \Delta t \tag{5}$$

Of course, during this time interval, the acceleration changes, as the planet flies to another position. 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 planet's new position 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 x-position is

$$x(t + \Delta t) = x(t) + v_x(t + \Delta t)\Delta t \tag{6}$$

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.....

2.3 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: 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, and recover Kepler's laws. 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. Applications

3.1 How to determine the precession

The usual way to characterise a precessing orbit is to specify the angle (with respect to the central body at the focal point) between the apsides, that is the periastron and apastron, the closest and farthest positions of the planet from the star. For a Kepler ellipse, this angle is exactly 180^o or π.

Rather than testing for the minimum and maximum of the distance r, it is much better to test for the change of the sign of the radial speed. The radial speed is the projection of the velocity vector on the direction to the star: The angle φ between this direction and the x-axis is

$$\varphi = \arctan(x/y) \tag{7}$$

Since the tangent function has a period of only π there is an ambiguity for the other semicircle. Thus, some programming languages offer also a function **atan2**(x, y). Then the radial component of the velocity is

$$v_{\rm rad} = v_x \sin(\varphi) + v_y \cos(\varphi) \tag{8}$$

To find the apsidal points, we simply check whether the sign of $v_{\rm rad}$ has changed between the last and the current time instant. When that happens, we know that it lies somewhere in the past time step. We then determine the position more accurately by linear interpolation between the data, for instance

$$\varphi = \varphi(t) - v_{\rm rad}(t)\frac{\varphi(t-\Delta t) - \varphi(t)}{v_{\rm rad}(t-\Delta t) - v_{\rm rad}(t)} \tag{9}$$

For higher accuracy you can use a quadratic interpolation.

Having obtained the angles φ for successive apastron and periastron, we compute the apsidal angle as the difference.

Since we are more interested in the deviation from Keplerian orbits - which have the apsidal angle π - it makes more sense to compute its difference with π! Thus, we get a quantity that increases with faster precession. It is the precession angle per orbit.

3.2 Numerical Precession

The best way to observe precession is to study elliptical orbits. If you do this, and if you had chosen a timestep not too small - because you curious to have results quickly - you will probably notice that the ellipse precesses visibly.

- what is the precession angle per orbit?
- change the timestep Δt. How does the precession angle per orbit change with timestep?
- How does this change with the eccentricity of the orbit?
- How does the precession relate to the error in the total energy?

You may convince yourself that this precession is not a fault of your programming: Our method conserves the angular momentum with machine accuracy. It does not conserve the total energy with the same precision, but with smaller timesteps, the simulation becomes more accurate, the energy error and also the precession rate decrease. It is simply the finite accuracy of our numerical method.

One might chose a sufficiently small time step, but as this increases the computation time, this might not be a convenient option.

Another approach is to use a more accurate numerical method. For instance, we can modify our simple method by using a halfstep in time:

1. compute forces
2. get new velocity for half time step $v_x = v_x + a_x \Delta t/2$
3. get new positions at full step $x = x + v_x \Delta t$
4. compute forces at new position
5. get new velocity for next half time step $v_x = v_x + a_x \Delta t/2$

Since on an elliptical orbit the speed is maximal close to the star, we could add some automatic adaptation of the time step, so that the change in position and/or in speed during one step remains below a certain tolerance ϵ:

$$\epsilon > \Delta|v| \approx |a|\Delta t$$

gives

$$\Delta t \approx \frac{\epsilon}{|a|}$$

Or we could demand that the relative change $\Delta v/|v|$ in speed should remain below the tolerance... Similar arguments could be applied to the change in position ... There is not "best" solution

Lastly, we could use a different and more sophisticated method: the well-known 4th order Runge-Kutta method seems better, but it does not conserve angular momentum with machine accuracy ... However, there is a symplectic version of RK4. Written here for the x coordinate only:

- compute acceleration(x)
- $sv_1 = b_1\, a_x\, \Delta t$; $x_1 = c_1\, sv_1\, \Delta t$
- compute acceleration(x_1)
- $sv_2 = sv_1 + b_2\, a_x\, \Delta t$; $x_2 = x_1 + c_2\, sv_2\, \Delta t$
- compute acceleration(x_2)
- $sv_3 = sv_2 + b_3\, a_x\, \Delta t$; $x_3 = x_2 + c_3\, sv_3\, \Delta t$
- compute acceleration(x_3)
- $sv_4 = sv_3 + b_4\, a_x\, \Delta t$; $x_4 = x_3 + c_4\, sv_4\, \Delta t$
- result: $vx = sv_4$; $x = x_4$

With these coefficients

$$c_1 = c_4 = (2.0 + 2^{1/3} + 2^{-1/3})/6.0$$

$$c_2 = c_3 = (2.0 - 2^{1/3} - 2^{-1/3})/6.0$$

$$b_1 = b_4 = 0;$$

$$b_2 = 1.0/(2.0 - 2^{1/3})$$

$$b_3 = 1.0/(1.0 - 2^{2/3})$$

In the applet http://astro.u-strasbg.fr/~koppen/MOND2/binary.html the methods Leap Frog, Leap Frog 2nd order, Symplectic 4, as well as some others are implemented.

3.3 Precession by other force exponents

Once you know how you can obtain results with a small error (10^{-7} for the precession per orbit are possible) you can measure the precession rates for force laws which have an exponent somewhat different from the value -2 of Newton's gravitation.

Newton himself derived an expression (cf. Valluri et al. MNRAS 358, 1273 (2005)): For a centripetal force of magnitude $\propto r^{n-3}$ the apsidal angle is $\theta = \pi/\sqrt{n}$ which is valid for nearly circular orbits only.

Try out with weakly elliptical orbits but also with more eccentric orbits. How does the precession rate per orbit depend on the exponent, and how does it depend on orbital eccentricty?

3.4 Precession by MOND

In its simplest form - and this will be sufficient for our exercise - the gravitational acceleration a computed by MOND is related to the newtonian gravitational acceleration g_N by this formula

$$a_N = a * \mu(|a|/a_0) \tag{10}$$

where μ is an interpolating function

$$\mu(x) = \begin{cases} 1 & \text{for } x \gg 1 \\ x & \text{for } x < 1 \end{cases} \tag{11}$$

and a_0 a value for the threshold acceleration, below which MOND modifies the gravitational acceleration. A simple continuously differentiable interpolating function is

$$\mu(x) = \frac{x}{1+x} \tag{12}$$

With this choice, you can solve eqn.(10) for a, the desired acceleration! (you'll get a quadratic equation, of which only one solution is physically sensible ... as $a_0 \to 0$ should converge against the newtonian case!)

From this we get the result that for accelerations much larger than a_0 we recover the standard Newtonian acceleration $a = a_N$, but for acceleration much smaller than a_0 (the "deep MOND regime") we get $a = \sqrt{a_0 * a_N}$.

This means that $|a| = GM/r^2$ changes to $|a| = \sqrt{a_0 GM}/r$ in the deep MOND regime, so that the gravitational attraction drops more slowly with distance ... this is the feature that permits MOND to explain the flat rotation curves of galactic disks in an easy way!

In our scaled system of units the value of a_0 determines in which regime a planet moves:

1. $a_0 = 0$ is the purely Newtonian situation
2. $a_0 \gg 1$ is the purely MOND situation
4. In absolute units one finds that about $a_0 = 10^{-9}$ m/s^2 reproduces the galactic rotation curves.
5. Note that in our normalized units, an acceleration of 1 is the acceleration of a planet on a circular orbit with radius 1.

Run simulations similar to the ones done earlier with Newtonian gravity, but increase the value of a_0. You will find that the greater the value of a_0 the greater is the precession.

Given a certain value of a_0 and a given orbital eccentricity, you will find a precession rate. If we interpret this as a deviation from the inverse square law, what exponent would it correspond? The precession depends on eccentricity in a certain way ... does it so in the same way in MOND as when playing with the force exponent (see above)? In other words: could one misinterpret MOND as deviations of the force exponent from the Newtonian value of -2?

Imagine you could measure the precession of a binary star with some certain accuracy. Even if you specify an accuracy that is not possible with present instruments, let us find out what would be the value of a_0 which produces precessions larger than your assumed detection threshold. We can explore several routes:

1. Which kind of binary systems (period, eccentricity) would be suitable for the detection of precession from values of a_0 as low as the ones needed for galactic rotation curves!
2. Which accuracy (in the precession rate) would be needed to detect MOND?

3.5 Precession by external tidal field

Stars are not alone in the galaxy. Therefore a binary system cannot be considered as a completely isolated system. There is the gravitational force from all components of a galaxy that has an influence on the movement of the stars. This general field of force can be represented by a function smoothly and gently varying with position

$$F_{\rm ext} = F_0 + F_1 x + F_2 x^2 ... \tag{13}$$

For the sake of simplicity, let us consider a variation in tye x coordinate only. The first term has no effect on the shape of the orbit, as a constant force will push both stars and the centre of mass. The second term is the tidal force which is zero at $x = 0$ the position of the massive star. It pulls the planet (i.e. the small star) away from the massive star, everytime the planet is on the 'left' or the 'right' side. A deformation and a precession of the orbit are the consequences.

If you add this term to your simulations, you will be able to determine the precession per orbit as a function of the force gradient F_1, as you increase it starting from the isolated case $F_1 = 0$. If you think in terms of detecting MOND or deviations from Newtonian gravity law by observing precessing binary stars, could the tidal forces due to the galactic gravity field perhaps render such an attempt impossible?

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

0. Before actually writing the program, do make a flow chart diagram in order to understand the sequence of what is 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.
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.
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. If you have a few variables that switch on these print-outs. Then, you can always check every part of the program, even after it has been considered finished. If later you 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 warning messages if something goes abnormal, e.g. if the automatic stepwidth goes below or above specified values. This becomes 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, especially when results look reasonable ... when you get weird results, negative temperatures, NaN, orbital catastropies, then you should be happy: only then you really know that there in an error in the program!
7. When you make tests, and later run 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".